

## Conteúdo

1.	Secure Embedded Logging .....	4
2.	Third party based authentication.....	11
3.	Web of Trust.....	15
4.	Password based key exchange .....	18
5.	Static Integrity Properties .....	23
6.	Dynamic Integrity Properties .....	27
7.	Software Container .....	31
8.	Integrity Protection .....	38
9.	Integrity Attestation (Remote Attestation).....	42
10.	Content-Dependent Authorizer .....	46
11.	Context-Enhanced Authorizer.....	51
12.	History-Based Authentication .....	56
13.	Fine-Grain Access Control .....	66
14.	Rights Based Fine Grain Access Control.....	70
15.	Whitelisting Firewall.....	76
16.	Trust Trap Mitigation .....	81
17.	Secure Boot (Authenticated Boot) .....	88
18.	Credential Renewal .....	94
19.	Multilevel Secure Partitions .....	99
20.	Alarm Monitoring.....	105
21.	Relays .....	109
22.	Access Control to Physical Structures .....	112
23.	Discretionary Access Control.....	117
24.	Mandatory Access Control.....	122
25.	Anonymity Set .....	128
26.	Morphed Representation.....	135
27.	Hidden Metadata .....	141
28.	Layered Encryption.....	149
29.	Informed Consent for Web-based Transactions .....	154
30.	Session-Based Attribute-Based Authorization .....	159
31.	Privacy-Aware Network Client .....	163
32.	Secure Pre-Forking .....	168
33.	Input Guard .....	177
34.	Output Guard .....	184

35.	Fault Container.....	191
36.	Actor-Based Access Rights .....	199
37.	Actor and Role Lifecycle .....	206
38.	Policy Enforcement .....	213
39.	Administrator Object.....	225
40.	Role Validator.....	229
41.	Privilege-Limited Role.....	232
42.	Role Hierarchies .....	235
43.	Risk-Based Authenticator.....	238
44.	Role Based Fine Grain Access Control.....	246
45.	Secure Storage .....	249
46.	Secure GUI System .....	253
47.	Neuralyzer .....	261
48.	Mutual Authentication.....	269
49.	Short Lifespan.....	273
50.	Gridmap Authorization.....	276
51.	Abstract IDS.....	280
52.	Abstract Virtual Private Network .....	286
53.	Asymmetric Encryption .....	291
54.	Authenticator .....	297
55.	Behavior-Based IDS .....	301
56.	Circle of Trust .....	306
57.	Controlled Access Session .....	309
58.	Credential .....	313
59.	Digital Signature with Hashing .....	320
60.	Identity Federation.....	328
61.	Identity Provider.....	334
62.	Layered Operating System Architecture .....	337
63.	Microkernel Operating System Architecture .....	342
64.	Modular Operating System Architecture .....	348
65.	Protected Entry Points .....	352
66.	Protection Rings .....	356
67.	Role-Based Access Control.....	362
68.	Virtual Address Space Structure Selection.....	366
69.	Access Control List.....	372
70.	Administrator Hierarchy.....	377

71.	Capability.....	381
72.	Policy-Based Access Control.....	386
73.	Transport Layer Security .....	392
74.	Multilevel Security.....	401
75.	Security Logger and Auditor.....	404
76.	Session-Based Role-Based Access Control .....	409
77.	Signature-Based IDS .....	413
78.	Symmetric Encryption .....	418
79.	Virtual Machine Operating System Architecture .....	425
80.	Front Door .....	430
81.	Reference Monitor.....	436
82.	Controlled Virtual Address Space.....	439
83.	Controlled Object Factory .....	443
84.	Execution Domain .....	447
85.	Access Control Requirements .....	451
86.	Asset Valuation.....	461
87.	Audit Requirements .....	469
88.	Audit Trails and Logging Requirements .....	477
89.	Authorization.....	485
90.	Automated I&A Design Alternatives .....	489
91.	Controlled Execution Environment .....	497
92.	Biometrics Design Alternatives .....	501
93.	Controlled Object Monitor.....	511
94.	Controlled Process Creator .....	515
95.	Check Point.....	519
96.	Demilitarized Zone .....	527
97.	Enterprise Partner Communication .....	534
98.	Enterprise Security Approaches .....	545
99.	Enterprise Security Services .....	556
100.	I&A Requirements .....	566

# 1. Secure Embedded Logging

## 1.1. Intent

Answering to the security needs by extending the data logging capabilities in the embedded platforms by adding security modules and services. The proposed pattern can be used to add the logging security features together with a design-focused logging solution.

## 1.2. Example

Electric vehicles contain specialized embedded platforms called Battery Management System (BMS), dedicated for control and management of battery cells used to power up the engine and other components [1,2]. Different derivations of BMS exist, with the modular and distributed BMS being more common than the others. Each Battery pack contains several dedicated sensors alongside battery cells [1]. The battery packs are controlled through Battery Cell Controllers (BCC), which are assigned to handle the immediate data control and throughput of these individual packs. A central BMS receives individual battery packs data from the BCCs. This data ranges from the sensor data (e.g., temperature data) to the voltage and current of a particular cell. They are used to extract information like state of charge (SoC) or state of health (SoH) [3]. Based on the data received, BMS can also store and handle error events.

When a battery pack gets depleted, it needs to be replaced. The replaced battery pack can often still be used as an active component for some other appliances, e.g., power grids. Here, battery packs are aimed to be shipped together with their assigned BCCs. In case the BCCs are to remain as part of the vehicle and its BMS, a design compromise needs to be established to enable the logged operational data to be shipped with the battery pack as well. Since BMS would be mass-produced, a design needs to be made in the earlier phases of the development.

It is desirable to enable the porting of the stored memory units together with the battery packs as to be able to track the health of the used battery packs or for any additional data post-processing. It is of critical importance that only valid battery packs are being transported and that it is possible to authenticate the memory units used with the previous battery packs. This constraint is essential to make sure that no malicious attacks through a modified memory unit are possible. Additionally, the data that is stored needs also to be secured. The reason for making it secure is to guard it against any potential malicious attacks or even faults that can arise from oversights during service and maintenance. The constraint is still present to handle these design steps in the initial development phase. This is done for the fact that the battery packs would be mass-produced. Any change that would otherwise be done later could jeopardize the security of the battery packs and add an additional cost.

## 1.3. Context

An embedded platform is being designed which uses local memory devices to handle the storage of lifetime logging data. For the reasons of the cost and memory size limitations, as well as not having, or having limited, access to a wide network, it is intended for the platform to rely on local solutions rather than remote services. Often, these types of systems are closed

and protected under a specific group. It is critical that the stored data maintains its integrity and is only managed through authorized handlers in this environment. The embedded system would consist of a selection of hardware modules, interfaces, and implemented software functions. The hardware modules are divided by their respective tasks and placement. These are usually tied to a specific architecture and their upgrade can be very difficult, or sometimes not even possible.

## 1.4. Problem

How to design an embedded platform that is able to securely handle, but also port and verify, logging data from its source to a designated entity?

In embedded platforms that use distributed module placement, logging process and porting of the logged data often introduce security risks. Porting would need to be done either by using a manual external device or having a connection to a network, both of which might be difficult, and even not feasible, under the platform constraints. Additionally, changes introduced to the system on a physical layer may hamper security during the transfer of the saved data and present a high level of porting complexity. Modules used would need to account for security functionality and have pre-defined elements that supplement them. These considerations result in making it a very challenging and expensive task.

Different malicious attacks can be mounted aimed directly at the content of the logged data during both the active logging period and during the offload transfer period. It is challenging to derive a definite list of threats, as these are usually use-case or application dependent. Here we focus primarily on generic threats that are found in embedded logging systems. Specifically, we consider the following main threats:

- Spying on the targeted process: If not properly secured, an attacker can derive information, and even knowledge, from the stored log data by a direct port access.
- Logged data tampering: Unauthorized change of the current, or previously stored, log content. This includes active attacks on the communication points during the ongoing logging process, but also direct tamper attacks on the devices.
- Counterfeited sources: Each logged data is tied to an affiliated monitored device that is also supplied from a certified manufacturer. During the offload transfer period of the device, i.e., when the change of the targeted monitored device happens, the device can be replaced with a counterfeited or a malicious one. A different attack would be by using the same device but replacing the data inside it.

The following forces apply to the possible solution:

- Streamlined HW/SW integration: Implementation of the hardware and software elements associated with the security functionality need to be easily replicated across multiple devices and vendors.
- Production cost: Changes made to the hardware and software design of the embedded systems can result in an increased manufacturing cost.
- Limited resources: The embedded system needs to be able to execute all necessary functions under different constraints.
- Security - confidentiality and integrity: Necessary measures need to be taken which should prevent the logged data to be tampered or spied on.

- Security - authenticity: The logged data that is stored needs to be able to be properly identified and verified that it comes from a valid source entity. This authenticity is also necessary each time the data needs to be accessed during the active period, i.e., when the data is retrieved for the analytic or other operational purposes

## 1.5. Solution

Ensure that the monitored logged data will be securely protected through an integrated security module relaying data to the memory module and authenticated by using necessary hardware and software critical components embedded during the deployment phase.

When implementing a logging procedure as part of the constrained embedded platform, the security requirement is achieved by integrating a Security Module (SM) as part of the EC. While adding the SM to individual EC devices adds to the overall cost, it does make the system more decentralized. Furthermore, this ensures that the security operations are distributed without heavily impacting the performance. EC device vendors could also not guarantee that the logged data would be secured since the EC itself would not handle that constraint. Therefore, it is necessary to also couple the security operations as part of the EC to appropriately address the security design and attest that the information stored will be protected.

## 1.6. Structure

Figure 1 depicts the design behind the solution and shows the recommended building blocks. The following components are listed:

- Embedded Controller (EC): Contains necessary interfaces for the communication, main driver logic, and the control bridge between the data that is to be stored and the security driver.
- Logging Memory Unit (LMU): Dedicated device for storing the encrypted data; contains necessary description data, encrypted security keys, and the encrypted data.
- Security Module (SM): Provides security operations; works as a security bridge between the EC and the LMU.
- Source Verification Device (SVD): Device tied to a particular LMU and used for the authentication purpose; can contain necessary authentication data, i.e., private-public key pair, and/or a certificate. It is also generally seen as the device from which logging process data is retrieved (data source).

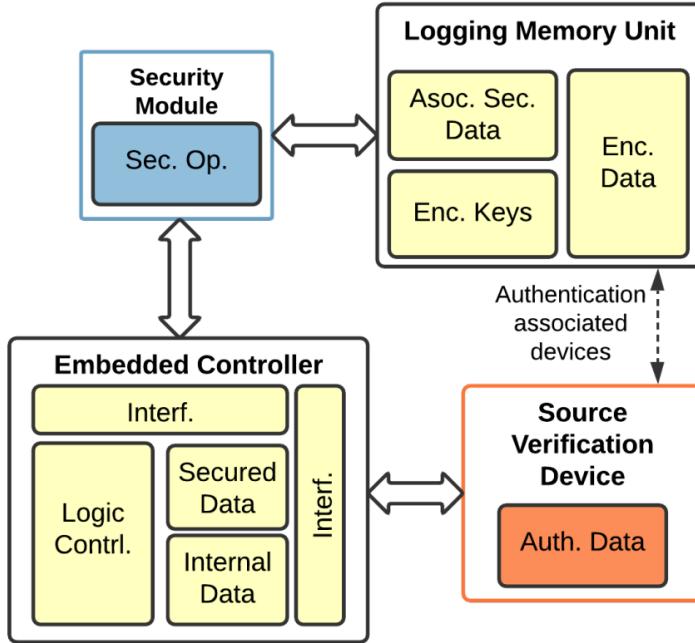


Figure 1: Design-based solution in task separation for handling security logging by providing secure operations and device authentication.

Software functions and associated security data would be handled by the SM itself. At the same time, it would use the logic controller of the EC to drive the overall processing and data preparation when storing it as part of the logging. It is necessary to keep the SM cheap in design. As a minimum-security requirement when storing the logging data, we propose to use encryption and authentication for the stored data. These can either be achieved by using separate security functions or applying a suite like Authenticated Encryption (AE) to handle this process. The data integrity check (additional AE data or a separate operation) would be saved in a separate memory block inside the LMU. These, however, do not need to be secured, but they do need to be checked by the EC from the SM each time a new LMU is authenticated. They also need to be periodically updated from the SM after new data is written. The SM should also offer the functionality of storing and handling the key data used by the security operations. Additionally, an EC together with its SM could also provide a Key Derivation Function (KDF). The basic principle of deriving and delivering the keys between the parties is left to the designers. The keys are generally securely encrypted and stored in the LMU secure section. The authentication operations can either be managed using symmetric-based authentication, e.g., AES challenge/response mechanism or by using asymmetric authentication, e.g., Public Key Infrastructure (PKI). The security operations can be handled entirely through software or be hardware-derived, where the hardware operations usually offer better performance, e.g., hardware implementation of the Advanced Encryption Standard (AES). While we consider using the integrated security engine through a dedicated SM as the most cost-effective solution, other dedicated hardware security components can also be examined. These include Secure Elements (SE) and Trusted Platform Module (TPM). However, unlike the integrated secure engine, SE and TPM are more complex to incorporate and much more costly.

## 1.7. Dynamics

The pattern is aimed at providing an affordable and secure solution when transporting and then replacing an LMU.

This process is depicted in Figure 2. Here, a user would receive the LMU together with the SVD from a previous socket. When integrating it into the new system, it might be necessary to verify this memory unit alongside the newly installed SVD, which has been formerly taken out from the older system. This is achieved by using the previously explained security verification functions that the new EC, through its design with SM, would possess as well. The verification process needs to be successfully completed for the LMU to be further used, be it just for the analytic or for continuing operations.

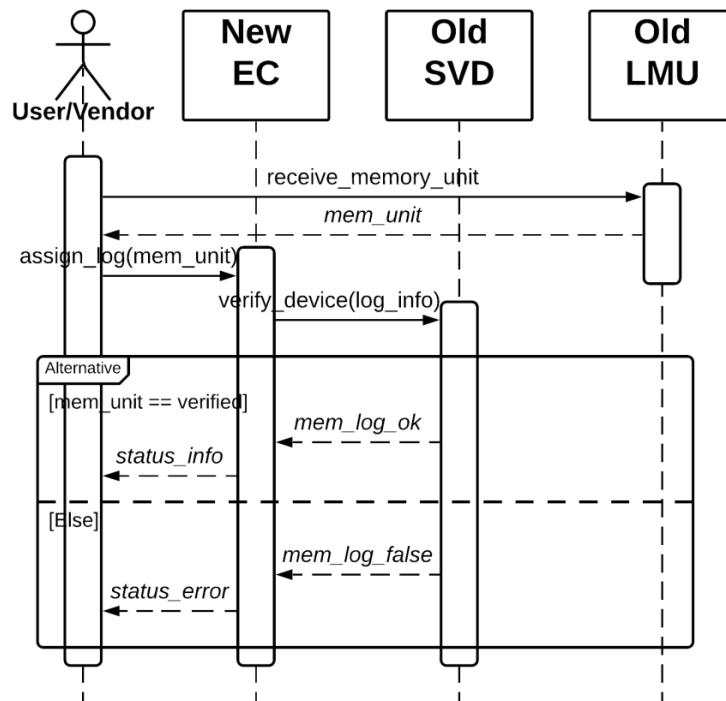


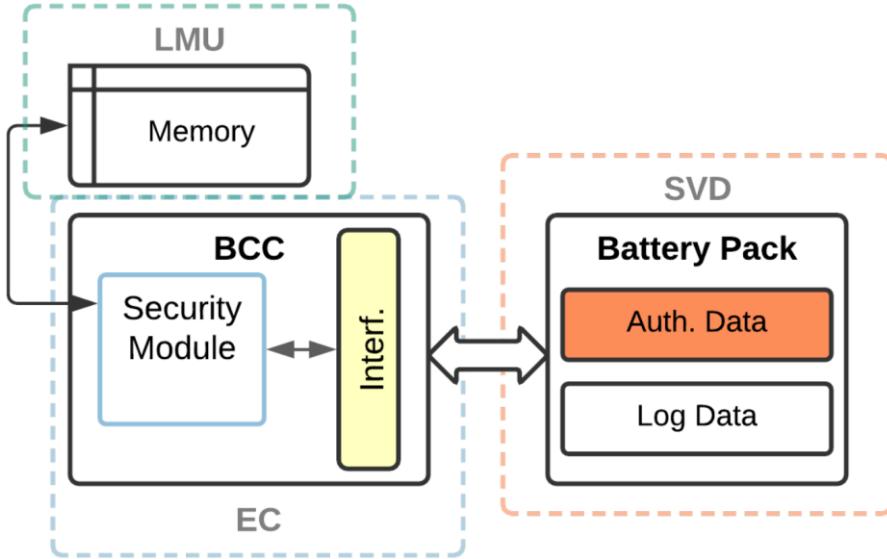
Figure 2: Sequence diagram describing the verification process during the porting of LMU and a SVD from a previous to a new embedded device platform.

## 1.8. Implementation

### 1.9. Example Resolved

Based on the open design question presented in Section 1.2, we present a solution in form of a module extension. Here, security is applied to guarantee: (i) confidentiality - protecting necessary system data by only providing data associated to the BMS operational cycle, (ii) repudiation – an action can be tied to the entity that caused it, and (iii) integrity – data has not been modified. The resulted block design is shown in Figure 3.

The security functionality is controlled by an internal SM service engine. The SM communicates directly with the EC and the memory interface.



*Figure 3: Realized example based on the Secure Embedded Logging pattern. Applied on a BCC of a BMS by extending its applicability for the secure logging process*

When the need arises for a battery pack to be replaced, using this design, it is possible to also port the whole BCC easily or just the memory module with the logged data, as indicated in the pattern solution. Based on the implementation, the BMS would verify the newly added BCCs, while a BCC can independently run the verification operation on the connected battery pack and memory module.

## 1.10. Consequences

This section lists the benefits and liabilities found when applying the SEL architectural pattern based on the Forces from Section 1.4. The benefits of the SEL pattern are:

- As the pattern suggests using a dedicated security module and predefined security functions per EC, the general production design can be applied on a larger scale.
- The pattern proposes the use of a dedicated SM that should allow, at a minimum, encryption and integrity check for handling the security of the stored data.
- Additionally, the SM needs to allow for a method of authentication and verification of individual memory modules that were previously tied to a specific pair of EC and MED.

The liabilities when using the SEL pattern are:

- As long as the security logging is only handled in a closed local embedded platform, further system updates and configurations are not handled with the proposed pattern.
- Each device in the suggested embedded platform is handled as a separate unit, meaning that each embedded controller comes with their own security module. This advantage at flexibility comes also with a drawback, and that is the increase of the general production cost.
- Many embedded devices today are limited in terms of the extension capabilities, i.e., either not containing their own security modules or not providing additional interfaces.

## **1.11. Known Uses**

## **1.12. See Also**

A pattern that is similar in design but different in intent and context would be the Security Logger and Auditor. It is focused on logging security-sensitive actions from different users. Hence, this pattern offers a security solution in tying recorded information with the particular users of a system on an architectural level. Another similar pattern would be the Secure Logger which is traditionally used for capturing targeted application events [4]. It is an implementation design pattern that can be applied on the software level in situations where otherwise system constraints are of no concern and are not taken into the design consideration.

## **1.13. References**

- [1] Andrea, D. (2010). *Battery management systems for large lithium-ion battery packs*. Artech house.
- [2] Deng, J., Li, K., Laverty, D., Deng, W., & Xue, Y. (2014). Li-ion battery management system for electric vehicles-a practical guide. In *Intelligent Computing in Smart Grid and Electrical Vehicles* (pp. 32-44). Springer, Berlin, Heidelberg.
- [3] Xiong, R., Cao, J., Yu, Q., He, H., & Sun, F. (2017). Critical review on the battery state of charge estimation methods for electric vehicles. *Ieee Access*, 6, 1832-1843.
- [4] Steel, C., & Nagappan, R. (2006). *Core Security Patterns: Best Practices and Strategies for J2EE", Web Services, and Identity Management*. Pearson Education India.

## **1.14. Sources**

- Basic, F., Steger, C., & Kofler, R. (2021, July). Embedded Platform Patterns for Distributed and Secure Logging. In *26th European Conference on Pattern Languages of Programs* (pp. 1-9).

## **2. Third party based authentication**

### **2.1. Intent**

### **2.2. Example**

One example could be that a company X is developing a server hosting a web-shop. If a customer wants to buy something from the shop, he needs to send his credit card information to the server. The customer is not interested in sending his credit card information in plain and hence a secure channel has to be established. Further, the customer is not interested in starting a symmetric key exchange process via a second secure channel (i.e. a key shipment using trusted couriers), because he wants to perform the operation now and not after some days when he finally received the symmetric key.

### **2.3. Context**

A system wants to establish a secure channel with another party via an insecure channel. Both communication parties are not limited in terms of computational power or resources (e.g. a personal computer / smart phone communicates with a server), but may not know each other in advance. A MITM can eavesdrop or change every sent message package. Further the MITM is able to resent previously eavesdropped packages (replay attack).

### **2.4. Problem**

Since both communication parties are not known to each other (i.e. no shared secret knowledge, nor a known public key) a mechanism is required such that trust is gained. No secret data shall be exposed to a third party listening or intercepting the communication. No previously recorded message package should trigger an unwanted action (e.g. a payment transaction).

The following forces need to be considered:

- User experience: The security controls should not negatively influence the functional behaviour (e.g. real-time capabilities) of the system.
- Computational complexity: Public key cryptography is more time consuming compared with symmetric cryptography.
- Eavesdropping: Each sent communication package may be eavesdropped or replaced by a malicious third party.

### **2.5. Solution**

Use public key cryptography based on certificates for authenticating each communication participant (abbreviated as Alice and Bob in the following). After successful authentication, a session key is generated to secure the further communication.

The solution of this problem is to have a trusted third party who proves the authenticity of each participant. This is usually called a Certificate Authority which proves the identity of the communication parties and signs their public keys.

## 2.6. Structure

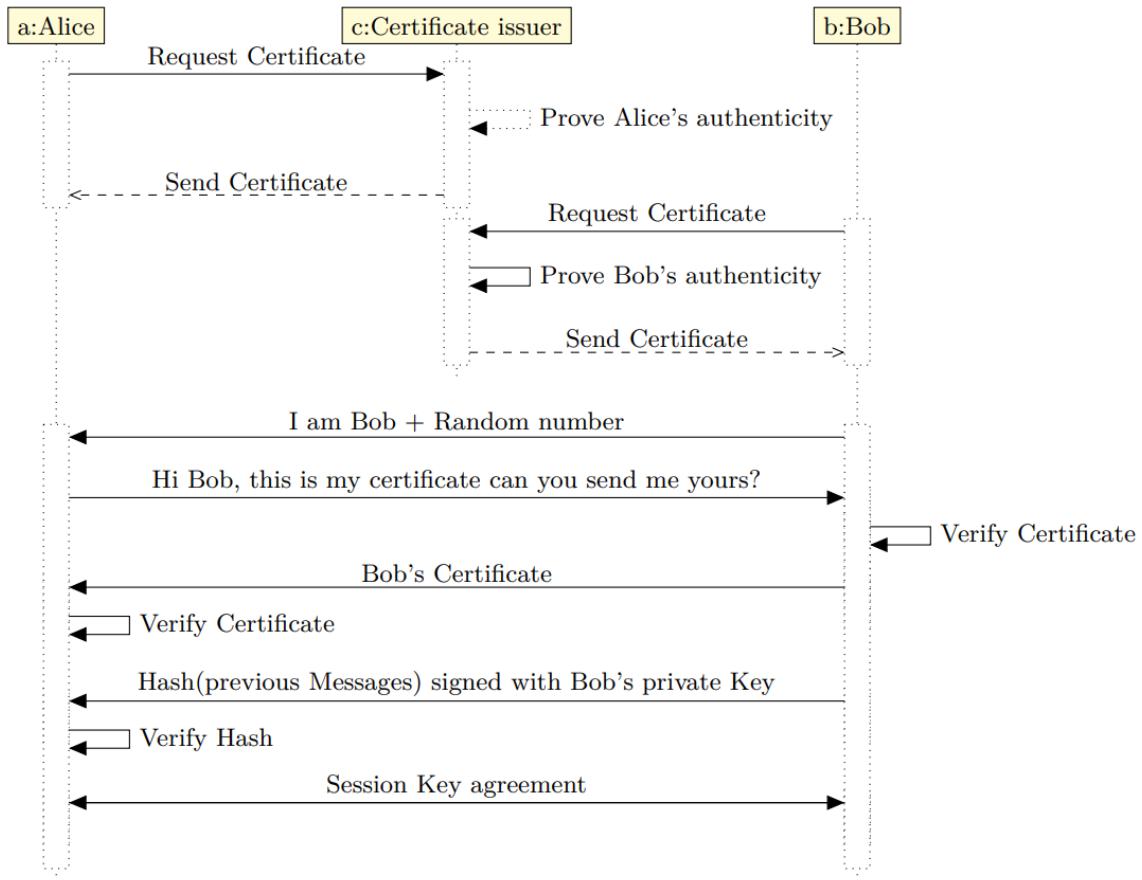
## 2.7. Dynamics

The principle sequence of steps for certificate based secure channel establishment is illustrated in Figure 4 and described in the following:

1. Before a communication is possible, Alice and Bob need to request a certificate issued by a trusted third party (CA). The CA proves the identity of Alice and Bob (e.g. by verifying the passport) and issues the certificate (i.e. signs the public key of Alice and Bob).
2. To start the communication, Bob sends a message to Alice including a random number for the authentication process. A time stamp should be incorporated to harden the system against replay attacks.
3. Alice answers the request with her certificate. Bob extracts the public key of Alice and verifies the certificate by using the public key of the CA.
4. If the verification was successful, Bob sends his certificate to Alice such that she can verify the properness of Bob's certificate and to retrieve his public key.
5. Until that point, no encryption is needed at all since the certificates are public anyway. In principle, these steps could have been performed by a malicious third party which has Bob's certificate. Thus, Bob sends the hash of all previous messages which is signed with his private key so that Alice can prove that Bob has the corresponding private key of the sent certificate. This is important since a malicious third party – which only owns the certificate – cannot compute the signature of the hash. As a result, Alice will terminate the communication if she cannot verify the signature.
6. Both parties need to agree on a session key to encrypt all future messages. This can be achieved by using dedicated key derivation functions (e.g. by using a Diffie-Hellman key agreement where a symmetric session key is generated based on the used key-pairs [Rescorla 1999]) or by randomly generating a new key. This generated key has to be sent encrypted to Alice / Bob using their according public key. Usually symmetric session keys are chosen due to performance reasons.

The presented solution covers the case that Bob is authenticated to Alice. For some applications this may not be sufficient. To ensure a mutual authentication, Bob has to send a challenge to Alice. In the simple case, this can be achieved by repeating step 5 where Alice is calculating the hash and Bob is verifying the signature of the message.

Each certificate has a specific date at which it gets automatically invalid. In some cases, it may happen that the owner of the certificate or the CA want to revoke the certificate earlier (i.e. the private key was broken / leaked, it was improperly issued, etc.). Therefore, so called revocation lists were introduced containing a list of invalid certificates. To ensure that such certificates are not used, each communication participant needs to consult the CA to retrieve this list. Thus, an online connection is necessary to regularly update these lists.



*Figure 4: Certificate based authentication*

## 2.8. Implementation

## 2.9. Example Resolved

## 2.10. Consequences

The benefits of the pattern are:

- Through the use of certificates, Alice and Bob can assure that they are talking to each other and not to a Man-In-The-Middle.
- There are open-source libraries supporting such authentication processes (e.g. OpenSSL)

The liabilities are:

- The Certificates are based on public key cryptography which can be time-consuming when operated on small embedded systems.

- If the Man-In-The-Middle is cooperating with the Certificate issuer, a Man-In-The-Middle attack cannot be detected / prevented, since he is able to manipulate the certificates of Alice and Bob. Such "hacks" cannot be prevented since the CA deals as a root of trust.
- Requires an online connection or a frequent update of revocation lists to ensure that no blacklisted certificate is accepted as valid.

## 2.11. Known Uses

- Secure Sockets Layer (SSL), Transport Layer Security (TLS) [1]. This system uses the process illustrated in Figure 4 using symmetric session keys.
- Secure key agreement algorithms like the extended Diffie-Hellman key exchange [2]. This process implements the presented pattern with a bi-directional challenge response protocol to ensure that Alice and Bob are the owner of the corresponding private keys.

## 2.12. See Also

The "Session Key Exchange with Server-side Certificate" or the "Session Key Exchange With Certificates" pattern are similar implementations of this pattern, where the trusted third party is not a CA but any other common third party which knows Alice and Bob in advance.

The "Session Key Exchange with Public Keys" pattern presents a similar solution, but does not address the issue when Alice and Bob are both unknown to each other. Thus, an eavesdropper could create a self-signed certificate to claim the identity of Alice or Bob.

The "Asymmetric Encryption" pattern is used multiple times in the presented pattern: The signature operation performed by Bob and / or Alice is an asymmetric encryption operation using the private key and a specific message. During the session key agreement asymmetric encryption can be used to send the session key.

The "Symmetric Encryption" pattern can be used in case of symmetric session keys.

## 2.13. References

- [1] Das, M. L., & Samdaria, N. (2014). On the security of SSL/TLS-enabled applications. *Applied Computing and informatics*, 10(1-2), 68-81.
- [2] Yooni, E. J., & Yoo, K. Y. (2010, June). A new elliptic curve diffie-hellman two-party key agreement protocol. In *2010 7th International Conference on Service Systems and Service Management* (pp. 1-4). IEEE.

## 2.14. Sources

- Sinnhofer, A. D., Oppermann, F. J., Potzmader, K., Orthacker, C., Steger, C., & Kreiner, C. (2016, July). Patterns to establish a secure communication channel. In Proceedings of the 21st European Conference on Pattern Languages of Programs (pp. 1-21).

## **3. Web of Trust**

### **3.1. Intent**

### **3.2. Example**

### **3.3. Context**

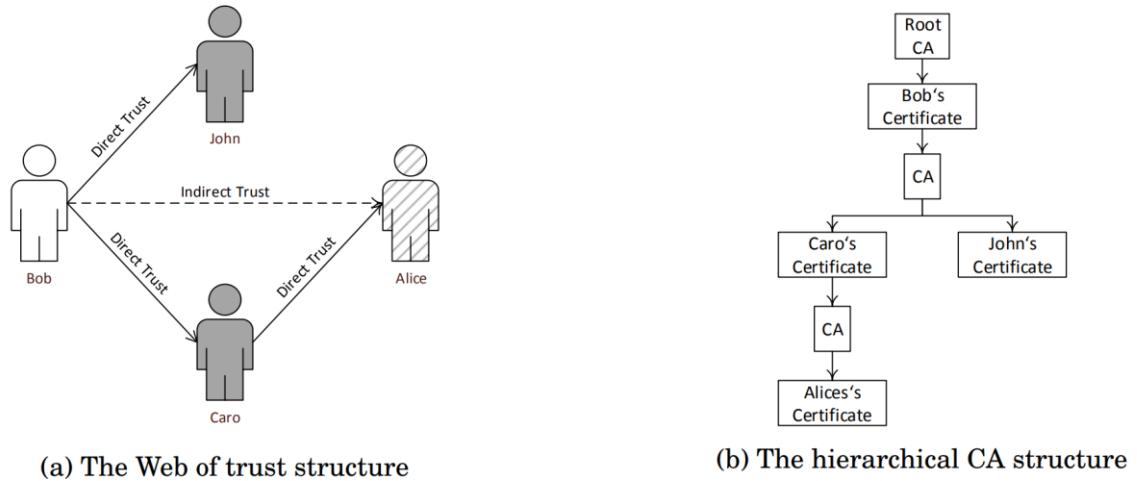
### **3.4. Problem**

In opposite to the Third party-based authentication pattern, Alice and or Bob do not want to trust a single third party.

### **3.5. Solution**

Trust in a key is gained by manually verifying the fingerprint. Thus, no central CA is necessary to prove the individuals identity.

"Web of Trust" is a term which is used to describe a certificate based system in which not a single authority (CA) is used to prove that a specific public key belongs to a specific person, but the community. Each signature can be marked with a confidence level which indicates how the authenticity of the key owner was proved. The basic concept is illustrated in Figure 5a. As illustrated, Bob trusts John and Caro directly since he personally checked the identity of them (e.g. via a face to face meeting). Further, Bob indirectly trusts Alice since she is trusted by Caro. In difference to the CA structure John does not trust Alice because there is no direct path of trust leading to her. This is different in the CA case illustrated in Figure 5b. John will always trust Alice because they have the same root CA.



*Figure 5: Differences of the CA structure compared to the web of trust structure*

### 3.6. Structure

### 3.7. Dynamics

### 3.8. Implementation

### 3.9. Example Resolved

### 3.10. Consequences

The benefits of the pattern are:

- Alice and Bob can assure that they are talking to each other and not to a Man-In-The-Middle.
- No single entity is used to prove the identity of a specific person / system.

The liabilities are:

- The usability is reduced since every communication party needs to prove the identity of the other parties. This is usually done during a face-to-face meeting.

### 3.11. Known Uses

- Pretty Good Privacy (PGP). Trust is managed by the individual communication participants by manually verifying the fingerprints of received public keys / certificates.

The user can decide if a key is fully trusted, partially trusted, or untrusted. These levels can be used to define the behavior of "indirect trust". This means, that a key which is signed by "n" fully trusted parties or "m" partially trusted parties may be accepted as trusted automatically.

- GNU Privacy Guard (GnuPG). A free and open-source implementation of PGP.

### **3.12. See Also**

The presented pattern makes use of the "Session Key Exchange with Public Keys" pattern in combination with a face-to-face meeting to verify the fingerprint of the public keys. Thus, a MITM attack can be prevented.

The "Symmetric Encryption" pattern can be used in case of symmetric session keys.

### **3.13. References**

Sinnhofer, A. D., Oppermann, F. J., Potzmader, K., Orthacker, C., Steger, C., & Kreiner, C. (2016, July). Patterns to establish a secure communication channel. In Proceedings of the 21st European Conference on Pattern Languages of Programs (pp. 1-21).

### **3.14. Sources**

## **4. Password based key exchange**

### **4.1. Intent**

### **4.2. Example**

One example could be a server which is operated by multiple users and need to be accessible from anywhere at any time. This requires that the operators may access the server from their private devices (like computers or smart phones) or via shared public available infrastructure (like public access points or internet coffee shops).

### **4.3. Context**

A system wants to establish a secure channel with another party via an insecure channel. The connectivity of the device is limited such that it is not possible to securely load or store long private keys on the device (i.e. a user needs to manually enter it).

### **4.4. Problem**

A user needs to manually enter a password to start the communication. In a realistic scenario, such passwords are "short" and designed to be easy to remember. Using only these passwords would result in an insecure system since dictionary attacks / brute force attacks could be performed by a MITM to break the password. Further the MITM is able to resent previously eavesdropped packages (replay attack).

The following forces need to be considered:

- Limited access: Symmetric key files, certificates or "Web of Trust" are not suitable since it is not possible to securely load / store a private key file on the device.
- User experience: The security controls should not negatively influence the functional behaviour of the system.
- Eavesdropping: Each sent communication package may be eavesdropped or replaced by a malicious third party

### **4.5. Solution**

Both communication parties (abbreviated as Alice and Bob) need to share a secret password. This password is used to protect randomly generated session key pairs which are used to further secure the communication.

### **4.6. Structure**

## 4.7. Dynamics

Since Alice or Bob wants to transmit high confidential data using a weak (easy to remember) symmetric key they need to issue a password based key agreement process. Asymmetric cryptography has been proven in that context [1,2,3]. Thus, Alice and Bob need to agree on a shared secret which is incorporated into the key agreement protocol such that the used public key scheme is authenticated using this shared secret. The principle sequence of steps is displayed in Figure 6.

1. Before starting a secure communication, Bob is generating a new random key pair (Public Key: PB; Private Key: PprivB ).
2. To start a communication, Bob sends a message to Alice containing the newly generated public key which is encrypted using the shared secret password.
3. Alice receives the message and generates a new random key pair (Public Key: PA; Private Key: PprivA ). The fact that Alice and Bob are both generating new key pairs is essential to increase the overall strength of the protocol. By doing so, it is getting more difficult to achieve offline brute-force attacks (see [2]).
4. Alice retrieves Bob's public key PB by decrypting the received message with the shared secret. Alice encrypts her generated public key PA with the retrieved public key PB (abbreviated as C). Further, the resulting cryptogram is encrypted with the shared secret password before sent to Bob.
5. Bob can retrieve PA by first decrypting the received message with the shared secret and by decrypting the result with his private key PprivB . After that, Alice and Bob are able to start a secure communication based on the generated key pairs. For large amount of data it is advisable to start a session key agreement to establish a strong symmetric key (i.e. AES-256).
6. The next steps illustrated in Figure 6 describes a basic challenge-response protocol to verify the validity of the generated cryptographic keys. This is done in most cryptographic protocols to ensure that Alice and Bob are able to decrypt messages which were encrypted with the according public key PA/PB.
  - a. Bob generates a random number R1 which he sends encrypted with the retrieved public key PA to Alice.
  - b. Alice retrieves the generated random number R1 by decrypting the received message. She generates a new random number R2 which she concatenates with R1. Alice uses the public key PB to encrypt the concatenated random numbers and sends the cryptogram to Bob.
  - c. Bob decrypts the received message and verifies that R1 is part of the received message. Further he retrieves the random number R2 and sends it encrypted with PA back to Alice.
  - d. Alice verifies the correctness of R2
7. Both parties need to agree on a session key to encrypt all future messages. This can be achieved by using dedicated key derivation functions (e.g. by using a Diffie-Hellman key agreement where a symmetric session key is generated based on the used key-pairs [4]) or by randomly generating a new key. This generated key has to be sent encrypted to Alice / Bob using their according public key. Usually symmetric session keys are chosen due to performance reasons.

The illustrated sequence used two times the shared secret password to encrypt a message (encryption of PB and C). In most cases, one of these encryption operations can or should be

omitted. As stated in [2], the most obvious reason to skip one of the encryption operation is, that the message which shall be encrypted has to be indistinguishable from a random number. Otherwise an attacker has knowledge about the principle structure of the plain text. Thus, an attacker could perform more sophisticated attacks against the shared secret password to break it.

The problem which remains is how to choose or how to establish the initial shared secret. This needs to be done using a different channel which can either be considered as secure, or the way the data is sent via this channel can be considered as being secure. For example, a password based secret can be used which is sent to Bob in a secure way. This could be achieved via a key shipment split on multiple key shares using trusted couriers (see [5,6,7] for recommendations).

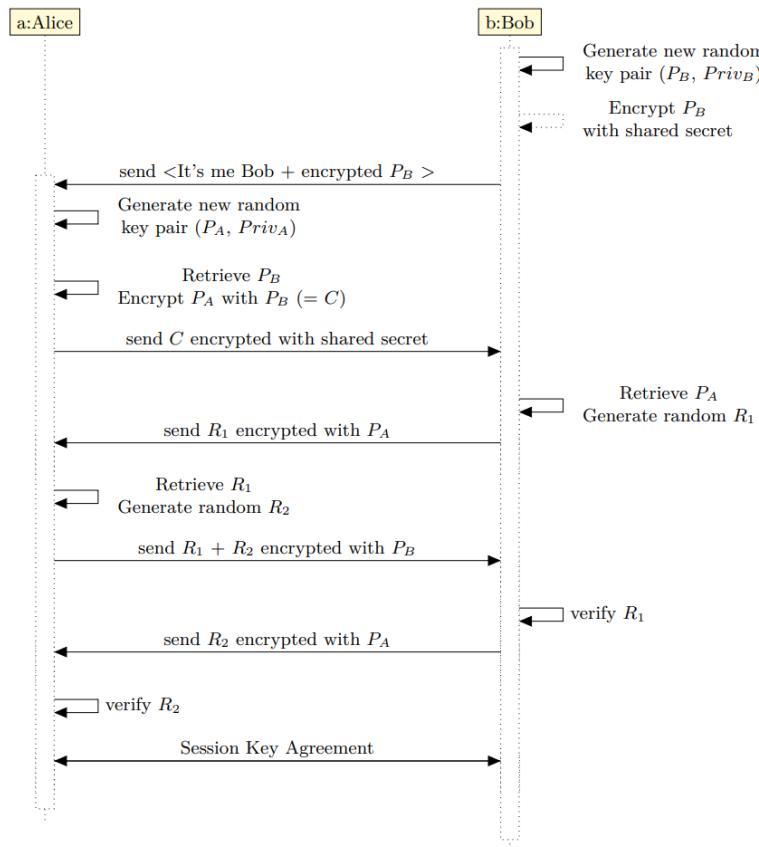


Figure 6: Shared secret authenticated public key establishment.

## 4.8. Implementation

## 4.9. Example Resolved

## 4.10. Consequences

The benefits of the pattern are:

- Using a "weak" shared secret produces a good channel protection against a Man-In-The-Middle attack since an attacker can only hack the system by brute-forcing the randomly generated public key (which should be of reasonable length).
- I do not need a third party claiming the authenticity of a communication partner.

The liabilities are:

- Establishing a shared secret without using certificate based public key cryptography is hard.
- Both parties need to agree on another channel via which the shared secret is established which reduces the overall usability of this pattern.

## 4.11. Known Uses

Use-Cases are mainly found in the domain of password-authenticated key agreements like:

- Encrypted Key Exchange protocols (e.g. [1,2,3]). These are based on the process illustrated in Figure 6 but provides slightly adaptations for different public key schemes (like RSA, ECC). The reason for this is to reduce the complexity of the overall process due to security properties of the different schemes.
- Dragonfly (see [8]). This protocol uses EC based cryptography in combination with a pseudo-random key derivation function to diversify the shared secret at the beginning of each session.

## 4.12. See Also

The presented pattern makes use of the "Session Key Exchange with Public Keys" pattern in combination with shared secret knowledge in form of a password. The shared knowledge is required to ensure that Alice and Bob are who they claim to be by proofing their knowledge about the shared secret.

The "Asymmetric Encryption" pattern is used to protect the messages sent between Alice and Bob to establish a secure channel. Further, it can be used for the session key agreement.

The "Symmetric Encryption" pattern can be used in case of symmetric session keys.

## 4.13. References

- [1] Bellovin, S. M., & Merritt, M. (1993). *U.S. Patent No. 5,241,599*. Washington, DC: U.S. Patent and Trademark Office.
- [2] Bellovin, S. M., & Merritt, M. (1993, December). Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *Proceedings of the 1st ACM Conference on Computer and Communications Security* (pp. 244-250).
- [3] Bellovin, S. M., & Merritt, M. J. (1995). *U.S. Patent No. 5,440,635*. Washington, DC: U.S. Patent and Trademark Office.
- [4] Rescorla, E. (1999). Diffie-hellman key agreement method.

- [5] NIST. (2016). *Special Publication 800-57-1: Recommendation for key management: Part 1: General*. Gaithersburg, MD, USA: National Institute of Standards and Technology, Technology Administration.
- [6] Industry, P. C. (2016). Data Security Standard: Requirements and security assessment procedures. *Version 3.2*, 1.
- [7] International Organization for Standardization (ISO). (2006). ISO/IEC 11770-4: Information Technology - Security Techniques - Key Management - Part 4: Mechanisms Based On Weak Secrets.
- [8] Harkins, D., & Zorn, G. (2010). *Extensible authentication protocol (EAP) authentication using only a password* (pp. 1-40). RFC 5931, August.

#### **4.14. Sources**

Sinnhofer, A. D., Oppermann, F. J., Potzmader, K., Orthacker, C., Steger, C., & Kreiner, C. (2016, July). Patterns to establish a secure communication channel. In Proceedings of the 21st European Conference on Pattern Languages of Programs (pp. 1-21).

# 5. Static Integrity Properties

## 5.1. Intent

The intent of STATIC INTEGRITY PROPERTIES is to reflect the integrity of a system by detection of changes in static system parts.

## 5.1. Example

## 5.2. Context

A system consists of multiple interacting components. Therefore, the occurrence of different types of faults is very likely. Faults may be of natural (e.g., hardware faults due to external events or material wear out) or human origin. Human-made faults can be non-malicious (e.g., accidental) or malicious (i.e., a security violation). All types of faults can result in an unintended manipulation of a component or data. Additionally, parts of the system may be constrained in terms of energy or computing resources.

## 5.3. Problem

How to detect whether a component would violate your system's integrity in case it would be executed? How to detect whether the usage of specific data value (i.e., executing a computation based on this value) would violate the system's integrity?

The following forces apply:

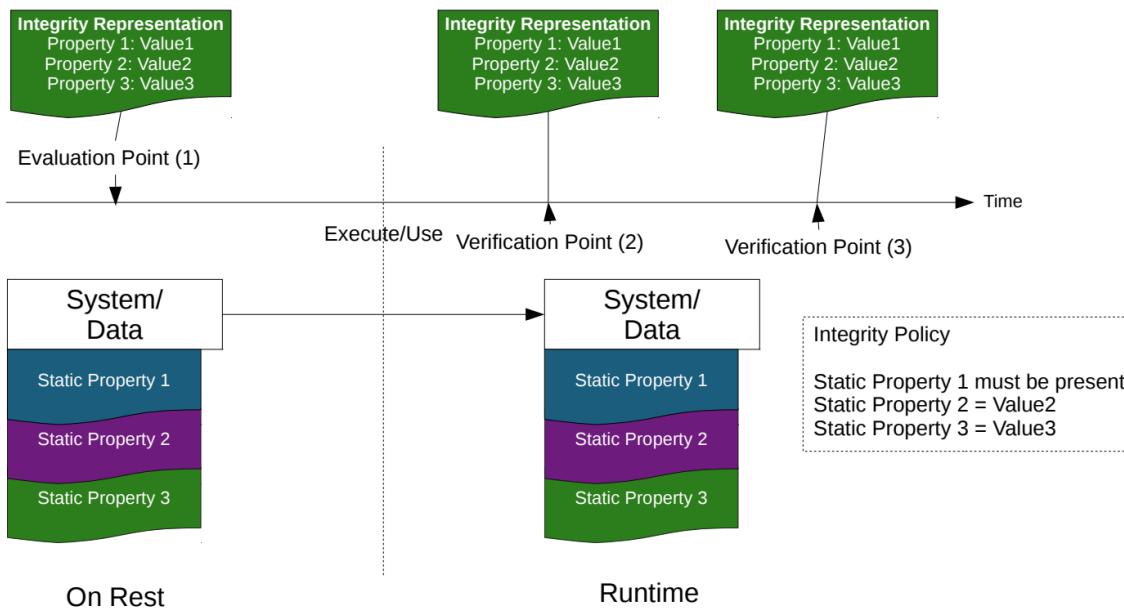
- Component Integrity: The system's integrity would be violated when the integrity of the component that is going to be executed or data value that is going to be used is violated.
- Integrity Properties: Integrity of a component or data cannot be computed directly.
- Offline Identification: Using components or data with violated integrity compromise the overall systems integrity. Such components should not be executed in the first place and low-integrity data must not be used since they would also lower the integrity of the using component. The integrity thus has to be ensured prior to any use.
- Performance Constraints: The system has strict resource constraints at run-time and additional overhead should be minimized.

## 5.4. Solution

An integrity representation of the component is generated before it is executed. Integrity cannot be computed directly, therefore other properties that reflect the integrity of the component are used. Since the evaluation is done prior the execution of the component, the properties have to be static. This means that these properties must not change during run-time and must not reflect any behavioral aspect of the component (behavior can only be examined during execution). The properties have to be identifiable and verifiable. This means

that it has to be possible to deterministically compute these properties with the given data or component as input. Moreover, it has to be possible to formulate a policy that enables a decision concerning the component's integrity based on the presence, absence or values of these properties. The properties have to be examinable before the component is executed.

As illustrated in Figure 7, these properties do not change during run-time. Therefore, only one point of evaluation (1) is needed to gain a representation of the system's integrity. The integrity representation is a list of properties and their corresponding values. At some point in time (before execution or during execution), the integrity representation is verified (2, 3). The verification is done by comparing the values of the integrity representation with an integrity policy that defines allowed values. During run-time, no re-evaluation is needed since the properties do not change anymore. Therefore, no additional computing overhead for evaluation is needed. However, the verification of the integrity can be done multiple times by different entities.



*Figure 7: Static properties are identified before a component or data is actually used and do not change over time. Therefore, the evaluated integrity representation is valid during the whole time of execution or use and there is no need for re-evaluation for each verification.*

## 5.5. Structure

## 5.6. Dynamics

## 5.7. Implementation

For data integrity, often CHECKSUM [1] is used for communication channels. A checksum represents the computable property that has to match a reference value. The policy is generated by the data source (i.e., a reference checksum is generated). Another component, which is using the data for computation (i.e., the data sink), generates a new checksum (the

integrity representation). Before it uses the data value, the sink verifies the integrity of the value by comparing the two checksums. Similar results (but with strong properties regarding security) are achieved with “DIGITAL SIGNATURE WITH HASHING” pattern.

## 5.8. Example Resolved

## 5.9. Consequences

The benefits of the pattern are:

- Component Integrity: The integrity of the component is reflected by the integrity representation that contains static integrity properties.
- Integrity Properties: Assuming a properly crafted policy, the identified properties reflect the integrity state of the components and data and therefore of the overall system.
- Offline Identification: The selected properties can be computed before execution and remain during run-time. Therefore, the computation of the properties can be done safely once, prior run-time.
- Performance Constraints: Since the properties do not change during run-time, re-evaluation is not necessary and therefore the identification of these properties do not cause any run-time overhead.

The liabilities are:

- Integrity Properties: Even with a very good policy, there exists no property or combination of properties that can ensure the integrity of the component or data without doubt. However, based on the type of the property, this remaining uncertainty can be minimized or adjusted to the needs of the actual system, e.g., by performing a risk assessment process with threat modeling.
- Offline Identification: Since the properties do not change during run-time and no additional checks are performed here, integrity violations during run-time are not detected.

## 5.10. Known Uses

Many protocols, such as CAN or USB use Cyclic Redundancy Check (CRC) to ensure data integrity. This check is done before the packet is forwarded to the actual application (i.e., before it is used by the application). In TLS, during the verification of the message authentication code, the receiver calculates a hash value (static property) of the received message and compares it with a reference value generated by the sender (policy). Systems that implement SECURE BOOT [2] are using signatures or hashes to statically ensure the integrity of software components.

## 5.11. See Also

## **5.12. References**

- [1] Hanmer, R. S. (2013). *Patterns for fault tolerant software*. John Wiley & Sons.
- [2] Löhr, H., Sadeghi, A. R., & Winandy, M. (2010, February). Patterns for secure boot and secure storage in computer systems. In *2010 International Conference on Availability, Reliability and Security* (pp. 569-573). IEEE.

## **5.13. Sources**

Rauter, T., Höller, A., Iber, J., & Kreiner, C. (2016, July). Static and dynamic integrity properties patterns. In Proceedings of the 21st European Conference on Pattern Languages of Programs (pp. 1-11).

# **6. Dynamic Integrity Properties**

## **6.1. Intent**

The intent of DYNAMIC INTEGRITY PROPERTIES is to reflect the integrity of a system by detection of abnormal behavior.

## **6.2. Example**

## **6.3. Context**

A system consists of multiple interacting components. Therefore, the occurrence of different types of faults is very likely. Faults may be of natural (e.g., hardware faults due to external events or material wear out) or human origin. Human-made faults can be non-malicious (e.g., accidental) or malicious (i.e., a security violation). All types of faults can result in an unintended manipulation of a component or data.

## **6.4. Problem**

How to ensure that a state transition of a component or a change of a data value does not harm the integrity of the overall system?

The following forces apply:

- Component Integrity: The system's integrity would be violated when the integrity of the component that is going to be executed or data value that is going to be used is violated.
- Integrity Properties: Integrity of a component or data cannot be computed directly.
- Online Violation: An integrity violation may happen during run-time and result in a behavioral change of a system component or in forged information.
- Offline Verification Not Possible: The component is not examinable from outside before it is used. Therefore, static properties cannot be used.
- Dynamic Properties: The properties are present before execution, and change, vanish or appear over time. Thus, a single evaluation is thus not reasonable.

## **6.5. Solution**

An integrity representation of the component is generated periodically or event-triggered during run-time. Integrity cannot be computed directly, therefore other properties that reflect the integrity of the component are used. The properties are used to reflect the behavior of the component (e.g., state transitions) and therefore have to be examinable during run-time. Moreover, it has to be possible to formulate a policy, that enables a decision concerning the component's integrity based on the presence, absence, or values of these properties.

The properties have to be identifiable and verifiable. This means that it has to be possible to deterministically compute these properties with the given data or component as input. Moreover, it has to be possible to formulate a policy that enables a decision concerning the component's integrity based on the presence, absence, or values of these properties. The properties have to be examinable before the component is executed.

As illustrated in Figure 8, the properties change over time. Therefore, one evaluation prior to each verification has to be done. At time (1), for example, the value of Property 1 differs compared to time (2). Moreover, Property 3 vanishes completely during time (1) and (2). The integrity policy, however, requires the existence of this property and therefore the component's integrity is considered violated at time (2).

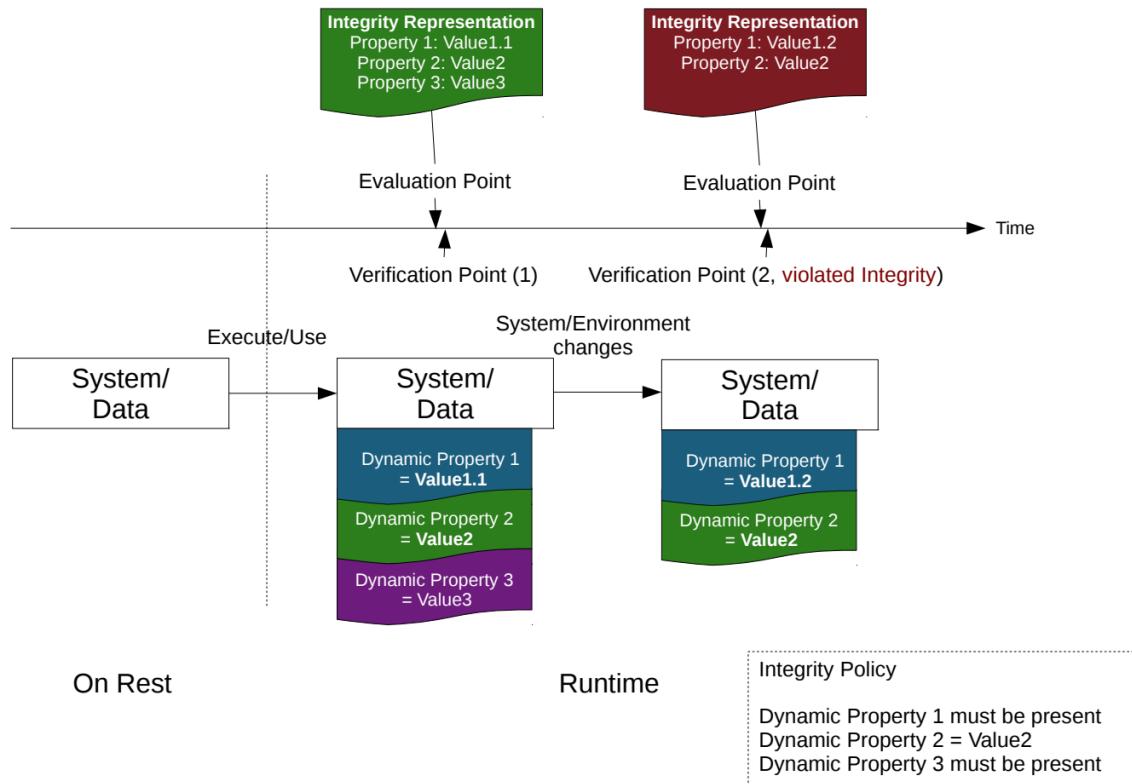


Figure 8: Dynamic properties may change over time. Therefore, they have to be re-evaluated prior to each verification.

## 6.6. Structure

## 6.7. Dynamics

## 6.8. Implementation

One method to examine the integrity of system components is to measure latency and set a REALISTIC THRESHOLD [1]. Another possibility is to use calls to critical system functions as integrity properties, as in “Controlled Virtual Address Space” pattern. In order to detect

security violations, the control flow of a program can be logged and verified with a model of valid control flow transitions. In terms of data integrity, dynamic properties could depend on different data sources. For example, a difference between two data sets has to meet a certain criteria. Whenever one data set changes, the property (i.e., the difference) is updated and an integrity violation can be revealed, even if the static integrity properties of the data sets are intact.

## 6.9. Example Resolved

## 6.10. Consequences

The benefits of the pattern are:

- Component Integrity: The integrity of the component is reflected by the integrity representation that contains static integrity properties.
- Integrity Properties: Assuming a properly crafted policy, the identified properties reflect the behavior of the components and data and thus the integrity of overall system is verifiable.
- Online Violation, Dynamic Properties: A violation of the system's integrity or a changed property during run-time is reflected by the identified properties at the next evaluation point.
- Offline Verification Not Possible: The pattern uses dynamic properties that reflect the behaviour. The integrity of the component can be checked later on.

The liabilities are:

- Integrity Properties: Even with a very good policy, there exists no property or combination of properties that can ensure the integrity of the component or data without doubt. However, based on the type of the property, this remaining uncertainty can be minimized or adjusted to the needs of the actual system.
- Online Violation: The properties may reflect violations of the components/data that occurred prior to the execution of the component or the data usage. However, especially if this violation is the result of a malicious fault, it may also mask itself or disable run-time evaluations.
- Online Violation: Run-time checks require additional resources such as CPU-time and thus may interfere with the actual function of the observed component.

## 6.11. Known Uses

Canaries (magic sequences) are widely used to detect hardware faults or malicious faults such as buffer overflows. Here, a specific value has to be present at a specific address in memory. VOTING [1] calculates the correctness of computation results by exploiting redundancy. Similar concepts are applied in the DISTRIBUTED SAFETY pattern [2]. Many systems use the property of what critical system functions are accessed to protect the integrity of other software components [3,4].

## **6.12. See Also**

## **6.13. References**

- [1] Hanmer, R. S. (2013). *Patterns for fault tolerant software*. John Wiley & Sons.
- [2] Eloranta, V. P., Koskinen, J., Leppänen, M., & Reijonen, V. (2010). A pattern language for distributed machine control systems. *Tampere University of Technology, Department of Software Systems*.
- [3] Loscocco, P., & Smalley, S. (2001). Integrating flexible support for security policies into the Linux operating system. In *2001 USENIX Annual Technical Conference (USENIX ATC 01)*.
- [4] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P., & Gligor, V. (2000). {SubDomain}: Parsimonious Server Security. In *14th Systems Administration Conference (LISA 2000)*.

## **6.14. Sources**

- Rauter, T., Höller, A., Iber, J., & Kreiner, C. (2016, July). Static and dynamic integrity properties patterns. In Proceedings of the 21st European Conference on Pattern Languages of Programs (pp. 1-11).

# **7. Software Container**

## **7.1. Intent**

A Software Container provides an execution environment for applications sharing a host operating system, binaries, and libraries with other containers. Containers are lightweight, portable, extensible, reliable, and secure.

## **7.2. Example**

Our organization has development and testing teams working at distributed locations. We generally use cloud computing in addition to company's local infrastructure. We need a quick and easy way to provide a standardized environment to execute and test all kinds of applications. Even if we have applications from different companies running on our system, we have to provide each one a standard platform for execution. In addition to this, necessary isolation between these applications is security requirement. We have to furnish frequent releases of our software so the deployment process and distribution is becoming difficult. For a cost-effective operation we need to reduce overhead and improve security. Our applications execute in a few commonly used operating systems, we do not have very specific customized platform requirements.

## **7.3. Context**

Software developers/Institutions developing many applications that will execute in multiple computer systems and/or cloud-based virtual environments and using a few types of operating systems. Security, reliability, and overhead are concerns

## **7.4. Problem**

We want to be able to run applications in a self-contained environment in such a way that both (application and execution environment) can be treated as a single unit.

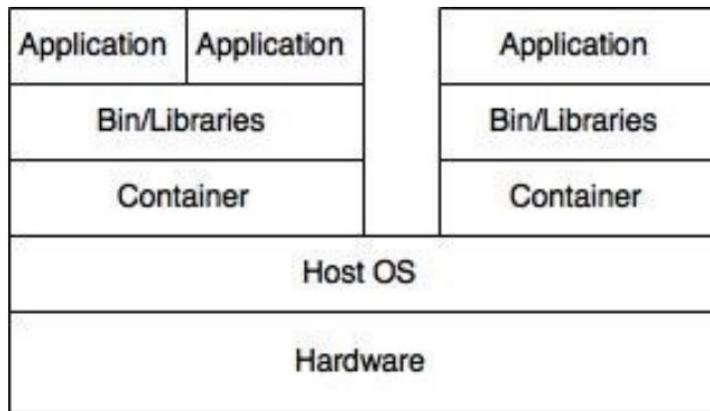
The solution to this problem is guided by the following forces:

- Overhead: We want the overhead of the execution environment to be as low as possible; otherwise, we can use more flexible solutions such as virtual machines.
- Portability: We want applications to be portable across execution environments; that is, they should be able to be moved, for example, from one location to another location, without large modifications.
- Controlled Execution: We want to control application execution in a simple and convenient way.
- Cost: The cost of running applications should be as low as possible in terms of resources.
- Isolation: When multiple applications are running in the same OS we want a strong isolation between them so that if one of them is malicious, compromised, or fails, attacks or errors do not propagate to other applications.

- Opaqueness: Applications running on the same OS should not be aware of each other to ensure security.
- Transparency: The specific environment should be transparent to the application.
- Scalability: The number of applications sharing one type of OS should be scalable.
- Extensibility: It should be possible to dynamically provide additional services to the hosted applications like logging/auditing, filtering, persistence, and others
- Recovery: We need to be able to recover/replicate the application easily.

## 7.5. Solution

Provide a runtime environment that can support the isolated execution of applications on a shared Host OS, this is a Software Container (Figure 9). Multiple processes can share one container in special circumstance [1]. They also may share binaries and libraries with other containers. Containers provide isolated execution and extensible services to the application.



*Figure 9: Two Containers sharing one OS*

## 7.6. Structure

Figure 10 shows the class diagram for this pattern. A Container controls a set of Applications sharing a Host OS that provides a set of Resources. An Interceptor mediates the services provided to the application by the container. Applications hosted in containers can be accessed remotely through Proxies, where the Container acts as a broker. The client interacts with the Application Proxy, which represents the application. The application interacts with the Client Proxy, which represents the client. The Container provides a set of Services to the applications. Container Images are stored in image repositories.

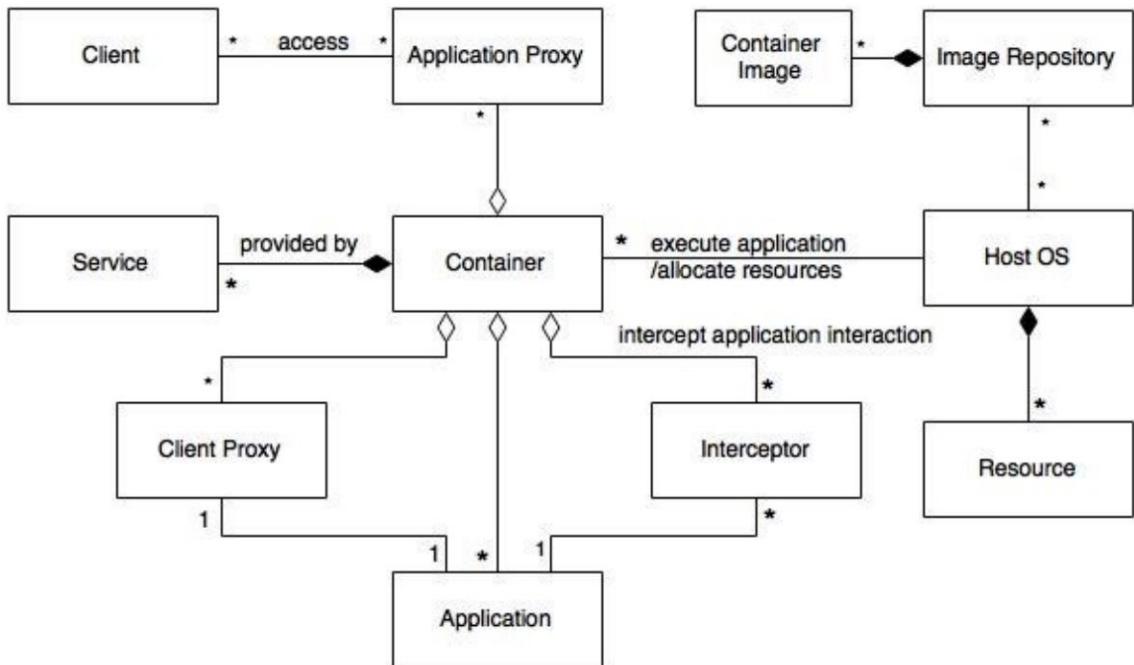


Figure 10: Class diagram of the Container pattern

## 7.7. Dynamics

Figure 11 shows a use case to execute an application in a container. A remote client executes an application through its proxy. The container transmits client requests to the Application through the Client Proxy. In order to execute the request or access a resource, application issues an OS call. This call goes to the interceptor before it is forwarded to Host OS.

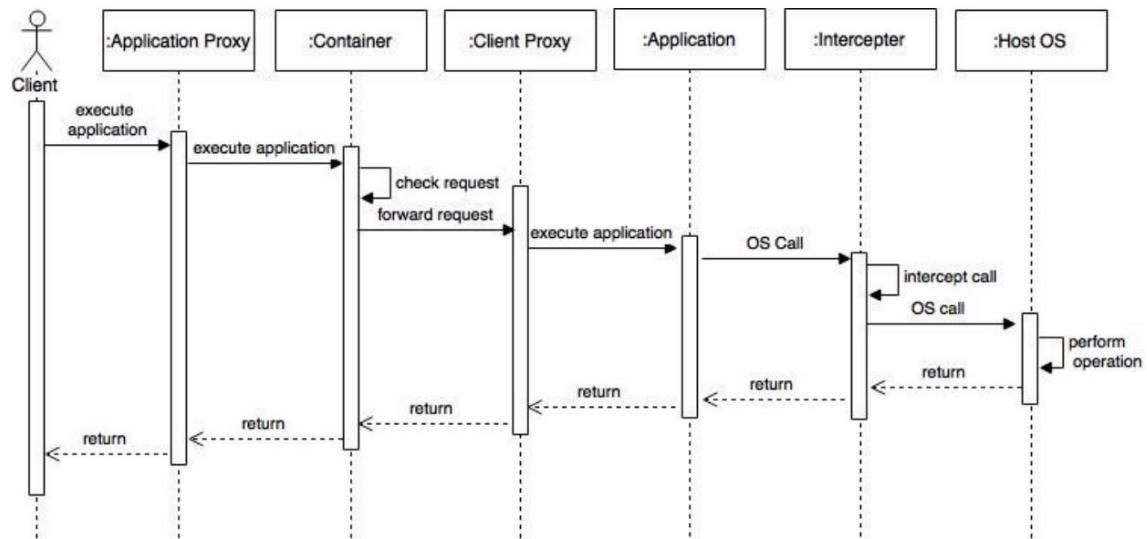


Figure 11: Sequence diagram for the use case 'Execute an application in container'

## **7.8. Implementation**

- In a virtualized environment containers can be mixed with virtual machines or bare metal servers. In other words, we can execute applications directly in a physical processor, in a virtual processor, or in a container.
- The Host OS should be able to have primitives to execute each process in strongly isolated execution environment; for example, Linux-based containers such as Docker use kernel namespaces and groups to isolate containers. Docker uses the libcontainer library to build implementations on top of libvirt, LXC [2,3]. In this way, resources can be isolated, and services restricted to let applications have a specific view of the operating system [4].

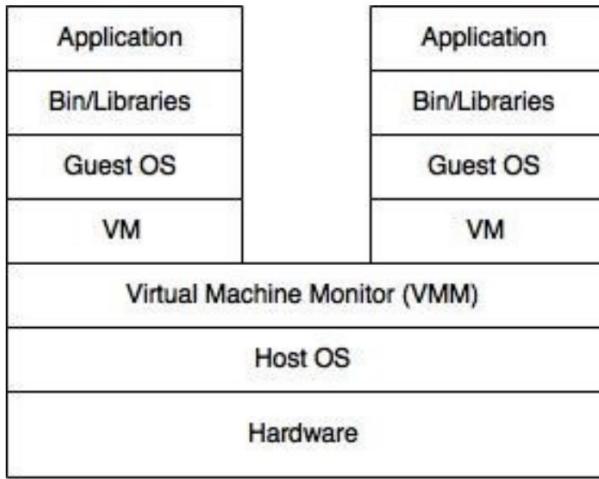
## **7.9. Example Resolved**

The company started using containers on their servers. They built images for their applications so that each could run in a separate container. This provided lower overhead and more security for the whole system. In addition the use of containers and application images made distribution and deployment of applications very easy

## **7.10. Consequences**

This pattern presents the following advantages:

- Overhead: Containers are more efficient since they do not require separate guest OSs as for the case of VMs (Compare Figure 12 with Figure 9). They will, however, be slower as compared to directly executing an application on Host OS without virtualization.
- Portability: Containers can be executed in any processor, and they can relieve application developers and testers of worrying about application distribution.
- Controlled Execution: Containers can control application execution as they can control and filter interactions with the Host OS.
- Cost: Host OS is shared by multiple containers, so unlike VMs we do not need to purchase separate licenses for Guest OSs on each VM.
- Isolation: The Container can use the facilities of the host OS to provide isolation between applications running on the same OS. This feature protects the other applications of attacks or errors in applications running in different containers.
- Opaqueness: Applications running in separate containers on the same OS are not aware of each other, which can prevent attacks.
- Transparency: The specific environment becomes transparent to the application when executed within a container. Changes made to the OS or Application can be handled by container modification, without affecting other containers.
- Scalability: Containers make the system more scalable since the number of applications sharing one OS can be increased provided enough hardware resources are available.
- Extensibility: Interceptors allow adding services such as logging/auditing, security, or others to an application.



*Figure 12: A Virtual Machine*

Liabilities of the pattern include the following:

- Use of containers can slow down application execution since we are using an additional layer of interceptors for indirection of messages between OS and application. However, this overhead should be smaller than using separate virtual machines.
- Containers are meant to provide isolation between applications, so if there is a need for collaboration between applications, the task becomes more difficult if they are executing in separate containers. This can still be achieved for example; Docker provides an option to securely link containers through ‘bridge’ network [5].
- We can only use one OS in each container. If we need different operating systems, we can have separate sets of containers or use virtual machines.
- Security or reliability flaws in the common OS affect all the applications running on it.

## 7.11. Known Uses

- Docker provides portable, lightweight containers, using Linux virtualization [4]. Docker was known to have security vulnerabilities but few security feature were added recently including hardware signing for container image developers, digital signatures for authenticating images in repository and image scanning for known vulnerabilities [6].
- Cisco - Cisco Virtual Application Container Services automate the provisioning of virtual private data centers and deploy applications with compliant, secure containers [7].
- Rocket - A product of CoreOS. This container attempts to be composable, secure, open format and runtime components, and simple discoverable images [8].

In addition to these, Windows server containers and Hyper V containers supported for Windows Server 2016. [9]. Other container providers include Google, Amazon Web Services, IBM, Microsoft, and HP.

## 7.12. See Also

- Interceptor [10]--allows services to be added transparently to a framework and triggered automatically in the present of specific events.

- Broker [11]--the Broker structures distributed systems with separate components that interact by remote service calls. A broker coordinates communications, including forwarding requests and sending back results and exceptions.
- Reference Monitor. In a computational environment in which users or processes make requests for data or resources, this pattern enforces declared access restrictions when an active entity requests resources. It describes how to define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations
- Virtual Machine Operating System. Provides a set of replicas of the hardware architecture (Virtual Machines) that can be used to execute (maybe different) operating systems with a strong isolation between them.
- Controlled Virtual Address Space (Sandbox). How to control access by processes to specific areas of their virtual address space (VAS) according to a set of predefined access rights? Divide the VAS into segments that correspond to logical units in the programs. Use special words (descriptors) to represent access rights for these segments.

A formal analysis of component containers is presented in [12].

## 7.13. References

- [1] Docker. (2015). Using Supervisor with Docker. Docker Docs. Retrieved December 17, 2015, from [https://docs.docker.com/engine/articles/using\\_supervisord/](https://docs.docker.com/engine/articles/using_supervisord/)
- [2] LXC. (2015). Linux Containers - LXC - Introduction. Retrieved June 9, 2015, from <https://linuxcontainers.org/lxc/introduction/>
- [3] Linux LXD. (2015). Linux Containers - LXD - Introduction. Retrieved June 9, 2015, from <https://linuxcontainers.org/lxd/introduction>
- [4] Docker. (2015). Docker. Retrieved April 10, 2015, from <http://www.docker.com/>.
- [5] Container Links. (2015). Legacy container links. Retrieved December 1, 2015, from [https://docs.docker.com/engine/userguide/networking/default\\_network/dockerlinks/#container-linking](https://docs.docker.com/engine/userguide/networking/default_network/dockerlinks/#container-linking).
- [6] Charles Babcock. (2015). Docker Tightens Security Over Container Vulnerabilities. Retrieved December 1, 2015, from <http://www.informationweek.com/cloud/platform-as-a-service/docker-tightens-security-over-container-vulnerabilities/d/d-id/1323178>.
- [7] Cisco. (2015). Virtual Application Container Services. Retrieved June 22, 2015, from <http://www.cisco.com/c/en/us/products/switches/virtual-application-container-servicesvacs/index.html>
- [8] Alex Polvi. (2014). CoreOS is building a container runtime, rkt. Retrieved April 25, 2015, from <https://coreos.com/blog/rocket/>.
- [9] Simon Bisson. (2015). First look: Run VMs in VMs with Hyper-V containers. Retrieved December 9, 2015, from <http://www.networkworld.com/article/3013224/virtualization/first-look-run-vms-in-vms-with-hyper-v-containers.html>.

- [10] Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000). *Pattern-Oriented Software Architecture Volume 2: Patterns for Concurrency and Networked Objects* (Volume 2 ed.). Wiley.
- [11] Buschmann, F., Meunier, Regine, Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture Volume 1: A System of Patterns* (Volume 1 ed.). Wiley.
- [12] Sridhar, N., & Hallstrom, J. O. (2006, March). A behavioral model for software containers. In *International Conference on Fundamental Approaches to Software Engineering* (pp. 139-154). Springer, Berlin, Heidelberg.

## 7.14. Sources

- Syed, M. H., & Fernandez, E. B. (2015, October). The software container pattern. In *Proceedings of the 22nd Conference on Pattern Languages of Programs* (pp. 1-7).

## **8. Integrity Protection**

### **8.1. Intent**

A system protects its integrity by preventing harmful actions. This is done by the interception of defined action classes and identifying their impact on the system's integrity. If a specific action would violate the integrity, the system refuses its execution.

### **8.2. Example**

### **8.3. Context**

A computer system runs a set of software modules. Different types of adversaries may benefit from altering existing modules to act maliciously or to add completely new modules that act in their behalf since the system may protect critical information or resources. The system may be a server, a PC, any kind of embedded system or a virtual machine. However, it runs autonomously (i.e., with little human interaction) most of the time.

### **8.4. Problem**

An adversary tries to tamper with the software modules on the system. A software module that is modified without authorization possibly leaks critical information or resources or harms the functionality of the overall system. Therefore, the integrity of running software modules has to be ensured.

The following forces apply:

- Prevent Execution of Malicious Software: Execution of malicious software modules harms the integrity of the overall system. After the modules are executed, the integrity violation can be located in any subsystem and may be hard to find. Therefore, the execution of the malicious module should be prevented in the first place.
- Integrity without Monitoring: The system might not be accessible, and it is not possible to monitor it all the time. Though, the integrity of the system should be ensured at any time.
- Updatability: The solution should provide the possibility to do software updates or install additional software modules without changing the underlying system.
- No Mutation: The solution should not require changes in existing modules.

### **8.5. Solution**

The system either ensures that all loaded software modules are known and verified before, or it ensures that no module is able to violate the integrity of other modules or the whole system.

## 8.6. Structure

As shown in Figure 13, the system is running a set of SoftwareModules. Calls to the SystemAPI are redirected to the DecisionUnit. In order to decide whether a specific call is allowed or not, the DecisionUnit performs an identification of the request via the RequestIdentificationUnit and compares the result to an IntegrityPolicy.

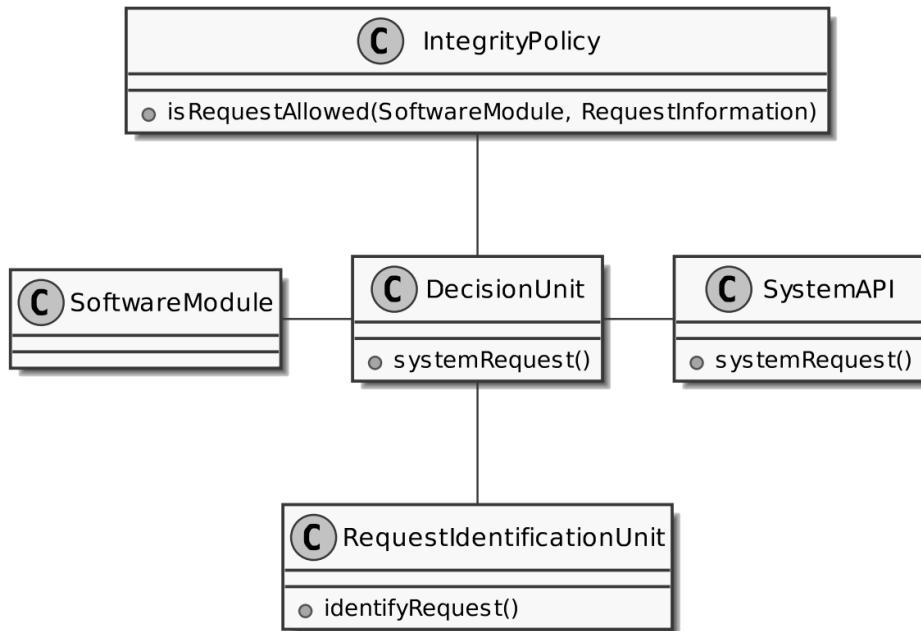


Figure 13: INTEGRITY PROTECTION: The system intercepts privileged system requests of all software modules to prevent integrity violations.

## 8.7. Dynamics

Figure 14 illustrates the basic procedure: Whenever SoftwareModule calls a function of the SystemAPI, the DecisionUnit intercepts this call and triggers the RequestIdentificationUnit. This module gathers information about the request and the result is compared to the IntegrityPolicy. Based on this policy, the DecisionUnit either forwards the call to the actual SystemAPI or returns an error.

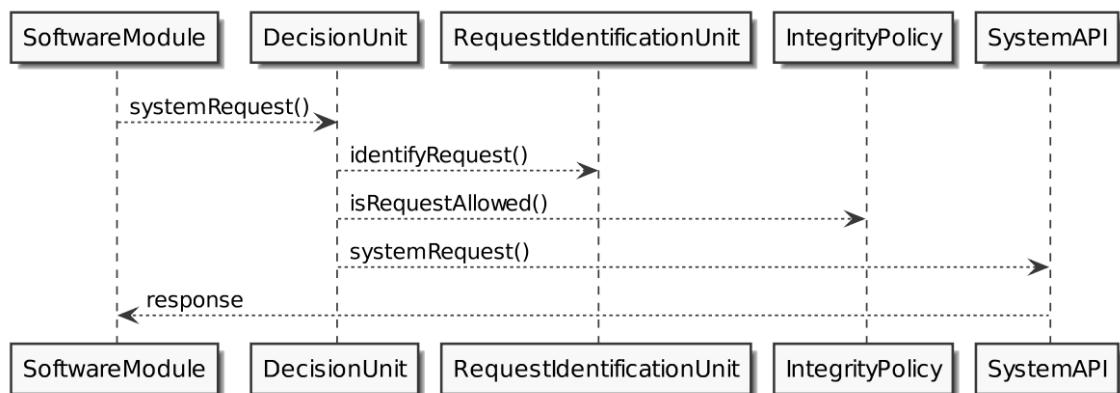


Figure 14: INTEGRITY PROTECTION: The decision unit intercepts system requests and only allows them if they conform to the policy (i.e., would not violate system's integrity).

## **8.8. Implementation**

The implementation of the DecisionUnit can be done in different ways. Systems that implement the SECURE BOOT pattern verify a software module by hashing its binary and comparing it to a known value prior to execution. On the other hand, systems based on the SANDBOX pattern are verifying all calls to the SystemAPI and decide whether it is allowed for the specific SoftwareModule or not.

## **8.9. Example Resolved**

## **8.10. Consequences**

Benefits:

- Prevent Execution of Malicious Software: Assuming that the IntegrityPolicy is complete, the integrity checks prevent malicious software from being executed on the system.
- Integrity without Monitoring: No external supervisor is needed. The system enforces its integrity autonomously.
- No Mutation: There is no need to alter the software modules itself. —(Updatability): Based on the chosen implementation, little (e.g., sign the new module) or no effort is needed to add additional non-harmful modules to the system.

Liabilities:

- Maintenance: In systems with very volatile and heterogeneous configurations, the maintenance of the IntegrityPolicy is potentially hard.
- Performance: INTEGRITY PROTECTION may reduce the performance significantly. Especially on lightweight devices, this might be a problem.
- Updatability: Adding software that does not conform to the IntegrityPolicy is not possible. Similar problems may arise with software updates.

## **8.11. Known Uses**

Since there exist a variety of implementations of both, the SANDBOX pattern [1,2] and the SECURE BOOT pattern [3], the INTEGRITY PROTECTION pattern is implemented in many systems. Some systems, like Android, combine both implementation patterns to ensure integrity. Another example is a shadow stack (example, ROPDefender [4]). Here redundancy is used to protect software against stack overflow attacks. A shadow copy of the actual stack is held and, on each return, the system checks whether the return addresses of the stack and the shadow stack are equal. If not, the stack integrity is properly harmed.

## **8.12. See Also**

## **8.13. References**

- [1] Loscocco, P., & Smalley, S. (2001). Integrating flexible support for security policies into the Linux operating system. In *2001 USENIX Annual Technical Conference (USENIX ATC 01)*.
- [2] Cowan, C., Beattie, S., Kroah-Hartman, G., Pu, C., Wagle, P., & Gligor, V. (2000). {SubDomain}: Parsimonious Server Security. In *14th Systems Administration Conference (LISA 2000)*.
- [3] Safford, D., & Zohar, M. (2005). Trusted computing and open source. *Information Security Technical Report*, 10(2), 74-82.
- [4] Davi, L., Sadeghi, A. R., & Winandy, M. (2011, March). ROPdefender: A detection tool to defend against return-oriented programming attacks. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security* (pp. 40-51).

## **8.14. Sources**

- Rauter, T., Höller, A., Iber, J., & Kreiner, C. (2015, July). Patterns for software integrity protection. In *Proceedings of the 20th European Conference on Pattern Languages of Programs* (pp. 1-10).

## **9. Integrity Attestation (Remote Attestation)**

### **9.1. Intent**

INTEGRITY ATTESTATION is a procedure, that allows a system to proof its maintained integrity state (i.e., that it was not changed in an unauthorized way) to another system.

### **9.2. Example**

### **9.3. Context**

A client is communicating with an application on a server. The application on the server requires sensitive information from the client (e.g., passwords or pins for online banking) or the authenticity of the information provided by the server is very important to the client. The client wants to check the server's integrity prior to sending sensitive information to the server.

### **9.4. Problem**

The client needs to make sure that the server is in a trustworthy state and the integrity of its running software modules is assured.

The following forces apply:

- Integrity Measurement: The server has to measure properties that reflect its integrity, otherwise the client is not able to verify it.
- Reference Measurement List: The server and the client have to share a pre-agreed measurement list or policy, that defines which measured properties define a maintained integrity.
- Integrity of Measurement List: Since malicious software may be executed on the server, the integrity and authenticity of the resulting measurement list has to be ensured too.

### **9.5. Solution**

A system (prover) proves its integrity to a client (challenger) by taking measurements of properties that reflect its integrity and ensuring that the measurement cannot be tampered with.

### **9.6. Structure**

As shown in Figure 15, the Prover is executing a set of SoftwareModules. The PropertyMeasurementUnit is in charge to measure integrity-properties of all modules. These properties have to reflect the integrity of the modules or the overall system (e.g., did somebody tamper with the software module). The MeasurementResults are stored in the ResultStorage. Since potential malicious applications may be executed on the system, the

ResultStorage has to be implemented in a way that prevents malicious applications from tampering with it. Moreover, the properties of a module have to be measured before the module is able to forge it (e.g., prior to its execution). Since the connection between the Client and the Prover may be insecure, the integrity and authenticity of the MeasurementResult has to be ensured by a DIGITAL SIGNATURE WITH HASHING. In order to prevent potentially malicious software from forging this signature on the prover, this signature has to be generated by hardware (i.e., the signature key is not accessible by software). The Client compares the MeasurementResults with an IntegrityPolicy that defines reference measurements that are considered trustworthy.

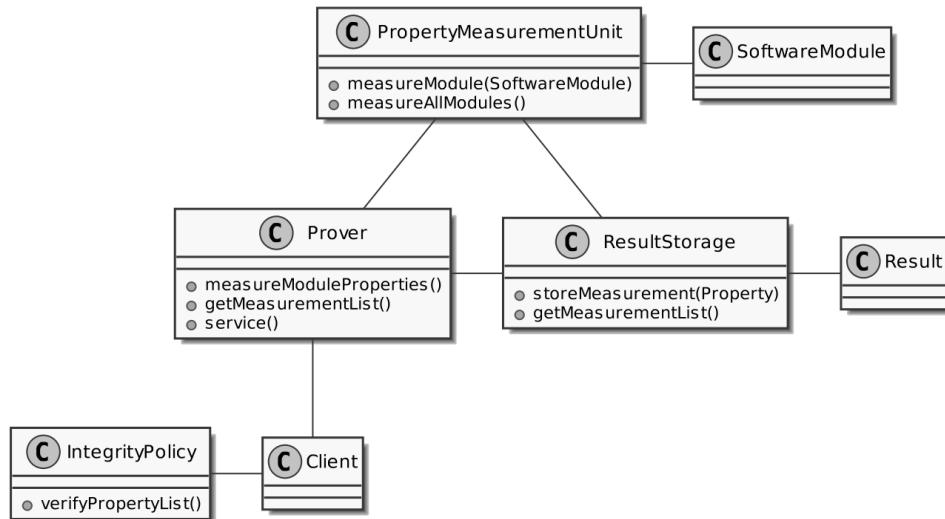


Figure 15: INTEGRITY ATTESTATION: The Prover observes its running software modules with the PropertyMeasurementUnit. The measurements

## 9.7. Dynamics

As shown in Figure 16, the Prover measures all its modules via the PropertyMeasurementUnit. Whenever a Client wants to communicate with the Prover, the Prover signs its MeasurementResults and sends it to the Client. Now, the client compares the received results with the IntegrityPolicy and is able to decide, whether the integrity of the Prover is given. Only if the received list matches the policy, the actual communication is started.

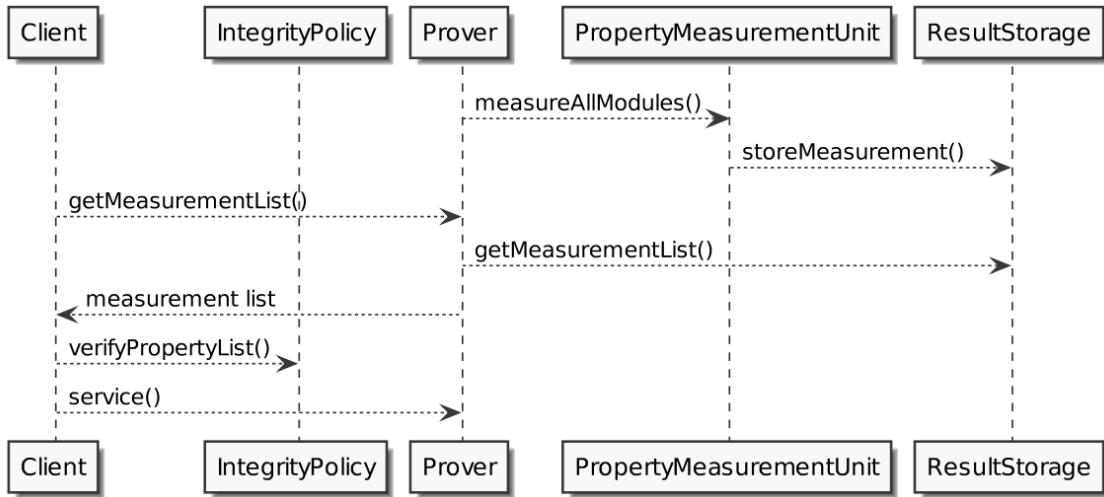


Figure 16: INTEGRITY ATTESTATION: The client verifies the integrity of the prover prior to the actual communication.

## 9.8. Implementation

Similar to INTEGRITY PROTECTION, this pattern can be implemented in different ways. In systems that implement authenticated boot (a variation of SECURE BOOT), the hash values of executed modules are used as integrity-properties. Another approach is to identify application behavior (similar to SANDBOX) by logging calls to critical system functions and their arguments. It is possible to implement INTEGRITY PROTECTION and measure the integrity of the integrity-protection system. In this case, the Client only has to verify this single measurement. If the integrity-protection system is in place, no modified software can be executed.

## 9.9. Example Resolved

## 9.10. Consequences

Benefits:

- Integrity Measurement: The integrity of the system's software modules is identified by the quantification of measurable properties.
- Reference Measurement List: The integrity policy can be defined based on the client's requirements of the server's state.
- Integrity of Measurement List: The MeasurementResults are stored in a SECURE STORAGE. Thus, malicious software is not able to corrupt this information.

Liabilities:

- Reference Measurement List: The IntegrityPolicy on the client has to be up-to-date in order to enable the client to verify the prover. Thus, it has to be ensured that the IntegrityPolicy conforms to the server's configuration everytime the server gets updated.

- Integrity of Measurement List: The prover has to ensure that the measurement list cannot be tampered with. This requires special care on the prover platform, as well as on the communication channel.

## 9.11. Known Uses

INTEGRITY ATTESTATION is implemented in many systems in different ways. In [1], this pattern is used to implement access control of devices to computer networks. An application-level implementation, called Integrity Measurement Architecture (IMA), exists for Linux [2]. Other approaches like DR@FT use SANDBOX-based methods [3].

## 9.12. See Also

## 9.13. References

- [1] Feng, W., Qin, Y., Yu, A. M., & Feng, D. (2011, November). A DRTM-based method for trusted network connection. In *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 425-435). IEEE.
- [2] Sailer, R., Zhang, X., Jaeger, T., & Van Doorn, L. (2004, August). Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security symposium* (Vol. 13, No. 2004, pp. 223-238).
- [3] Xu, W., Zhang, X., Hu, H., Ahn, G. J., & Seifert, J. P. (2011). Remote attestation with domain-based integrity model and policy analysis. *IEEE Transactions on Dependable and Secure Computing*, 9(3), 429-442.

## 9.14. Sources

- Rauter, T., Höller, A., Iber, J., & Kreiner, C. (2015, July). Patterns for software integrity protection. In *Proceedings of the 20th European Conference on Pattern Languages of Programs* (pp. 1-10).

# 10. Content-Dependent Authorizer

## 10.1. Intent

How can we restrict subjects (users, systems, parties) to access (read, write) only data with specific values? Filter the values obtained from applying authorization rules to the data according to a predicate (condition) that selects specific values.

## 10.2. Example

A hospital uses RBAC to protect its patient database. The role “doctor” gives access to all patient records. A doctor found out that a famous singer was in the hospital. Although he was not treating him, he got his medical information and sold it to a newspaper. The singer sued the hospital and collected a large sum of money. Another event like this and the hospital could go out of business.

## 10.3. Context

Some institutions need to keep large amounts of sensitive information. In particular, some institutions keep sensitive information about individuals, other institutions, or about their own businesses. Users can make requests for data after being identified and authenticated (Authenticator pattern). We have mechanisms to determine the values of data items. The Authorizer pattern implements the Access Matrix model ( $s,o,t$ ), which describes access by subjects  $s$  (actors, entities) to protected objects  $o$  (data/resources) in specific ways  $t$  (access types), as represented in Figure 17. The association class Right indicates the access type that can be used by a subject to access an object. The operation “checkRights” can be used to find the objects accessible to a subject or the subjects that can access an object.

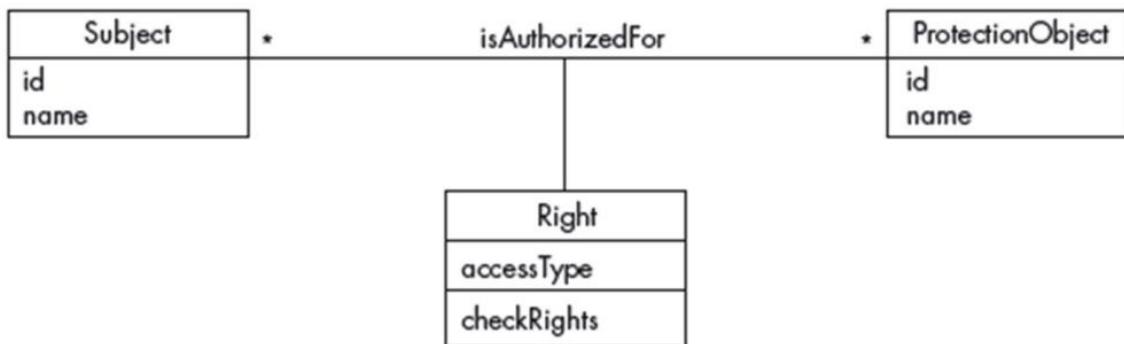


Figure 17: Class diagram for the Authorizer (Access Matrix) pattern [Fer13].

## 10.4. Problem

How can we restrict subjects (users, systems, parties) to access only data which is relevant to their functions or concerns them? Without some control subjects may access too much information which may lead to misuses.

The solution to this problem is constrained by the following forces:

- Policy—we should be able to express authorization policies that restrict users to access (read or write) only data relevant to their functions or about themselves.
- Overhead—evaluating access decisions should take a reasonably low time.
- Security information protection—the information used to decide access should be protected or unauthorized accesses could be possible.
- Transparency—the users should not need to do any special actions to access the data they want; they should only get security violation warnings in case they ask for data they should not access.
- Scalability—the number of subjects and objects should be scalable.
- Accountability—because we are dealing with sensitive data, we should keep track of accesses for future auditing.

## 10.5. Solution

Add to each authorization rule a predicate to constrain the records that the subject can get; now the authorization rule becomes  $(s, o, t, p)$ , where  $s$  is a subject,  $o$  is the object,  $t$  is a access type, and  $p$  is a predicate that evaluates to True or False. The records where the predicate is True are returned to the requester.

## 10.6. Structure

Figure 18 shows the class diagram of this pattern. The Subject represents an active entity able to request access to records. The Object is the class that describes the type of records requested. Right represents the access type allowed to the subject.

The Reference Monitor OCL constraint is:

```
if  $\exists r = (s, o, t, p)$  IN Auth_Rules •  $p(s, o, t) = \text{True}$ 
then • records ( $p=\text{True}$ ) s can apply t
else securityViolation
```

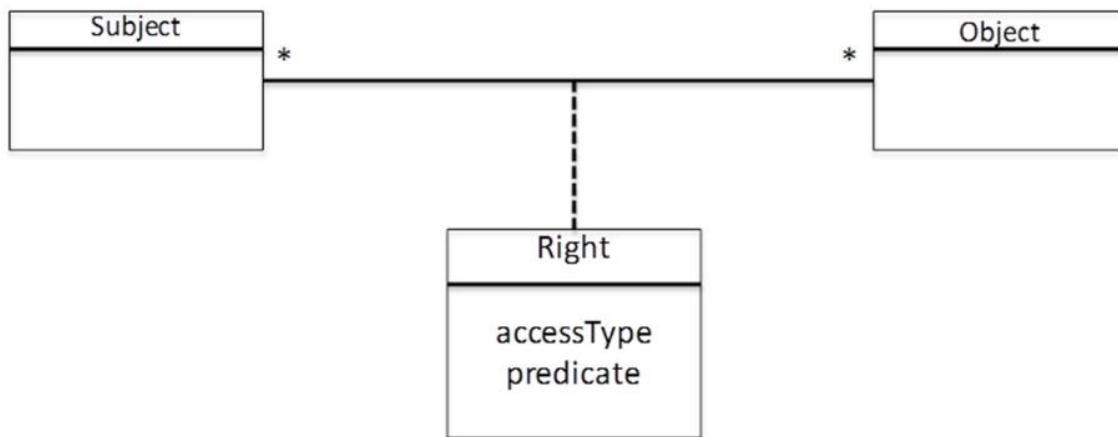


Figure 18: Class diagram of the Content-dependent pattern.

## 10.7. Dynamics

Figure 19 is the sequence diagram for the use case “Request some information”.

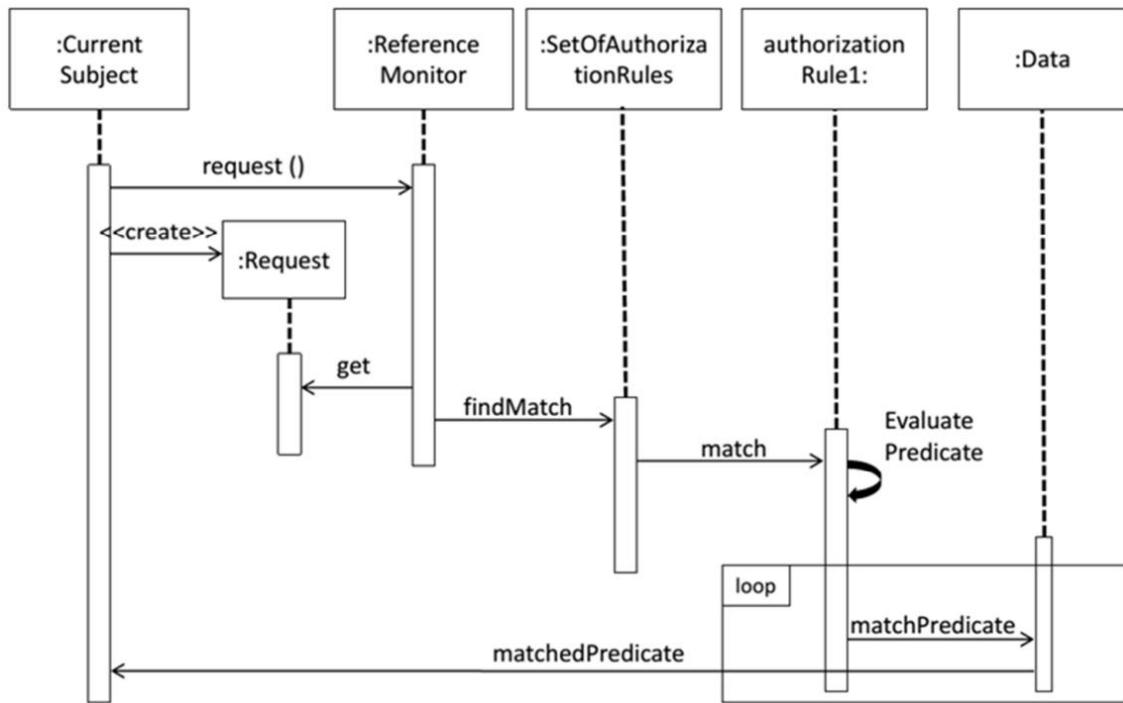


Figure 19: UC: Request some information by value.

The predicate can refer to a system variable, e.g. day of the week, alarm condition, or similar.

## 10.8. Implementation

- An operating system access control for files is not enough; we need a DBMS so we can select records to return to the requester by checking the value of specific fields in records.
- If the number of records is high, it may take a long time to find the matching records.
- Record selection can be accelerated by using compile-time evaluation of queries.

## 10.9. Example Resolved

The affected patient was not a patient of the doctor who leaked his information. The hospital now uses content-dependent authorization and restricts doctors to access only the information of their own patients. This will not totally eliminate the problem because doctors can still leak information about their own patients but will significantly reduce it.

## 10.10. Consequences

This pattern has the following advantages:

- Policy—we can express any access control policy that depends on data values or combinations of them.
- Overhead—evaluating access decisions takes a reasonably low time if the number of records is not too high or we use some acceleration approach.

- Security information protection—the information used to decide access can be protected in the same way (put authorization rules in tables of the relational database).
- Transparency—the users do not need to do any special actions, except to execute queries, they may get security violation warnings.
- Scalability—the number of subjects and objects can increase significantly. Specific implementations can limit the number of records to be retrieved.
- Accountability—we can keep track of accesses for future auditing using the Security Logger/Auditor together with this pattern.

Its liabilities include:

- Overhead—if many records or relational joins are involved, the overhead can be considerable.

## 10.11. Known Uses

Almost all DBMS products can enforce this type of authorization:

- Oracle [1]. The Oracle DBMS includes authentication, authorization, and logging. The authorization mechanism provides content-dependent access control.
- DB2 [2]—a family of relational database servers. In recent years some products have been extended to support object-relational features and non-relational structures, in particular XML.
- Microsoft DB Server [3].
- MySQL [4].

## 10.12. See Also

- Authenticator: When a user or system (subject) identifies itself to the system, how do we verify that the subject intending to access the system is who it says it is?
- Authorizer: A.K.A. Access Matrix. Describes who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled. It indicates for each active entity, which resources it can access, and what it can do with them.
- Reference Monitor: Enforces the authorization rules defined by different types of authorizers.
- Security Logger/Auditor: How can we keep track of user's actions in order to determine who did what and when? Log all security-sensitive actions performed by users and provide controlled access to records for Audit purposes.
- Need to know: Grants users only the rights they need to perform their duties.

## 10.13. References

[1] Oracle. (2012, July). *Security and Compliance*. Oracle Database. Retrieved February 28, 2014, from <https://www.oracle.com/technetwork/database/security/index.html>

[2] IBM. (n.d.). IBM Db2 – Data Management Software. <https://www.ibm.com/analytics/db2>

[3] Wikipedia contributors. (2014). Microsoft SQL Server. Wikipedia.

[https://en.wikipedia.org/wiki/Microsoft\\_SQL\\_Server](https://en.wikipedia.org/wiki/Microsoft_SQL_Server)

[4] MySQL. (n.d.). Security in MySQL. Retrieved April 2, 2014, from

<https://dev.mysql.com/doc/mysql-security-excerpt/5.1/en/>

## 10.14. Sources

Fernández, E. B., Monge, A. R., Carvajal, R., Encina, O., Hernández, J., & Silva, P. (2014, July). Patterns for content-dependent and context-enhanced authorization. In *Proceedings of the 19th European Conference on Pattern Languages of Programs* (pp. 1-10).

# 11. Context-Enhanced Authorizer

## 11.1. Intent

When an authorized subject requests data about an object, provide some contextual information to make this data more explicit and meaningful.

## 11.2. Example

Ernest was wrongly arrested, and he was acquitted immediately after the error was discovered. He applied for a job several years later and he was turned down because his potential employer got a record from one of the several background databases available in the US and other countries that only said he was arrested.

## 11.3. Context

Many institutions that provide data about individuals have a large amount of information, but the requestor needs to know what to ask. To decide or to understand better a problem, we need information which we might not even know exists and therefore we do not ask for it. This information is sensitive, and its access is usually controlled. We assume users are authenticated and authorized before they can get any data from the system. Because the additional information is provided for specific records, the Content-Dependent Authorizer must first be applied.

## 11.4. Problem

In many cases some data is not meaningful unless it has a context. Without the context the data requestor may come to erroneous conclusions. How can we provide the requestor with the necessary information to make a proper interpretation?

The solution is affected by the following forces:

- Relevancy—the additional information we return to the user must be relevant and meaningful; otherwise, we may confuse the requestor.
- Overhead—the time to return the answer to a request should not be too long.
- Policy—the owner of the data should have policies about what data to include in each context.
- Accountability—we need to record the metadata associated to each query so we can audit it later in case of legal problems.
- Transparency—the requestor should not have to indicate what specific data he wants as context.
- Protection of the security information—the definition of contexts should be protected of tampering which could disclose sensitive information.

## 11.5. Solution

Append to each predicate used to select objects a list of data fields that define a selected context for the request. The specific context is selected based on policies of the institution according to the application.

## 11.6. Structure

Figure 20 shows the class diagram of this pattern. Add a context predicate to class Right that adds a list of values corresponding to the context to each retrieved record. Since the context is a further refinement of content access, the content predicate should be applied first. In the context of access control models the context is what is called an obligation.

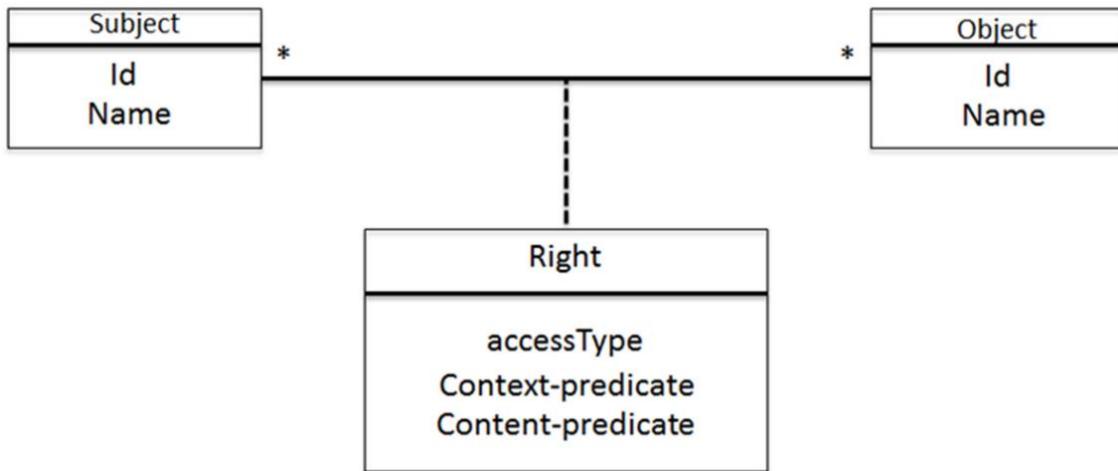


Figure 20: Class diagram of the Context-Enhanced Authorizer pattern.

## 11.7. Dynamics

This solution is an extension of the Content-dependent access control pattern. The Context Data Manager evaluates the context predicate and appends to the original request the information indicated by the predicate. This information is produced by the Generate Context process of the Context Data Manager. Figure 21 shows the UC “Request information”.

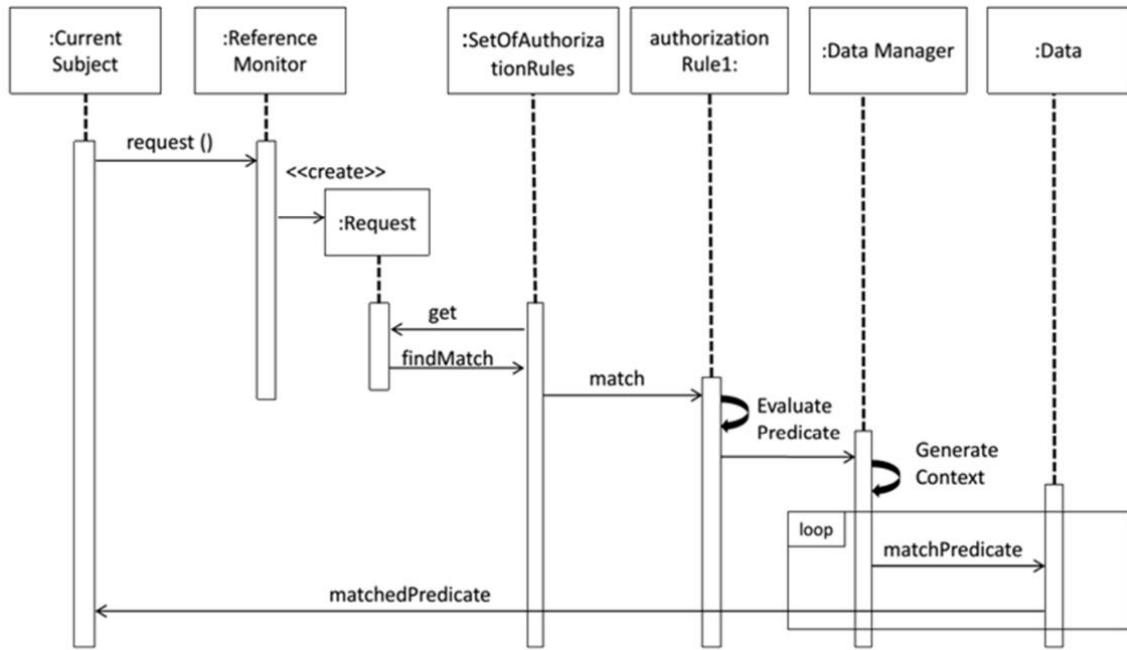


Figure 21: Use case: “Request information”.

## 11.8. Implementation

The institution must define policies about what data to use in each context, depending on the type of application. The predicate in the authorization rule must also have an “obligation”, an extension to the content-dependent authorization rules.

The records that satisfy a specific condition must be retrieved first, the corresponding contexts are then appended to these records’.

## 11.9. Example Resolved

The employer who turned down Ernest realized that they needed more information about potential employees and started using a context-dependent DBMS. Now any suspicious events about candidates can be better interpreted.

## 11.10. Consequences

This pattern presents the following advantages:

- Relevancy—we can define policies to return to the user only relevant and meaningful information.
- Overhead—content-dependent access control must happen first. The extra overhead of adding a context is small for a reasonable amount of context information .
- Policy—the owner of the data can have contexts to add to each type of request.
- Accountability—we can log the metadata associated to each query and we can audit it later.
- Transparency—the requestor does not need to indicate what specific data he wants as context.

- Protection of the security information—the metadata that describes authorization rules can include the contexts.

Liabilities include:

- Possible overhead in retrieving the data for the context.
- It may not be clear for some applications what to put in the context.

## 11.11. Known Uses

- This model comes from [1].
- CaseMap it is a tool from Lexis Nexis for legal teams during a trial that can be used to obtain relevant information for a case [2].
- Lexis Nexis uses the ECL language to correlate data from several databases and presents all what is known about a specific individual [3].

## 11.12. See Also

- Content-Dependent Access Control: Since the information retrieved belongs to a specific individual in the DBMS, this pattern is usually used together with the CtDA.
- Authorization: A.K.A.: Access Matrix. Describe who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled. It indicates for each active entity, which resources it can access, and what it can do with them.
- Context-Dependent Access Control: The access decision is based on the context of a collection of information rather than content within an object. A firewall makes a context-based access decisions when it considers state information on a packet before allowing it into the network; a cell phone has different rights according to a context including location of the user, subscribed services, etc. [4,5].

## 11.13. References

- [1] Fernández, E. B., Summers, R. C., & Coleman, C. D. (1975, May). An authorization model for a shared data base. In *Proceedings of the 1975 ACM SIGMOD international conference on Management of data* (pp. 23-31).
- [2] LexisNexis. (n.d.). Case Map. <http://www.lexisnexis.com/en-us/litigations/products/casemap.page>
- [3] Wikipedia contributors. (n.d.). ECL (data-centric programming language). Wikipedia. [https://en.wikipedia.org/wiki/ECL\\_%28data-centric\\_programming\\_language%29](https://en.wikipedia.org/wiki/ECL_%28data-centric_programming_language%29)
- [4] Kirkpatrick, M., & Bertino, E. (2009, April). Context-dependent authentication and access control. In *International Workshop on Open Problems in Network Security* (pp. 63-75). Springer, Berlin, Heidelberg.
- [5] Nitsche, U., Holbein, R., Morger, O., & Teufel, S. (1998). *Realization of a context-dependent access control mechanism on a commercial platform* (pp. 160-170). na.

## **11.14. Sources**

Fernández, E. B., Monge, A. R., Carvajal, R., Encina, O., Hernández, J., & Silva, P. (2014, July). Patterns for content-dependent and context-enhanced authorization. In *Proceedings of the 19th European Conference on Pattern Languages of Programs* (pp. 1-10).

# 12. History-Based Authentication

## 12.1. Intent

## 12.2. Example

This example sketches the THOFU Security System, hereinafter referred to as TSS. The system has been designed within the framework of the project THOFU, a multidisciplinary collaboration intending to perform a prospective study on technologic concepts and solutions enabling advanced services in the context of the hotels of the future [1]. One of the main research areas of THOFU is related to security. Predictably, monitoring of comfort, safety and security of hotels and many other social and industrial environments of the future will be assigned to pervasive systems. Such systems, operating with little or no human intervention, should in turn control and protect themselves to assure their correct performance and their compliance with requirements such as security. An essential security requirement in such a system is to verify the identity of entities intending to perform tasks or to access resources, i.e. to authenticate them.

The system is composed of entities that can be categorized into several types, according to their roles:

- robots, responsible for the surveillance of the facilities and resources of the organization. Robots may be grouped into patrols.
- controllers, responsible for monitoring the performance of other entities (e.g., robots and other controllers). In case an asset (resource or capability) is unavailable or underperforms, they trigger a corrective action. Controllers may be in turn controlled by other entities.

Next, we present some examples of events of the lifecycle of the system. Let R1, R2, R3 and R4 be robots and C1, C2 controllers:

1. At 01/11/2013 07:30:00, the control of R1 is assigned to controller C1.
2. A new patrol, identified as P1, controlled by C1 and composed of R1, R2, and R3, is created at 01/11/2013 09:00:10. The patrol components are informed about this.
3. At 04/12/2013 09:45:00 C1 notifies R1, R2, and R3 that a new entity, R4, joins the patrol since that time.
4. R1 communicates C1 that sensor S0 11 has reported a temperature above 50o C at 06/12/2013 10:45:00 in room B and proceeds to probe into the incident.
5. At 01/10/2014 07:30:00, the control of R1 is transferred to controller C2.
6. At 01/10/2014 07:30:01, C2 issues R1 the order: "You must send me a greeting signal on security time intervals  $\delta t$ " (the intention of C2 is to maintain continuous contact with R1).
7. At 10/01/2014 23:05:30 the stipulated security time interval  $\delta t$  expires, and C2 receives no greeting signal from R1 (Up to this moment, greeting signals had been periodically received in time).

8. At 10/01/2014 23:35:30 C2 receives a greeting signal from an entity R which claims to be R1.

Thirty minutes is time enough for an attacker to capture R1 and replace it by an impostor. How can C2 be sure of the identity of R?

### **12.3. Context**

A dynamical system comprising uniquely identifiable artificial entities, endowed with processing, communication, and memory capabilities, such as robots or smart objects. The purpose of the system is to provide a service to an organization (e.g. security or logistic services). Moreover, the system is responsible for assuring its own security and monitoring its own performance.

An essential feature of the system is its dynamism. The system evolves over time, changing from one state to another, which gives rise to the system history. Two highly desirable properties of such a system are self-management (i.e. minimizing human intervention) and traceability of entities and processes. Self-management implies monitoring over time the performance of the system entities in order to allow early reaction in case of misbehavior or underperformance. Traceability can be used for suggesting process improvements, strengthens confidence in the entities and allows accounting for the provenance of valuable assets.

### **12.4. Problem**

For the sake of security, it must be assured that only legitimate entities are allowed to accomplish tasks related to the service and to access any kind of resource from the system. For this reason, the entities must be properly authenticated, that is, they must prove that they are what they claim to be. The system must be protected from attackers trying to impersonate legitimate entities. Different types of attack are possible, ranging from stealing identification and authentication information to cloning the entity. The realization of such threats could result in harms such as privacy loss, misuse of the system for malicious purposes, unreliability of the system, or system disablement.

The following forces apply:

- What the claimant has may be lost, damaged, stolen or cloned. To have deeper insight into the weaknesses of particular solutions, such as smart cards, we refer the reader to [2]. Moreover, due to costs or technical reasons, it may not be feasible to provide non-human entities with such devices.
- Though promising, the existing techniques aiming to authenticate artificial entities through biometric-like features (see, e.g., [3] or [4]), are not yet mature or widely applicable.
- The information required for authentication must be difficult to steal or forge.
- As a result of previous force, one static password is not sufficiently secure: it may be guessed by an attacker or discovered by eavesdropping. A set of passwords, possibly changing over time, would provide an extra level of security. But an entity should focus on its performance: generating new authentication information and communicating it to authentication authorities should not be a distraction from its work.

- The more comprehensive authentication is, the more difficult it becomes to circumvent it. If situation requires so (e.g. when trying to access critical resources) authentication should be as comprehensive as possible.
- Proportionality requirement: the resources assigned to a task should be proportional to its frequency and criticality level. Different contexts call for different levels of rigour in the authentication process.
- The entities responsible for authentication should not be able to impersonate an entity that has been authenticated by them (i.e. inside attacks should be prevented).
- Error and tampering threats: data corruption, malicious injection of wrong data or illegal manipulation of data stored by an entity should be detected early, if possible before providing the entity with access to the required resources. While this force is not strictly related to authentication, an authentication mechanism checking the integrity of data would provide an added value.

## 12.5. Solution

Base authentication upon the entity's history.

Forces F1 and F2 suggest ruling out two authentication factors (what an entity has, what an entity is) and focus on the remaining one: what an entity knows. The system has a history, which grows and becomes more complex as the system evolves. The enhanced capabilities of the entities allow them to be aware of their history and process queries related to it. Taking advantage of this fact, authentication can be based upon the entity's history. To be successfully authenticated, an entity should be able to answer a set of questions regarding its history. The structure of the inquiry could range from simple question-answer interchange to an exhaustive, extraordinarily complex interrogation (e.g. before providing the entity with access to a critical resource).

Some entities are specialized as authenticators. To exercise their role, they must be provided with adequate authority and capabilities, as well as knowledge about the history of the entities to be authenticated. In particular, the authenticator should be able to increase such knowledge by aggregating information regarding the entity provided by other relied-upon entities. In this way authentication benefits from the collective memory.

## 12.6. Structure

As previously stated, we consider a system of uniquely identifiable entities endowed with enhanced capabilities. This fact has been represented in the UML class diagram of Figure 22 with the class Entity, which has an attribute (Identity) and several methods. We restrict ourselves to the methods that model the capabilities supporting our mechanism such as communicate, which represents the communication capability of entities.

We understand the history of an entity as a set of elementary components, which are in turn modelled by the notion of occurrence. An occurrence is defined as something (an event, an incident or something else) that happens to one or several entities during a period of time. In our example, each of the events described in subsection 12.2 is an occurrence. We are aware that there is an even simpler concept defined as something that happens, without referring to any entity. This simpler concept is similar to the notion of event defined in the glossary [5] as “anything that happens, or is contemplated as happening”. However, because our goal is to

manage the entities' history, our atomic concept will be occurrence. Our current concept of occurrence generalises the one we introduced in [6].

An entity builds up its history by registering the occurrences in which it is involved. Specialised types of occurrences can be derived from the basic notion. A type especially relevant to our proposal is Communication, which models the registration of a communication act. The registered occurrence is “an entity has taken part in an act of communication whose content is the communicatedObject”. Two associations between Communication and Entity are used to indicate the role that each entity plays (sender or receiver).

In particular, an entity can communicate with another one of its occurrences so that, in this case, the communicated object is an occurrence. As a consequence, an entity can acquire (at least a partial) knowledge of the history of other entities; this feature is highly relevant for our authentication mechanism.

Two different types of entities can be differentiated in the authentication process, the entity that has to be authenticated (authenticable entity) and the entity that authenticates it (authenticator).

An authenticable entity is an entity capable of recording, recognising and reporting on its own history. In our proposal, the history of an entity enables it to get authenticated. Note that, because a history grows over time, it becomes more and more complex (in size and complexity) and thus more and more difficult to steal or forge. The strength of our mechanism is thus an increasing function of time. An authenticable entity should be able to generate an output in response to such an input. This output, hereinafter referred to as the proof of identity (token authenticator, according to [7]), proves that the entity knows the correct answers and serves to authenticate it to the system. The simplest case of a proof of identity is just the concatenation of the answers to the posed questions. Proofs of identity may be strengthened by further processing the answers through computation methods such as hashing functions, encryption with secret keys or zero knowledge proof techniques. We remark that an authenticable entity is inextricably linked to its history: if dissociated from it, the entity is no longer able to generate a proof of identity and thus ceases being an authenticable entity. An example of a history-aware entity would be a smart thing such as a spime [8], defined as a new category of space-time objects that are aware of their surroundings and can memorise real-world events; if provided with the capability to generate proofs of identity, a spime would be an authenticable entity.

On the other hand, an authenticator (verifier, according to [7]) is an entity endowed with both the necessary capabilities and the authority to authenticate other entities. To exercise its role, it must have access to the history of the entity to be authenticated. Such knowledge may be direct (derived from its own history) or communicated by other trusted entities.

We note that specialisation of Entity into AuthenticableEntity and Authenticator is incomplete and overlapping. Incompleteness permits the existence of other sorts of entities in the system, such as sensors. In addition, overlapping allows an authenticator to be, in turn, authenticated by another entity.

Our concrete proposal can be seen as a specialization of the Authenticator Abstract Security Pattern proposed in [9] since our proposal includes the Abstract Authenticator classes plus several other classes specific to our approach. Specifically, the Authenticator and Proof of identity classes of the abstract pattern appear explicitly in our proposal, the Subject class

corresponds to our AuthenticableEntity class and the Authentication Information class corresponds to the history associated to the authenticator.

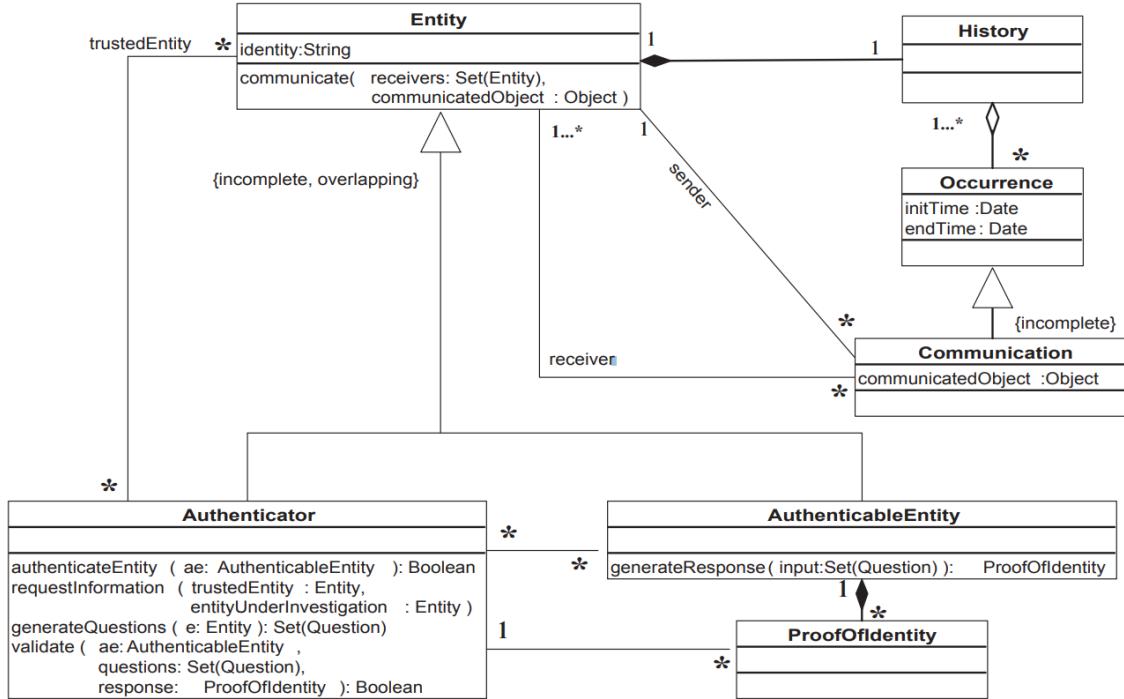


Figure 22: History-based Authentication Class Diagram.

## 12.7. Dynamics

There are two actors involved in an authentication process: an entity claiming to be AE and an authenticator Aut. The steps of the authentication process are described below, and shown graphically in Figure 23.

1. The authenticable entity AE identifies itself to the authenticator Aut by means of the method `communicate(Aut, identity)`.
2. To authenticate AE, Aut first must possess sufficient knowledge about the history of AE. Aut can rely on its current knowledge or gather information from trusted entities (i.e., demand information on AE to a trusted entity T E through `requestInformation(T, AE)` and receive a set of occurrences regarding AE through `communicate(V, Set(O))`). Depending on the centralised/distributed architecture of the system, such trusted entities could be a central repository, a set of credential service providers (whose credentials would authoritatively bind the identity of an entity and a subset of its history), or just trusted peers.
3. Based upon the collected knowledge, Aut generates a set of questions `Set(Q)` by invoking `generateQuestions (AE)`.
4. Such questions are posed to AE: `communicate(AE, Set(Q))`.
5. AE, in turn, generates a proof of identity (`generateResponse (Set(Q))`) and
6. returns it to Aut (`communicate(Aut, Response)`).
7. After validating the received proof of identity (`validate (AE, Set(Q), Response)`), Aut may reach a definitive conclusion on the authenticity of AE's identity or pose new questions, possibly after gathering further information from its trusted parties. To meet the proportionality requirement, the thoroughness of the questions posed

during the interrogation depends on the criticality of the information/services/location the entity intends to access and the cause that triggered the authentication process. Some of the new questions can be made depending upon the answers formerly provided by AE (e.g., questions can be raised to clarify an ambiguous answer or to obtain more detailed answers).

8. Finally, Aut reports on the success or failure of the authentication process to AE (communicate(AE, result)) . If the claiming entity's answers are right, the authentication is positive. Otherwise, if the claiming entity fails to prove its knowledge about the history of AE, the authentication is negative.

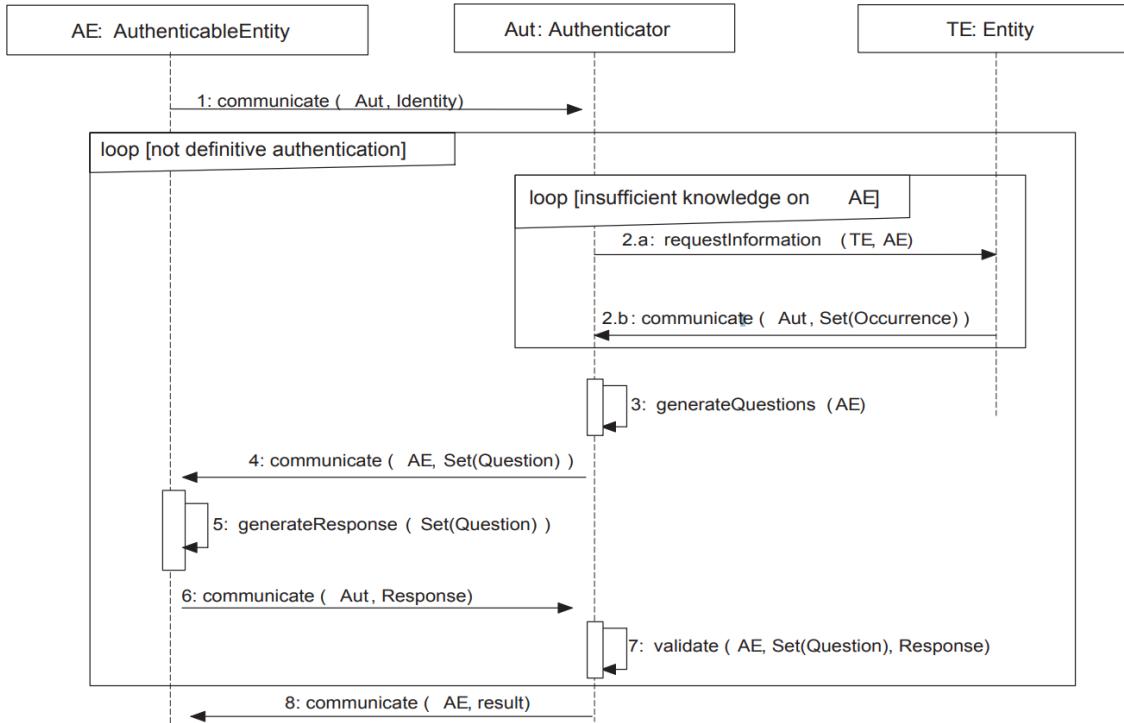


Figure 23: History-based Authentication Sequence diagram.

## 12.8. Implementation

In order to implement the pattern, authenticable entities must first be registered within the system. A registrar, on behalf of the system, issues an identity certificate to each new entity. As a consequence, an identity management model is needed, and unique names should be assigned to entities. Furthermore, if a new entity has a history prior to entering the system, the registrar should validate the identity proofs provided by the entity

During the authentication process an entity AE must not leak information about its occurrences to non-authorised parties such as eavesdroppers (history secrecy). Moreover, an eavesdropper should not be able to trace the history of the interactions between AE and the system. In particular, the link between AE and its communications (i.e. individuating the communications where AE has taken part) should not be established with a reasonable effort (unlinkability). A possible solution is to encrypt communications between authenticable entities and authenticators. Moreover, for the sake of performance, the answers provided by an authenticable entity may be hashed.

To perform its task, an authenticator does not need to know the whole history of an authenticable entity. Care must be taken to assure that an authenticator doesn't have such thorough knowledge (which would allow him to impersonate the entities he is supposed to authenticate).

According to the proportionality requirement, the complexity of each realisation of authentication must be proportional to the criticality of the circumstances. To meet this requirement both the different possible circumstances and authentication complexity levels should be categorised. Circumstances should be classified depending on the criticality of the information/services/location the entity intends to access. Furthermore, authentication complexity levels should be defined by establishing the thoroughness of the questions posed during the interrogation. Moreover, a policy is required to assign minimum and standard authentication complexity levels to each circumstance.

Persistence structures for storing the occurrences must be set up, as well as a language for communicating occurrences (including its syntax and semantics).

## **12.9. Example Resolved**

Each of the events described in subsection 12.2 is an occurrence, and has been recorded by the entities involved in it. Robots are required to identify and authenticate themselves to the system, thus they are authenticable entities, as described in Section 12.7. Because controllers should be able to authenticate the entities under their control, they are a type of the authenticators described in Section 12.7.

To authenticate R, C2 asks its trusted entities if any of them has ever worked with R. Among others, C1 answers affirmatively. C2 asks C1 further details of the history of R1. C1 signs a credential binding the identity of R1 to the set of occurrences (1) to (4) (see subsection 12.2), and sends it to C2. For the sake of security, the credential can be encrypted using C2's public key. Provided with such information, C2 asks R1 "What temperature did you report at 06/12/2013 10:45:00 ?". AE provides the right answer. C2 goes on asking: " When was such temperature measured? Who was your controller in that occasion? Who were your patrol partners ? ". R1 fails to answer satisfactorily the last set of questions. C2 reacts to this situation by putting AE into quarantine and transferring the problem to a specialised subsystem that, after performing in-depth hardware and software testing on AE, decides the subsequent step (e.g., repair or retire R2).

## **12.10. Consequences**

Authentication is reinforced. Confidentiality, integrity, authenticity, accountability and non-repudiation are strengthened over time as the history increases. The system has been endowed with an authentication mechanism that exploits the knowledge about its history. To get authenticated, an entity C claiming to be AE needs to prove an up-to-date knowledge of the history of AE. Failing to provide a satisfactory proof of identity, based on the answers to the questions posed by the authenticator, is a sign that there is something wrong with C: inability to account for its own history may be due to a communication error (either in the reception of the request made by Aut or in the transmission of the message), memory corruption (C is really AE, but has forgotten part of its history) or the entity being an impostor impersonating AE.

#### Benefits:

- Is fully applicable to artificial entities, provided they have sufficient storage and computational capacities.
- Enforces the prevention of impersonation. Even though an attacker has forged the identity certificate of an entity, it still must capture a significant part of the history of the entity. As time goes by, this becomes an increasingly overwhelming task. This consequence matches force F3. In particular this can be applied to authenticators: since a single authenticator does not know the complete history of an entity AE and, moreover, does not know in advance what questions may be posed to AE by another authenticator, insider attacks are prevented. This matches force F7.
- Resorts to information that comes out naturally with the entity performance. It does not require an extra effort to generate such information. This consequence matches force F4.
- Allows dynamism and flexibility: the authenticator can choose in any moment which and how many questions an entity must answer to be authenticated. The number and nature of questions can be adapted to the criticality of the resource an entity intends to access or to its observed behaviour. If an intrusion detection system raises suspicion about an already authenticated entity, a more exhaustive authentication process may be triggered. This consequence matches forces F5 and F6.
- Provides a basis for error tampering detection: inability to provide satisfactory answers to questions related to history may be an indication of integrity loss. This consequence matches force F8.
- Allows collaboration: an authenticator can complement its knowledge on the entity's history by consulting other trusted 'colleagues'. Such a feature is highly relevant in decentralised environments such as the IoT.

#### Liabilities:

- Need for storage and some computational capability in each entity. The storage resources needed increase as time goes by.
- Manageability: The overhead required to maintain the history of the entities implies a larger impact on manageability than other, simpler authentication technologies, such as passwords. For this reason, a trade-off between manageability and the required security level should be considered. Being aware of this issue, we establish the requirement of proportionality.
- Performance: Performance is affected by the need to store and manage the history of the entities. In general, choosing more secure procedures will generally have an adverse effect on performance.
- Cost: Costs will be incurred from the additional processing power required and the additional management overhead. However, this investment could cost less in the long term, because the system will be less vulnerable to attacks, which are inherently expensive to detect and recover from.

### 12.11. Known Uses

The idea of benefiting from the knowledge about an entity's history to verify its identity is not new. It has been extensively applied to persons over time. Moreover, such an idea underlies several solutions proposed in the literature. For example, in [10], mobile sensors in a wireless

network authenticate each other based on the history of their previous contacts. In [11] a history-based mechanism has been proposed for robot authentication: a robot builds its own personal profile, by tracing selected data regarding the robot's sensors and actuators. The robot communicates such a profile to a Trusted Authority TA, that can make use of its knowledge of the personal profile in future authentications. In [12], a person is authenticated based on the history of his interactions with his smartphone.

We have followed this approach in two development projects. AWHS Project [6] aimed to support the collaborative management of biobanks collecting, storing, processing and lending biological samples. In order to satisfy the requirement for traceability, the objects (biological samples and their aliquotes) were augmented with their history, to which they remained inextricably linked. If an object is dissociated from its history and it is not possible to assure its provenance, it ceases to be acceptable for the purposes of the biobank. The history of an object was defined as the set of all occurrences related to it: how it was created, what changes it has experienced, what other entities it has been related to, what processes it has been subject to (e.g. sample aliquoting, box storage) and what protocols have followed such processes. The ability to build up the whole history of an object allowed to set up a "chain of custody" (the object provenance). Among other outcomes, it would allow to undoubtedly assure the object's identity, i.e. to authenticate it.

Project THOFU is a multidisciplinary collaboration intending to perform a prospective study on technologic concepts and solutions enabling advanced services in the context of the hotels of the future [1]. One of the main research areas of THOFU is related to security. A pervasive security system was designed to operate in a self-managed way. The system benefits from the knowledge about its own history to assure its correct performance and its compliance with requirements such as privacy and security.

## 12.12. See Also

Our pattern is related to the Authenticator Pattern, which aims to authenticate distributed objects. Furthermore, our concrete proposal is a specialization of the Authenticator Abstract Security Pattern [9].

## 12.13. References

- [1] Thofu: Technologies of the hotel of the future. (2013). Thofu. Retrieved January 2014, from <https://thofu.es/>
- [2] Madhusudhan, R., & Mittal, R. C. (2012). Dynamic ID-based remote user password authentication schemes using smart cards: A review. *Journal of Network and Computer Applications*, 35(4), 1235-1248.
- [3] Suh, G. E., & Devadas, S. (2007, June). Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference* (pp. 9-14). IEEE.
- [4] Cobb, W. E., Laspe, E. D., Baldwin, R. O., Temple, M. A., & Kim, Y. C. (2011). Intrinsic physical-layer authentication of integrated circuits. *IEEE Transactions on Information Forensics and Security*, 7(1), 14-24.

- [5] Luckham, D., & Schulte, R. (2011). Event processing glossary-version 2.0. Real Time Intelligence & Complex Event Processing. Retrieved May 2014, from <https://complexevents.com/2011/08/23/eventprocessing-glossary-version-2-0/>
- [6] Domínguez, E., Pérez, B., Rubio, Á. L., Zapata, M. A., Lavilla, J., & Allué, A. (2013). Occurrence-oriented design strategy for developing business process monitoring systems. *IEEE Transactions on Knowledge and Data Engineering*, 26(7), 1749-1762.
- [7] Burr, W. E., Dodson, D. F., & Polk, W. T. (2011). *Electronic authentication guideline* (pp. 800-63). US Department of Commerce, Technology Administration, National Institute of Standards and Technology.
- [8] Sterling, B., & Things, S. (2005). Shaping Things. The MIT Press.
- [9] Fernandez, E. B., Yoshioka, N., Washizaki, H., & Yoder, J. W. (2014, April). Abstract security patterns for requirements specification and analysis of secure systems. In *WER*.
- [10] Zhu, W. T., Zhou, J., Deng, R. H., & Bao, F. (2012). Detecting node replication attacks in mobile sensor networks: theory and approaches. *Security and Communication Networks*, 5(5), 496-507.
- [11] Adi, W. (2009, August). Mechatronic security and robot authentication. In *2009 Symposium on Bio-inspired Learning and Intelligent Systems for Security* (pp. 77-82). IEEE.
- [12] Das, S., Hayashi, E., & Hong, J. I. (2013, September). Exploring capturable everyday memory for autobiographical authentication. In *Proceedings of the 2013 ACM international joint conference on Pervasive and ubiquitous computing* (pp. 211-220).

## 12.14. Sources

Ciria, J. C., Domínguez, E., Escario, I., Francés, Á., Lapeña, M. J., & Zapata, M. A. (2014, July). The history-based authentication pattern. In *Proceedings of the 19th European Conference on Pattern Languages of Programs* (pp. 1-9).

# 13. Fine-Grain Access Control

## 13.1. Intent

A holistic access control to business objects is not sufficient as business objects are often comprised of multiple parts (fields) and may contain references to other business objects. Objects need to be designed with built-in support for fine-grained access control to their elements. Consequently, we need a model which provides a fine-grain access-controlled business objects. In this pattern we present an approach for access control to an object and to its individual attributes and operations.

## 13.2. Example

Consider a simple customer business object, which consists of customer name, address, telephone, email, credit card info, and ordering history.

The customer data should only be available to individuals who need it and have the right (permission) to access it. Most of the time, only parts of the customer's data are needed by those who access it. For example, a marketing department might need to see the customer's order history and contact info, but they should not be able to see the customer's credit card info. On the other hand, the ordering department should have access to the customer's credit card info in order to be able to validate it but might not have a valid business reason to see the order history.

Access to the customer object and its fields could be programmed into the object itself. This approach however is not desirable as it does not consider constantly changing business conditions and processes. Any changes to the way customer object are accessed and controlled would require additional programming, which is often expensive and time consuming.

An alternative is to associate permissions with the customer object. Thus, for example, a Marketing Department permission can be associated with the customer business object, and its contact info and ordering history details. At the same time, an Ordering Department permission can be associated with the customer business object's credit card details. Both permissions are associated with the customer object, but each propagates to a different set of its fields, thus each permission provides a fine-grain access control over the customer object.

## 13.3. Context

You are designing a new business object and need a way to provide fine-grain access control to the object and its fields when your business object is deployed in your business application. The fine-grain access control should be configurable and verifiable. Finally, the fine-grain access to the business object should also provide control over the business object's fields visibility and editability

## 13.4. Problem

Determining business object's access control and fine-grain accessibility of its fields at design time or programmatically is not always feasible or possible. How can I design a business object's fine-grained access control which can be parameterized, and thus configurable at run time or at the time of deployment?

## 13.5. Solution

The UML class diagram for the pattern's design is shown in Figure 24. The business object metamodel is suitable for different runtime environments (such as web client, web backend, web services, Java virtual machine). It includes the following concepts:

- BusinessDataObject class: for business objects. Such an object is uniquely identified and all objects from the same class have similar states and behavior, as per the object-oriented paradigm. A class state is defined by its attributes (i.e. properties) and an object is capable to execute a set of operations.
- Attribute class: an object of this class describes the name and type of a business object's attribute. The runtime value of an attribute is not addressed in this pattern. A BO may have zero or more Attribute objects, one for each of its attribute.
- Operation class: each operation supported by a BO is described by an Operation object. Its attributes may include name, return value, parameter list.

Access rights and permissions are represented in this design by corresponding classes for BOs, their attributes, and operations. A BO instance has an optional DataAccessRightsDescriptor object that represents the rights assigned to the BO in an abstract and opaque way. An Attribute object is also associated with an optional DataAccessRightsDescriptor instance, describing the access rights endowed to this particular attribute. The OperationRightsDescriptor class describes the rights assigned for a particular operation for a BO, again in an abstract way.

The Subject class represents the entity that access a BO's attributes and executes its operations. A Subject's access permissions are described by a PermissionsDescriptor object.

The Runtime class abstracts the runtime system (e.g. web application server, JVM, browser JavaScript engine) and aggregates subjects and BO instances.

The PermissionAuthority object is responsible for authorizing access by Subjects to BOs, attributes, and operations depending on the access rights descriptors associated for a particular access attempt by a Subject. The PermissionAuthority checkDataAccess() and checkOperationAccess() methods implement the validation for access to a BO's attribute and operation, respectively, and encapsulate policies for dealing with defaults and conflicts.

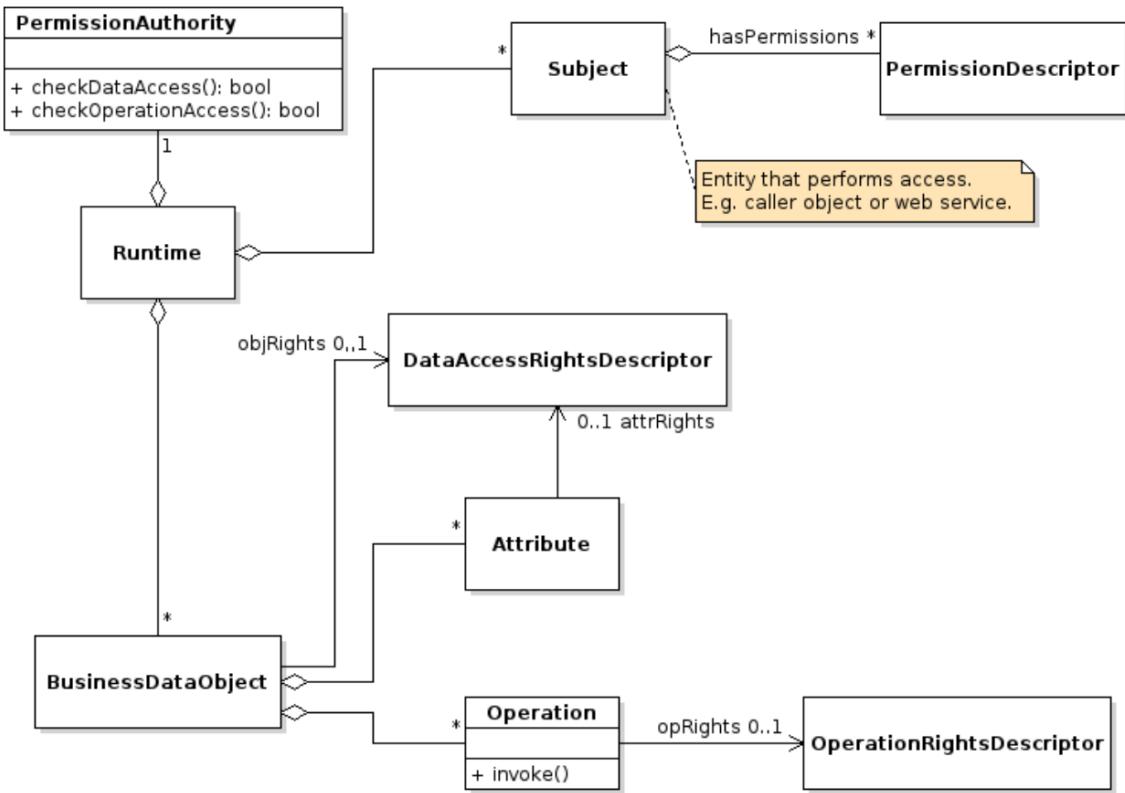


Figure 24: The design of the Fine Grain Access Control for Business Objects pattern.

## 13.6. Structure

## 13.7. Dynamics

## 13.8. Implementation

## 13.9. Example Resolved

## 13.10. Consequences

## 13.11. Known Uses

Most, if not all major Commercial Off The Shelf (COTS) Enterprise Resource Planning (ERP) developers such as Oracle and SAP to name the two biggest, employ some sort of permission-

based access to their data forms and data objects. They too apply permission-based access at both the full object level, as well individual object fields. Ariba for example uses permission objects which can be assigned to role objects, which in turn are then assigned to user objects. Developers can then assign permissions to actions such as editability and visibility of the fields within a given object. This allows Ariba business objects to be accessed via permissions, without the need for further coding

## 13.12. See Also

The Discretionary Access Control (DAC) pattern enforces access control based on user identities and the ownership of objects. The owner of an object may grant permission to another user to access the object, and the granted user may further delegate the permission to a third person.

The Mandatory Access Control (MAC) pattern governs access based on the security level of subjects (e.g., users) and objects (e.g., data). Access to an object is granted only if the security levels of the subject and the object satisfy certain constraints. The MAC pattern is also known as multilevel security model and lattice-based access control.

The Role Based Access Control (RBAC) pattern enforces access control based on roles. A role is given a set of permissions, and the users assigned to the role acquires the permissions given to the role. Since the RBAC pattern is based on roles which are in general fewer than the number of users, it is useful for managing a large number of users.

Attribute-based Access Control (ABAC) defines an access control architecture whereby access rights are granted to users through the use of policies which combine attributes together. The policies can then use any type of attributes (user attributes, resource attribute, etc...). Attributes can be compared to static values or to one another thus enabling relation-based access control. [1]

The Semantic Access control (SAC) model was created in 2002. The fundamentals of this semantics-based access control model are the definition of several metadata models at different layers of the Semantic Web. Each component of SAC represents the semantic model of a component of the access control system. The semantic properties contained in the different metadata models are used for the specification of access control criteria, dynamic policy allocation, parameter instantiation and policy validation processes. [2]

## 13.13. References

[1] Yuan, E., & Tong, J. (2005, July). Attributed based access control (ABAC) for web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE.

[2] A Semantics-based Access Control Model for Open and Distributed Environments. (n.d.). Semantic Access Control. <http://www.lcc.uma.es/%7Eyague/Semantics-basedAccessControl.html>

## 13.14. Sources

RUBIS, R., & CARDEI, D. I. (2014). Patterns for fine-grain access-controlled business objects. PLoP.

# 14. Rights Based Fine Grain Access Control

## 14.1. Intent

A basic approach for access control that requires granular permissions to objects, attributes, and operations in environments with a manageable number of subjects.

## 14.2. Example

## 14.3. Context

Business objects and their attributes require rights for specific types of access, such as create/read/update/delete and copy. The access right for an operation is to execute it. The environment makes it feasible to assign access rights to subjects for specific objects, attributes, and their operations. This could be an execution runtime (OS), a virtual machine, or a distributed (web) system not open to external clients, for instance.

## 14.4. Problem

Access to Business Objects, their attributes, and operations must be given individually to subjects in a granular way. A subject may be given access to an object, but not to all its attributes or operations. Also, permissions to read, update, copy, create, and delete must be differentiated – per attribute. The system must apply policies to deal with situations where a subject's rights for an object conflicts with the specific access rights of that subject for the required attribute/operation. For instance, a subject that has the Read access right for an object may lack the Read right for an attribute it needs to read. One incarnation of this pattern may apply the principle of least privilege and deny access.

## 14.5. Solution

The design in Figure 25 assigns RightsDescriptor objects to a subject that describe specific access type allowed for the subject to individual objects, their attributes, and operations. An optional RightsDescriptor object is assigned to each BO, attribute, and operation to describe access rights that apply for all subjects that request access. The PermissionAuthority object validates access by matching the rights a subject has for a specific object, attribute, or operation with the (optional) rights descriptor owned by corresponding object, attribute, or operation. Validation must consider cases when a subject does not have access rights objects for the requested access and when the object (or attribute/operation) has no rights descriptor. Best practices (assigning least privileges) can be applied.

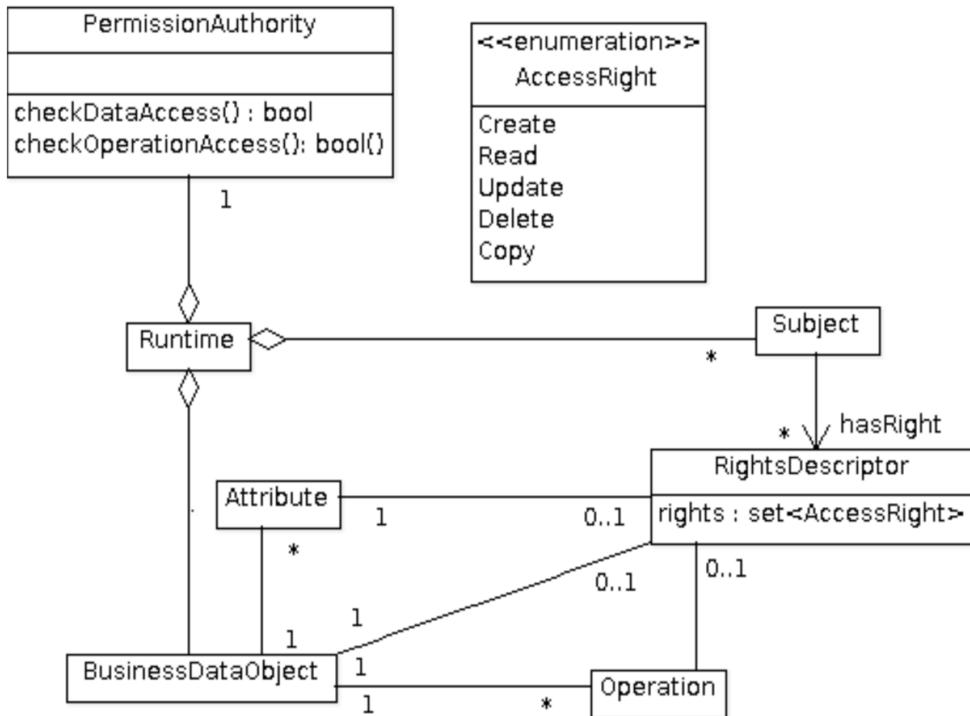


Figure 25: The class diagram for the Rights Based Fine Grain Access Control pattern

There are 5 access types which control how an instance of a business object can be accessed and what the accessor can do to that instance of a business object. The basis for the access types has been borrowed from the database CRUD operations (Create, Read, Update, Delete). The 5 types applicable to the common business objects instances are CRUDC (Create, Read, Update, Delete, Copy).

For each business object we can map the five access types to permissions which controls the particular access type. This enables us to have a fine-grain access control over any business object. Furthermore, the actual mapping does not have to take place at design time and can be performed at the time the business object is deployed to the business application.

At design time we create each business object with a set of attributes which serve as place holders for the actual permission mappings when the business object is deployed.

Thus, it is expected that each business object will have 5 permission attributes:

- **Create Permission**. A permission value mapped to this attribute indicates that users with this permission are allowed to create the given business object. Creating a new instance of a business object should not be a widely open process available to everyone. Only certain users should be allowed to create new instances of business objects, and this attributes would hold a set of permission values which allow the Create operation.
- **Read Permission**. Another word for reading an object is viewing it. As business objects are comprised of business data, the ability to view their contents is essential. However, there are numerous cases when the business objects should only be viewed for certain group of people and should not be visible to anyone outside of that group. For that reason, the Read Permission attributes allows for mapping the object's visibility (readability) to a permission or a set of permissions. By the same token, there are times when an object should be visible to a wide set of audience, but some of its fields

should be hidden from that audience, and only available to a small group of people. A social security number on an employee object or a credit card info on a customer object are examples of such fields. In those cases, we can map a read permission(s) to individual fields, which would override the Read permission(s) at the object level.

- **Update Permission.** Updating business object is a daily occurrence within any organization. For example, an employee's marital status changes and needs to be reflected in the employee object, or a customer address changed, and needs to be updated in the customer object. However, the ability to update business objects should not be granted to everyone. A clerk in the warehouse should be able to view customer address for shipping purposes but should not be able to update it. The update access control can be solved via permissions. By mapping business object's Update Permission attribute to a permission (or a set of permissions), we can control who can update the given business object. We can also apply more fine-grain control to business fields by applying a different permission (or a set of permissions) to those fields which should only be accessible for update to a narrower audience (see example in Table 1).
- **Delete Permission.** Deleting a business object is not the same operation as deleting the object in a programming language. Deleting a business object may require deleting some language objects, deleting some database entries, etc. Deleting a business object should be the least used action, and least accessible action, but it is an extremely sensitive one. Deleting business objects is highly discouraged and should only take place in rare circumstances. Some business applications, such as Ariba, do not allow any deletion of business objects. Instead, they "deactivate" unwanted business objects within the application, thereby rendering them unavailable to the end users and processes. Even in the case of deactivation (logical delete), only a very narrow set of audience should have ability to delete business objects. Mapping a permission to Delete Permission attributes will accomplish that limited access. Delete operation can also be applied at business object field level. Note: in this pattern delete of field data is different from updating of field data. Updating field involves changing field value from one value to another (i.e. changing employee marital status from Single to Married). Deleting field value is removing any value within that field and leaving it blank (or null or null).
- **Copy Permission.** Copy is probably the least utilized action in most business applications yet could greatly increase user productivity and decrease data errors. For example, when reordering office supplies, it would be much quicker to copy an existing order and just modify item quantities as opposed to creating a new order from scratch, especially if the order has a large number of items. Not only does copy offer much faster way to order the needed supplies (user productivity), it also eliminates errors which could have been introduced has the order been created from scratch. At the field level copy is most often utilized when a business object contains data of Master/Detail type. Copy can be used to copy a detail field (which often contains numerous fields).

All object level permissions are propagated by to the attributes and operations of the object. At the attribute and operation level, these permissions can be overridden by explicitly assigning another permission to an attribute or operation.

At design time we do not need to assign any value to each of the above attributes. Rather, each attribute will be mapped to a permission via parameter.

The same set of attributes is also added to individual fields within the business object at design time. The attributes at the object level and at the object field level allow for a fine-grain control over the business object and its fields at runtime, without the need to know the actual permissions at design time.

The following table provides an example of how permissions can be mapped to a business object and its fields. We use “/” to denote object/field relationship. For example, a customer object and customer credit card would be denoted as Customer/CreditCard.

	Customer Object	Customer/CreditCard Field	Customer/Telephone Field
Create Permission	CustomerService		
Read Permission	CustomerService or Finance	Finance	
Update Permission	CustomerService or Finance	Finance	
Delete Permission	Finance		
Copy Permission	CustomerService		

*Table 1: Permission-based field access*

The Above table shows permission mappings for Customer object and its fields. Users with CustomerService permission are allowed to create, read, update, and copy a customer object, however only users with Finance permission can delete a customer object. Because Customer.Telephone field does not have any permissions assigned explicitly, it inherits the access permissions of the object itself. The field Customer/CreditCard on the other hand can only be updated or viewed by users with Finance permission. Consequently, users with CustomerService permission may enter customer credit card info at the time of customer business object creation, but do not have access to updated or view customer credit card info subsequently.

Thus, via permissions we can provide a fine-grain access control to a business object as a whole, and to its individual fields.

The permissions associated with the given business object access do not have to be determined at design-time. Rather, the mapping of permissions to objects should be parameterized, thereby enabling the creation of the associations at run-time. Assigning permissions at run-time also enables us to group the business objects based on the permission(s) to which they are mapped.

As our intention is to provide fine-grain access control to business objects, we need to define additional permissions for the object attributes and the object operations. These are different from the object permissions but must be related to them at least by a default propagation scheme. Object attributes will have these permission attributes: Create Permission. A permission value mapped to this attribute indicates that users with this permission are allowed to create the given business object attributes. This is only possible for those attributes that allow this by their structure.

Read Permission. Another word for reading an object attribute is viewing it. A permission to read an operation allows access to the history of operation calls. This is a common and useful action in business objects. For instance, consider an account object. It is frequent that there is a need to consult the account transactions.

**Update Permission.** Updating business object attribute is a daily occurrence within any organization. For example, using the example above, an employee's marital status changes and needs to be reflected in the employee object, this action needs permission to update the object, as well as permission to update the "MaritalStatus" attribute.

**Delete Permission.** Deleting a business object attribute is only possible for those attributes that allow this by their structure.

As for operations, we also need specific permissions that are different from the object permissions but must be related to them at least by a default propagation scheme. Object operations will have these permission attributes:

**Call Permission.** This is a permission that allows invoking the given operation on the given object instance. Calling an operation requires a permission for updating the object instance, plus a permission to call the operation.

Finally, we should define default propagation rules for permissions so that when an object is assigned permissions the attributes and operations are consistently assigned their corresponding permissions.

Because we can group permissions into roles, we can easily incorporate our configuration into an existing Role Based Access Control (RBAC) type architectures.

## **14.6. Structure**

## **14.7. Dynamics**

## **14.8. Implementation**

## **14.9. Example Resolved**

## **14.10. Consequences**

This pattern is suitable for systems where it is feasible to assign all subjects rights descriptors for objects they would access, including for their attributes, operations. For each object (and each of its fields) it only requires optional specification for just one rights descriptor that applies to all subjects. This mechanism is simpler, but gives less control compared to the role-based alternative, and alsoe does not scale with a large number of subjects.

## **14.11. Known Uses**

## **14.12. See Also**

The Discretionary Access Control (DAC) pattern enforces access control based on user identities and the ownership of objects. The owner of an object may grant permission to another user to access the object, and the granted user may further delegate the permission to a third person.

The Mandatory Access Control (MAC) pattern governs access based on the security level of subjects (e.g., users) and objects (e.g., data). Access to an object is granted only if the security levels of the subject and the object satisfy certain constraints. The MAC pattern is also known as multilevel security model and lattice-based access control.

The Role Based Access Control (RBAC) pattern enforces access control based on roles. A role is given a set of permissions, and the users assigned to the role acquires the permissions given to the role. Since the RBAC pattern is based on roles which are in general fewer than the number of users, it is useful for managing a large number of users.

Attribute-based Access Control (ABAC) defines an access control architecture whereby access rights are granted to users through the use of policies which combine attributes together. The policies can then use any type of attributes (user attributes, resource attribute, etc...). Attributes can be compared to static values or to one another thus enabling relation-based access control. [1]

The Semantic Access control (SAC) model was created in 2002. The fundamentals of this semantics-based access control model are the definition of several metadata models at different layers of the Semantic Web. Each component of SAC represents the semantic model of a component of the access control system. The semantic properties contained in the different metadata models are used for the specification of access control criteria, dynamic policy allocation, parameter instantiation and policy validation processes. [2]

## **14.13. References**

- [1] Yuan, E., & Tong, J. (2005, July). Attributed based access control (ABAC) for web services. In *IEEE International Conference on Web Services (ICWS'05)*. IEEE.
- [2] A Semantics-based Access Control Model for Open and Distributed Environments. (n.d.). Semantic Access Control. <http://www.lcc.uma.es/%7Eyague/Semantics-basedAccessControl.html>

## **14.14. Sources**

- RUBIS, R., & CARDEI, D. I. (2014). Patterns for fine-grain access-controlled business objects. PLoP.

# 15. Whitelisting Firewall

## 15.1. Intent

We want to prevent the client to get access to an external site or service (any kind of IP address or port) that is considered untrustworthy, or to stop traffic from an untrusted site. whitelisting firewall (WLF) defines a list of sites with which we want to communicate.

## 15.2. Example

## 15.3. Context

Nodes connected to the Internet and to other networks who need to enforce their policies to all of the sites as one layer in a defense in depth strategy in their site.

## 15.4. Problem

Some sites are insecure and may contain malware, which could attack our host or download malware if we visit them. How to prevent or stop the traffic of a system so we do not get access to external sites that are considered untrustworthy?

The solution will be affected by the following forces:

- Security: We handle sensitive assets. We only want to communicate with sites that are known by the user and are trusted. This will increase security.
- Transparency: The users of the system should not need to perform special actions, or they may not use the filtering.
- Overhead: Filtering should not significantly reduce performance.
- Usability: It should be easy to apply and manage filtering policies.
- Number of interlocutors. The number of sites with which we want to communicate is reasonable small.

## 15.5. Solution

Use a Whitelisting Firewall which is a filtering mechanism that can enforce communication with only approved sites by keeping a list of acceptable interlocutors (hosts or sites).

## 15.6. Structure

The class diagram for the Whitelisting Firewall is shown in Figure 26. The Whitelisting (WL) Firewall intercepts the traffic between a LocalHost and a set of ExternalHosts. The WLFirewall includes a Whitelist, which is composed of a set of ordered rules. The ordering of rules accelerates checking by avoiding checking of sites included in site groups. Some of the rules may be ExplicitRules or may be ImplicitRules (default) rules, which apply to all sites.

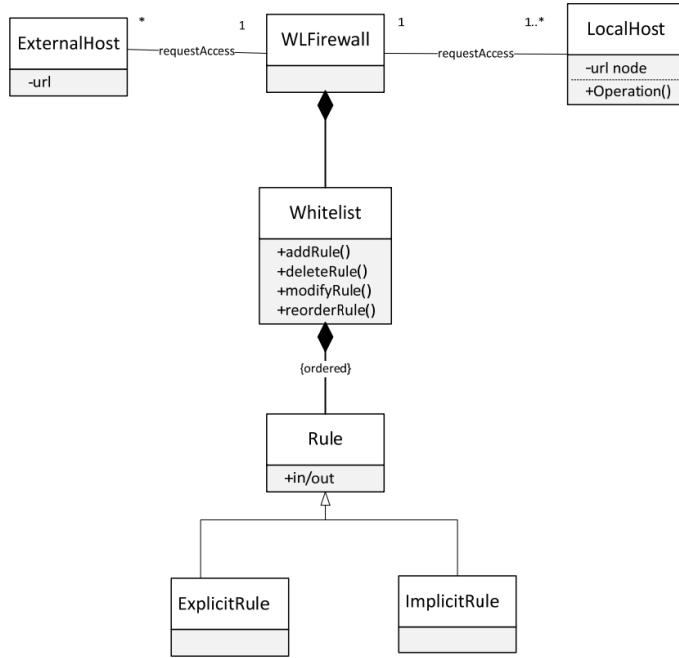


Figure 26: Class diagram for the Whitelisting Firewall pattern.

## 15.7. Dynamics

We describe the dynamic aspects of the Whitelisting Firewall using a sequence diagram for one of its basic use cases. Basically, the Whitelisting Firewall applies the policy of a closed world, where every address is denied access unless explicitly permitted. Figure 27 shows a sequence diagram for the Use Case "Filter an incoming request". A node requests to run some service or application. The Whitelisting Firewall receives this request and using a list of known and trusted sites checks if the request comes from or goes to a trusted site and allows the connection or denies the request.

Use Case:	Filtering an Incoming Request – Figure 27
Summary	A host in a remote network wants access to a site to either transfer or retrieve information. The access request is made through the firewall, which according to its set of rules determines whether to accept or deny the request.
Actors	A host in an external network trying to access a site (local host).
Precondition	A set of rules to filter the request exist in the whitelist of the firewall.
Description	<ol style="list-style-type: none"> <li>An external host requests access to the local host.</li> <li>A whitelisting firewall filters the request according to its rules to accept or deny the access.</li> <li>If the request is accepted, the firewall allows access to the site.             <ul style="list-style-type: none"> <li>If no rule allows the access, the request is denied.</li> </ul> </li> </ol>
Alternate Flows	
Postcondition	The firewall has accepted the access of a trustworthy site to the Localhost.

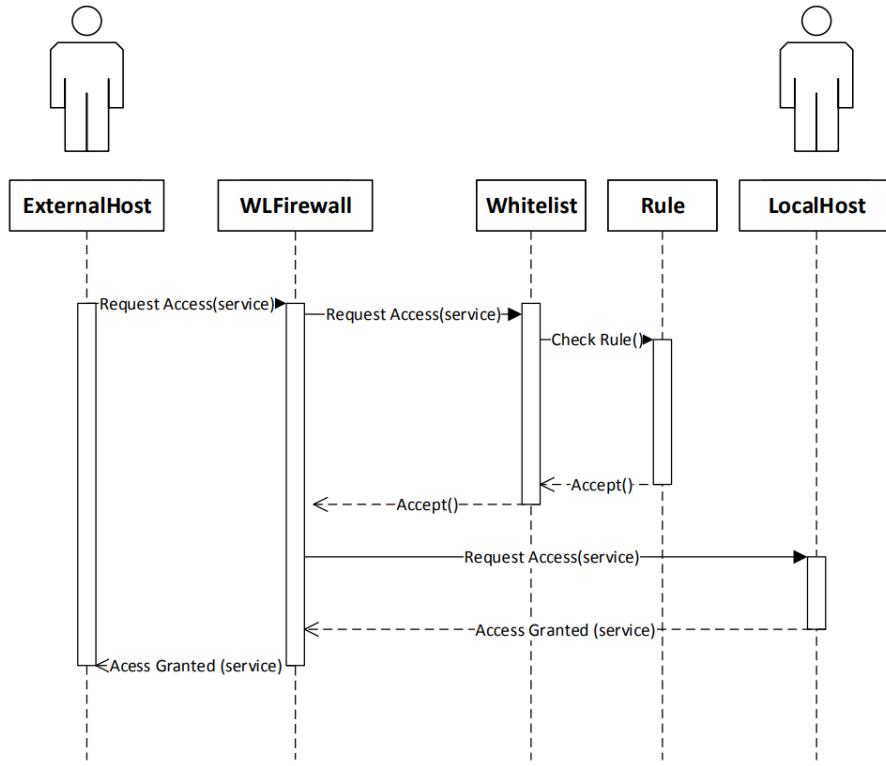


Figure 27: Sequence diagram for the UC: Filter an incoming request.

## 15.8. Implementation

The Whitelisting Firewall should reside on a host computer and maintain a local set of rules, which are maintained by a security administrator. The institution must define these rules according to their policies. The whitelist is usually rather short and there is no need for hardware assistance to search it and the filtering should happen at the IP layer. It is possible to filter also at the application layer and stateful firewalls could accelerate filtering.

There are specific approaches to use with the Whitelisting Firewall such as Gold Image and Digital Certificates. With the Gold Image for static systems, the list is created by first hashing a standard workstation image. After using an image to build the first whitelist, keeping it up to date will be the biggest challenge facing most groups.

On the other hand, digital certificates are one of the most effective techniques to trust certain publishers of software. Based on their signature the certificates are automatically in the whitelist and are considered trusted [1].

## 15.9. Example Resolved

## 15.10. Consequences

This pattern has the following advantages:

- Security. Whitelisting is more secure than blacklisting because we only deal with trusted sites.
- Transparency: Filtering is transparent to the users.
- Overhead: Checking is faster because whitelists are shorter than blacklists.
- Usability. The list is short and only requires updates when a new site is added.

This pattern has the following disadvantages:

- Annoyance. It may cause some users to be annoyed because they cannot download applications from anywhere or visit sites not in the list.
- Effect of address changes. If a trusted site in the whitelist changes its address, we need to update the list.
- Wolf in sheep clothes. A trusted site may still be harmful.
- Breadth. It is inappropriate if we need to offer products or services and we want to reach a wide audience.
- Number of interlocutors. If the number of sites with which we want to communicate is large, this may not be a convenient approach.

## **15.11. Known Uses**

- Bit9 Parity: It uses a software registry, a locally installed management server and a client to enforce software policies throughout the enterprise. Bit9 developed an adaptive whitelist strategy. The proprietary Global Software Registry is an online index. This list contains unique applications and the registry acts as a reference library for IT administrators building their whitelists [2]. Bit9 has also built a Parity Suite based on the qualities of transparency, flexibility and scalability [3].
- Coretrace Bouncer: It contains a standard file-based whitelisting protection mechanisms [4].
- McAfee: This centrally-managed whitelisting solution uses a dynamic trust model and security features that try to control advanced persistent threats [5].

## **15.12. See Also**

- Packet Filter Firewall pattern: Filter traffic from untrusted sites based on blacklists.
- Credential pattern: Provide secure means of recording authentication and authorization information for use in distributed systems.

## **15.13. References**

[1] SANS Institute. (n.d.). Application Whitelisting: Panacea or Propaganda? SANS. Retrieved November 17, 2012, from  
[http://www.sans.org/reading\\_room/whitepapers/application/application-whitelisting-panacea-propaganda\\_33599](http://www.sans.org/reading_room/whitepapers/application/application-whitelisting-panacea-propaganda_33599)

[2] Bit9. (n.d.). Bit9 Security Platform. Retrieved November 17, 2012, from  
<http://www.bit9.com>

- [3] EMA Corporation. (2012, October). Realistic Security, Realistically Deployed: Today's Application Control and Whitelisting. Bit9. Retrieved November 17, 2012, from <http://www.bit9.com>
- [4] Coretrace Corp. (n.d.). Coretrace products. Coretrace. Retrieved November 17, 2012, from <http://www.coretrace.com/products-2/bouncer-overview#application>
- [5] McAfee. (n.d.). McAfee Application Control. McAfee Products. Retrieved March 21, 2012, from <http://www.mcafee.com/us/products/application-control.aspx>

## **15.14. Sources**

Villarreal, I. N. B., Fernandez, E. B., Larrondo-Petrie, M., & Hashizume, K. (2013). A Pattern for Whitelisting Firewalls (WLF). PLoP.

# 16. Trust Trap Mitigation

## 16.1. Intent

As human beings, we naturally vary our scrutiny of others based on the level of emotional trust we have in them. We spend longer studying the appearance of a stranger on our doorstep than that of a good friend. Unfortunately, lessened scrutiny of those we trust can blind us to the precursors of betrayal. This pattern attempts to solve that problem in organizational settings.

## 16.2. Example

Joe is a mid-twenties high school graduate planning on a career in government service. He takes a job as a clerk in the city traffic court. His pleasant personality and eagerness to ingratiate him please soon with his supervisor, Tracy, and the other staff, and before long he receives administrative access to the traffic fines database and 24-hour access to the courthouse.

Joe is a very social person who likes to hang out with his friends in the evening. One night his friend Tom is furious at the \$250 traffic fine he has been given for driving 120 km/h in a 70 km/h zone. Joe casually mentions that perhaps he could take care of the problem for Tom. On his way home, Joe swings by the courthouse, accesses the traffic fine database, and changes the status code on Joe's ticket to "no fine."

Word of Joe's feat spreads, and soon Joe is doing bumper business fixing fines for many of the regulars where Joe and Tom hang out.

## 16.3. Context

Insider threat issues frequently involve two distinct trust domains: intentional and operational. The intentional view focuses on the mental state of individuals, while the operational view involves the actions of individuals. Thus, there are two types of trust within an organization. Intentional trust is the aggregate set of emotions and beliefs held by the people in the organization about an insider; it can be measured by asking questions such as "Would you trust Joe with the corporate bank account?" Operational trust consists of the actions of the people in the organization regarding the insider; it can be measured by observing whether, for example, Joe is actually allowed to access the corporate bank account.

There can be a discrepancy between the intentional and the operational domains. Implicit premises that are difficult or impossible to capture in operational terms frequently underpin the intentional domain. For example, when X says that he trusts Joe to "edit the database," he really means "edit the database for the benefit of the company," which is very difficult to express in an operational system.

More specifically, the current pattern is applicable to IT-intensive organizations with a traditional management structure, a mature IT staff, a culture of process improvement, and resources sufficient for addressing insider threat issues.

## 16.4. Problem

Too often organizations fall into the “trust trap” [1], which says that as intentional trust grows, operational trust should grow likewise. On the surface, this approach has an appealing, common-sense feel to it: surely the better feelings management has about individuals, the less those individuals need to be monitored and the fewer controls need to be applied to them. It seems to improve morale by showing insiders that they are respected and valued; maximize organizational resources by not deploying unnecessary monitoring; and let management feel that they have created a mature, well-run organization that does not rely on sanctions to function efficiently.

However, despite its intuitive appeal, this approach engenders an important weakness: a vicious cycle involving monitoring and intentional trust. The less an organization monitors an employee, the fewer opportunities there will be to detect behavior that would diminish the intentional trust placed in the employee, which leads to even less monitoring. Observing fewer bad behaviors leads the organization to trust the employee more and monitor him or her even less. Combined with a frequently concomitant loosening of the access controls applied to the employee, and with the fact that decreased monitoring can embolden employees contemplating malicious activities, the trust trap exposes the organization to the possibility of a serious insider attack. The current pattern helps organizations balance operational and intentional trust to avoid the trust trap.

Figure 28 is a causal loop diagram<sup>3</sup> [2] that illustrates the trust trap. The loop on the left is a reinforcing loop: the initial intentional trust of the insider causes the monitoring of the insider to decrease, which means that suspicious acts are more likely to go undiscovered, which allows the intentional trust of the insider to go up, all in a self-reinforcing pattern. The loop on the right is a stabilizing loop: the monitoring of the insider leads to the discovery of suspicious acts by the insider, which investigation may reveal to be actual malicious or criminal acts, which decreases the number of insider malicious acts through either the termination or the punishment of the insider.

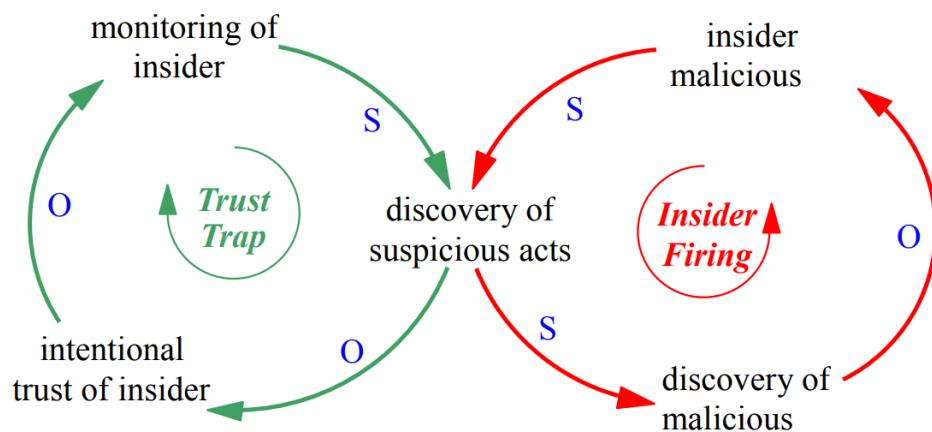


Figure 28: A causal loop diagram of the trust trap

## 16.5. Solution

The trust trap is so widespread in part because adjusting operational trust to match intentional trust does, in fact, work most of the time and can be an effective heuristic. There is even a

biological basis for its effectiveness. Recent research has demonstrated a neurological basis for the interplay of trust and trustworthiness. Individuals who perceive that they are trusted show a rise in their levels of oxytocin, the so-called social binding hormone. This rise in oxytocin is in turn associated with more trustworthy behavior in those individuals [3, 4].

It is rare that an employee you feel is trustworthy will betray you. It is essentially impossible to completely abandon the heuristic and apply the same degree of monitoring to trusted insiders as to outsiders. However, there are a number of activities that can help balance the forces.

Maintain morale. The potential adverse effects of employee monitoring on morale can be mitigated by two measures. The first is effective communications with employees about monitoring and why it is done. It is unlikely that employees in North America will come to be flattered by monitoring as is the case in some Asian countries, but a well-planned awareness campaign can go a long way.

The second measure is to make sure that the monitoring is scrupulously fair and applied impartially. Unfairness has been shown to be an important component of disgruntlement, which in turn has been shown to be a major component of insider threats.

Optimize resources. To minimize the cost of an employee monitoring program, three techniques have proven useful. The first is to use spot checks rather than continuous monitoring. The second is to avoid distributing the details of the monitoring activities so that employees assume that there is more monitoring than there actually is. The third is to increase monitoring at times of greatest risk, such as during the last month of employment (see “A Pattern for Increased Monitoring for Intellectual Property Theft by Departing Insiders”).

Educate management. We all like to be loved, and scrutinizing employees to detect signs of betrayal is something that many managers are loath to do. Here too an awareness program that makes clear that charisma and bonhomie are not enough, and that good managers must be prepared to sacrifice some of their congenial aura to protect the assets of the organization.

Monitor accurately. Given the current state of technology, it is usually impossible to express intentional policies in operational terms. The intentional policy may be something like “Alert me whenever Mr. Smith sends suspicious e-mail to Uruguay,” but implementing that would require complex programming, simplifying assumptions, and questionable approximations. The pattern “Early Detection of Insider Activity” is the framework for improving detection, but two specific best practices are recommended here. The first is to use a formal language such as Description Logic (DL) [5] to express policies. Even if that language is not processed automatically, it will at least give a concise, unambiguous version of the policy. The second is to start from a clean, unambiguous model of insiders within the organization, using something like Gates and Bishop’s lattice model [6] of groups, assets, and controls, as this gives a clean framework that is tailored for insider threats.

Secure results. The monitoring infrastructure, and especially the data collected, are highly sensitive and need to be protected by suitable authentication and authorization mechanisms

## 16.6. Structure

Figure 29 shows the structure of the solution. The trust trap mitigation steps have increased the monitoring of the insider so that the organization can detect his or her suspicious acts, resulting in decreased intentional trust of the insider.

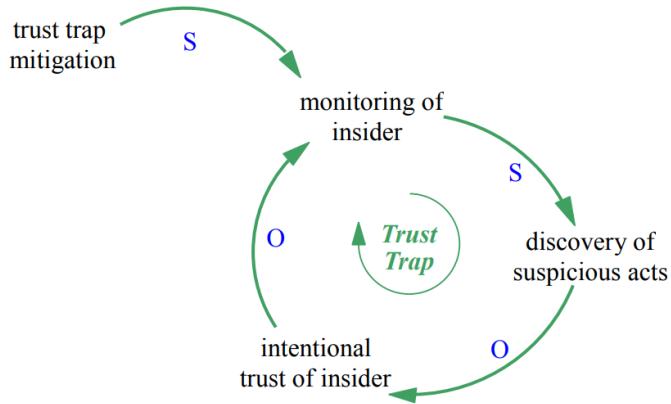


Figure 29: The stabilized trust trap

## 16.7. Dynamics

The dynamics of the solution are conceptually simple, although they can seem cumbersome to management. Following established policies and procedures, the organization's entire staff—the IT staff especially—monitors what goes on with an eye to detecting events that could be malicious activity or its precursors. If staff detect suspicious events, they report them through the organization's documented reporting process. This activates the right-hand loop in Figure 28, lowering the organization's trust in the individual involved. In extreme cases, that diminution in trust can result in the termination of the insider.

## 16.8. Implementation

The following checklist is useful when using this pattern.

1. Understand your legal obligations and responsibilities regarding privacy. If at all possible, obtain employee consent for monitoring (see [7]).
2. Review policies to make sure you give employees a clear general overview of employee monitoring within your organization.
3. Determine your budget for employee monitoring and what level of monitoring fits within that budget.
4. Design a monitoring program that is flexible enough to respond to changes in your trust level and to allow easy spot checks and increased granularity.
5. Map your high-level monitoring design onto low-level rule sets to implement it.
6. Establish a threat model and use it to define what situations warrant increasing the level of monitoring.
7. Have your monitoring plan reviewed by an ethics office, by human resources, or by a legal representative to ensure its fairness.
8. Plan an awareness campaign for managers that explains their role in avoiding the trust trap

## 16.9. Example Resolved

Despite the emotional trust Joe's manager Tracy has in him, she requests daily reports from her physical security staff of any unusual after-hours comings and goings. So the day after Joe

fixes Tom's ticket, Tracy calls Joe into her office and asks him what he had been doing at the office so late the day before. Joe flushes bright red and stammers that he had realized he had made an error in the database and simply could not rest until it was fixed.

Tracy had reviewed the database logs, but the log was not fine-grained enough to show exactly what Joe had done, only that he had accessed the database. Tracy's suspicions are aroused, and she chides Joe about his after-hours action, gently making the point that she has her eye on him. She has IT increase the granularity level on the database log and increase the monitoring level on Joe's activities for a two-month period.

Joe, embarrassed at having been caught and now fearful of Tracy's monitoring, decides it is not worth jeopardizing his career just to appear to be a big shot in front of his friends. He continues to work in the court office another five years with no further incidents.

## 16.10. Consequences

The net result of this pattern is the moderation of the reinforcing trust trap loop via inputs that increase monitoring. The primary mechanism is management awareness of the problem and willingness to perform the monitoring, but other mechanisms, such as fairness and minimal resource expenditure, play an important role in diminishing unintended consequences of such monitoring.

This pattern results in the following benefits:

- Constant verification of emotional and behavioral trust. The balancing of emotional and behavioral trust is not a new problem. "Trust is good, but control is better," said Vladimir Lenin, while Ronald Reagan was pithier: "Trust, but verify." The current pattern avoids the extremes of paranoia and gullibility.
- Continued demonstrations of respect for employees. By including employees in the monitoring design process, and by educating managers in disgruntlement mitigation strategies, this pattern enables the organization to have its cake and eat it too: the organization can obtain the data it needs while maintaining the good will of its employees.
- Optimal use of monitoring resources. Monitoring can be a very expensive proposition: the quantity of data is enormous; the analysis of that data is an AI-complete problem, meaning that it cannot be truly solved without strong artificial intelligence; and high false positive rates are endemic. The strategies in the current pattern alleviate this problem.
- Employee acceptance of the need for monitoring. The goal of this pattern is not just to avoid disgruntlement, but also to enlist employees in the enterprise-wide insider threat mitigation program.
- Protection of organizational assets. By increasing the level of detection, this pattern contributes to decreasing the organization's exposure to theft of intellectual property, fraud, and sabotage.

This pattern suffers from the following drawbacks:

- Complex monitoring program. Designing a monitoring program that balances the forces is not easy, and evaluating its effectiveness is, as with any security measure, problematic.

- Continuing gap between monitoring policy and its implementation. Given the current state of technology, it is not possible to specify exactly what events should trigger alerts, let alone to implement those specifications exactly.

## 16.11. Known Uses

The authors are unaware of any implementation of the pattern in a production environment. However, the individual components of the solution—employee monitoring, spot checking, hiding the details of monitoring, risk-based monitoring, management education, awareness programs, formal policy specifications, and securing the results of monitoring—are widely deployed. Further, most of them are embodied in cyber-security standards such as the Federal Information Security Management Act of 2002 (FISMA), ISO/IEC 27002:2005, and the CERT® Resilience Management Model. For example, NIST Special Publication 800-53 specifies security controls for FISMA and incorporates best practices for position categorization and personnel screening (PS-2 Position Categorization and PS-3 Personnel Screening), maintenance of explicit rules of behavior and consequences for violation (PL-4 Rules of Behavior and PS-8 Personnel Sanctions), audit log monitoring (AU-6 Audit Review, Analysis, and Reporting), and security awareness and training (AT-2 Security Awareness and AT-3 Security Training). The requirement that controls must be based on broad community consensus prior to their inclusion in NIST 800-53 and the requirement that federal agencies must by law implement these controls indicate that the basis for the Trust Trap Mitigation pattern is tried and true.

Random spot checking is required by federal policy. Hiding the details of monitoring is a widely used practice in the intelligence community. Increased monitoring during periods of increased risk is an obvious strategy, recently embodied by Hanley [8]. Cranor [9] exemplifies formal specification of privacy policies, and Breaux [5] exemplifies privacy regulation.

## 16.12. See Also

The parent pattern for the Trust Trap Mitigation pattern is the “Early Detection of Insider Activity” pattern, which stresses the need to monitor insider attack precursors to detect actual attacks as early as possible. A child pattern of the Trust Trap Mitigation pattern is “A Pattern for Increased Monitoring for Intellectual Property Theft by Departing Insiders,” which optimizes employee monitoring by reinforcing it during periods of increased risk, such as just prior to insider termination. A sibling pattern is the “Rapid Escalation” pattern for making sure that alerts are handled in a timely fashion. All of these patterns will appear in Moore [10].

## 16.13. References

- [1] Band, S. R., Cappelli, D. M., Fischer, L. F., Moore, A. P., Shaw, E. D., & Trzeciak, R. F. (2006). *Comparing insider IT sabotage and espionage: A model-based analysis*. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.
- [2] Anderson, V., & Johnson, L. (1997). *Systems thinking basics* (pp. 1-14). Cambridge, MA: Pegasus Communications.
- [3] Zak, P. J., Kurzban, R., & Matzner, W. T. (2004). The neurobiology of trust. *Annals of the New York Academy of Sciences*, 1032(1), 224-227.
- [4] Zak, P. J. (2008). The neurobiology of trust. *Scientific American*, 298(6), 88-95.

- [5] Breaux, T. D., Antón, A. I., & Doyle, J. (2008). Semantic parameterization: A process for modeling domain descriptions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 18(2), 1-27.
- [6] Bishop, M., & Gates, C. (2008, May). Defining the insider threat. In *Proceedings of the 4th annual workshop on Cyber security and information intelligence research: developing strategies to meet the cyber security and information intelligence challenges ahead* (pp. 1-3).
- [7] Huth, C. (n.d.) Monitoring and Privacy: A Legal Perspective. Unpublished.
- [8] Henley, M. (2011). *Deriving candidate technical controls and indicators of insider attack from socio-technical models and data*. CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST.
- [9] Cranor, L. (2002). *Web privacy with P3P*. " O'Reilly Media, Inc.".
- [10] Moore, A., et al. (n.d.) Architecting the Enterprise to Protect Against Insider Threat. Unpublished.

## 16.14. Sources

- Mundie, D. A., & Moore, A. P. (2011, October). A pattern for trust trap mitigation. In *Proceedings of the 18th Conference on Pattern Languages of Programs* (pp. 1-7).

# 17. Secure Boot (Authenticated Boot)

## 17.1. Intent

This pattern addresses how to ensure that violations of integrity properties of the software stack that is booted on a platform can be either prevented (secure boot) or detected (authenticated boot).

Authenticated boot refers to an architecture that ensures that local or remote parties can verify properties of the software that has been booted. While secure boot interrupts the boot process if a modification of the loaded modules is detected, authenticated boot continues the boot process even in that case. However, any modification is detected and recorded securely for later inspection or verification.

## 17.2. Example

Consider a user who wants to use a computing device that was left unattended or that was used by another person before. How can the user be sure that the system software is in the intended operational state, i.e., that no critical component of the operating system or other software applications has been modified in a malicious or unauthorized way? Typically, a file integrity checker program can check the integrity of system and application files. However, any file integrity checker program must rely on trusted reference values and that those values have not been tampered with. Moreover, the user wants also to be sure that the file integrity checker itself is not tampered with or deactivated at all.

## 17.3. Context

Users of security-sensitive applications want to be sure about the operational integrity of their applications and execution environment. Unauthorized changes to the application code or the operating system may lead to unintentional program behavior or violation of security goals. Users trust the hardware, but they need a way to verify that the software loaded on this hardware has not been tampered with.

## 17.4. Problem

On conventional platforms, software can be manipulated or exchanged. Local users cannot verify if they are interacting with the “correct” software, and remote platforms cannot verify the software of their communication partner.

Before applications can be used on a computer system, the system has to be bootstrapped. Typically, the hardware starts by loading a piece of code (firmware, or BIOS on PCs), which in turn loads the bootloader from a pre-defined place from main storage (the disk drives). The bootloader loads the operating system kernel, and the operating system kernel loads system services, device drivers, and other applications.

At any stage of the bootstrap process, software components could have been exchanged or modified by another user or by malicious software that has been executed before.

The following forces have to be resolved:

- You want to ensure the integrity of the loaded software on the system, otherwise malicious software could run without being noticed.
- You want the computer system to always boot in a well-defined secure state. Otherwise, attackers could violate security goals by putting it into an insecure state.
- You want to allow modifications of the operating system or application binaries. Otherwise, software installation and updates would be problematic.

## 17.5. Solution

Based on the assumption that the hardware of the computer system is correct, the integrity of lower layer boot modules is checked, and control is transferred to the next stage if and only if the integrity of that stage is valid. Hence, every stage is responsible for checking the integrity of the next stage. Integrity checking can be performed in different ways: two common methods are comparing hash values or verifying digital signatures. In the first case, hash values of program binaries are computed using a cryptographic hash function (e.g., SHA-1) and compared to reference values. If the computed value does not match the reference value, the binary has been modified. In the second case, each program binary is cryptographically signed by its vendor with a signature key that is only known to the vendor. The signature of the binary can be verified to check that it has not been modified since its signature generation.

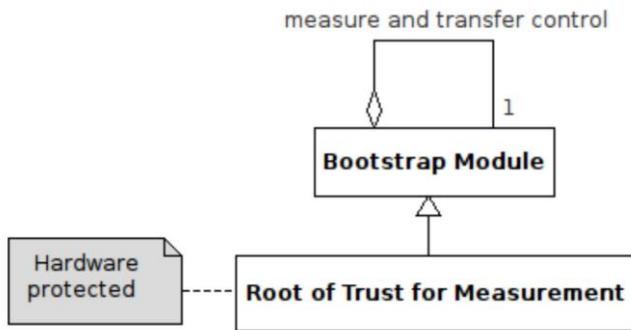
If one stage detects an integrity violation, execution is stopped and the system halts. The sequence of these integrity checks builds the chain of trust. Thus, the user knows implicitly that the system has booted valid program code if it is running at all.

If the first module of such a sequence of integrity checks has already been modified in an unauthorized or malicious way, then the user cannot trust on subsequent integrity checks. The modified module could cheat or even skip any integrity checking. Therefore, the very first boot module is the root of trust for the whole chain of integrity measurements and needs to be protected against unauthorized modifications. The integrity verification data (hash reference values, or signature verification keys) must be protected, too.

To protect the initial boot module (and its verification data) and to reliably build the chain of trust, the root of trust is realized in hardware. Hardware is assumed to be more secure than software because it cannot be changed or read out as easily as software. Moreover, hardware security modules can be protected against various physical attacks – at least to some extent.

## 17.6. Structure

Figure 30 shows the elements of the Secure Boot pattern. The Root of Trust for Measurement is the first module in the bootstrap chain and realized and protected by hardware. A Bootstrap Module has a link to the next Bootstrap Module, which is “measured” for its integrity (typically, by computing a hash value) before control is transferred. Each Bootstrap Module has to maintain (and protect) its corresponding integrity verification data. The verification data of the Root of Trust module is also protected by the hardware, e.g., stored in protected memory registers.



*Figure 30: Elements of the Secure Boot pattern.*

Usually, the last Bootstrap Module is the operating system, which can load several different applications and not only one. In addition, applications can also start other applications or load libraries. In this case, applications also have to measure the integrity of the corresponding components.

## 17.7. Dynamics

When the computer system is started, the Root of Trust for Measurement is executed. It loads and measures the program code of the subsequent Bootstrap Module and verifies the integrity of this code. If this fails, the Root of Trust stops the execution, and the system is halted. Otherwise, the Root of Trust transfers control to the subsequent Bootstrap Module.

The Bootstrap Module loads and measures the code of the next Bootstrap Module and verifies the integrity of this code based on the verification data. If this fails, the system is halted, otherwise control is transferred to the next Bootstrap Module. This continues until the last module in the chain of trust has received control (typically, applications).

## 17.8. Implementation

Authenticated boot can be implemented using a hardware security module in the following way: The first module in the bootstrap chain is trusted by assumption (e.g., implemented in hardware), and measures the integrity of the next module. This integrity measurement is then stored in protected hardware registers. The measurements taken at load-time of the software can later be reported by the security module. For this, the module signs the stored values.

## 17.9. Example Resolved

Based on secure boot, the user can be sure that the correct system has been booted on the computer without changes, because any modifications would have caused the boot process to abort. Moreover, it is not possible to boot a completely different system, because the boot chain starts with a root of trust protected by hardware.

## 17.10. Consequences

The benefits from the secure boot pattern include:

- The software integrity state is verified at boot time, and only software that passed this verification is booted.
- With the variant authenticated boot, it is possible to boot any software, however, the integrity state of the software at boot time can be checked later.

The liabilities from the secure boot pattern include:

- Integrity verification data must be installed and updated (e.g., due to revocation) in a secure fashion (preserving integrity, authenticity, and freshness of the data).
- Software updates could be an issue, because after an update, the integrity state is changed (the software is modified). Hence, specific mechanisms for updates are needed to allow the system to boot properly afterwards.
- The integrity of modules during runtime needs to be ensured by additional mechanisms.
- The integrity verification of large modules (e.g., an entire OS) may be time consuming. Hence, the OS may require adaption to include integrity verification of its modules and applications.
- This pattern adds complexity and overhead. It needs to be coordinated with the other protection mechanisms.

## 17.11. Known Uses

There are various implementations of secure boot:

- AEGIS [1] is a secure boot architecture for PCs, where a chain of trust is constructed at boot time by hashing components, and comparing the result with digitally signed reference values. An expansion card implements the necessary hardware root of trust for measurement.
- The Cell Broadband Engine processor [2] implements a secure boot mechanism to load application code into an isolated execution mode of one of its multiple processor cores. The Cell processor provides a hardware root of trust that verifies the integrity of the loaded code cryptographically. Only if the verification succeeds, the code is executed. Moreover, every time an application wants to use this isolated execution environment, it has to pass the secure boot process. Within the isolated mode, the application code is protected from software running on other cores and even from the operating system running on the supervisor core. The Cell Broadband Engine is used in IBM Cell blade servers and also in the Sony Playstation3 game console.
- For mobile devices, requirements for secure boot have been collected in Open Mobile Terminal Platform (OMTP) recommendations [3] that take into account different industrial standards and specifications, including specifications from the Open Mobile Alliance (OMA), the TCG, and the 3rd Generation Partnership Project (3GPP). The document describes the mechanisms for secure boot and authenticated boot in an abstract way, and hardware manufacturers implement them differently.

There is a well-known example for authenticated boot:

- The TCG specified authenticated boot based on the TPM [4]. The TPM contains special-purpose registers called Platform Configuration Registers (PCRs) that can be used to store hash values. These PCRs cannot be overwritten but only “extended” by computing hash values. Starting from the root of trust for measurement, all software

modules in the boot chain of a TCG-enabled PC (BIOS, boot loader, operating system kernel, etc.) are first hashed, a PCR is extended accordingly, and then control is transferred. By this, the entire bootstrap chain is measured and recorded in PCRs. The TPM also offers a function TPM\_Quote, where the recorded hash values are digitally signed and reported to the caller.

## 17.12. See Also

Boot Loader [5] describes the boot process as a sequence of single bootstrap stages. Each stage loads the image of the next one and transfers control after validation of the image according to certain requirements, typically verifying error correction checksums. In contrast, Secure Boot requires a secure verification of the integrity of the image in each stage, e.g., based on cryptographic hash functions. In addition, Secure Boot defines a root of trust for measurement and requires its protection by trusted hardware to establish a chain of trust throughout all stages.

Authenticator verifies the identity of a subject and creates a proof of identity for later use, e.g., in access control decisions. However, it is typically intended to authenticate users and it does not include a protection for the Authenticator itself. Hence, it cannot provide a chain of trust. But Authenticator can be combined with Secure Boot to extend the proof of identity to the underlying system components.

In [6], the authors introduce patterns for TPM usage. Their patterns concern the communication of software with the TPM, whereas the Secure Boot pattern describes more abstract concepts that can be realized based on TPMs, or based on alternatives, such as secure co-processors.

## 17.13. References

- [1] Arbaugh, W. A., Farber, D. J., & Smith, J. M. (1997, May). A secure and reliable bootstrap architecture. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)* (pp. 65-71). IEEE.
- [2] Shimizu, K. (2006). The cell broadband engine processor security architecture. *IBM DeveloperWorks*. <http://www.ibm.com/developerworks/power/library/pa-cellsecurity/>
- [3] Open Mobile Terminal Platform Consortium. (2009). OMTP advanced trusted environment: OMTP TR1 (v1.1). OMTP. <http://www.omtp.org/Publications.aspx>
- [4] Trusted Computing Group. (2007). TCG TPM specification version 1.2, revision 103. <https://www.trustedcomputinggroup.org/specs/TPM>
- [5] Schutz, D., (2006). *Boot loader*. Proceedings of the 11th European Conference on Pattern of Languages of Programs.
- [6] Gurgens, S., Rudolph, C., Mana, A., & Munoz, A. (2007, September). Facilitating the use of TPM technologies through S&D patterns. In *18th International Workshop on Database and Expert Systems Applications (DEXA 2007)* (pp. 765-769). IEEE.

## **17.14. Sources**

Löhr, H., Sadeghi, A. R., & Winandy, M. (2010, February). Patterns for secure boot and secure storage in computer systems. In 2010 International Conference on Availability, Reliability and Security (pp. 569-573). IEEE.

# 18. Credential Renewal

## 18.1. Intent

A valid delegation certificate must exist throughout the duration of the delegation, but the proxy certificates may expire before the delegated job finishes. When the proxy credentials are close to expiry time for jobs still running, the grantors try to renew the proxy certificate for grantees.

## 18.2. Example

A user at site A requests to run a simulation on a powerful server at site B. The user delegates his rights to the simulation to access input data or invoke services on another server at site C by issuing a short-lived proxy credential to the simulation. The proxy credential may expire while the delegated process is still running on site C. Site C may then stop the simulation's access to its data or service as it does not trust the simulation anymore. The simulation may, therefore, be prematurely halted or interrupted, which is clearly undesirable. Figure 31 summarizes this scenario.

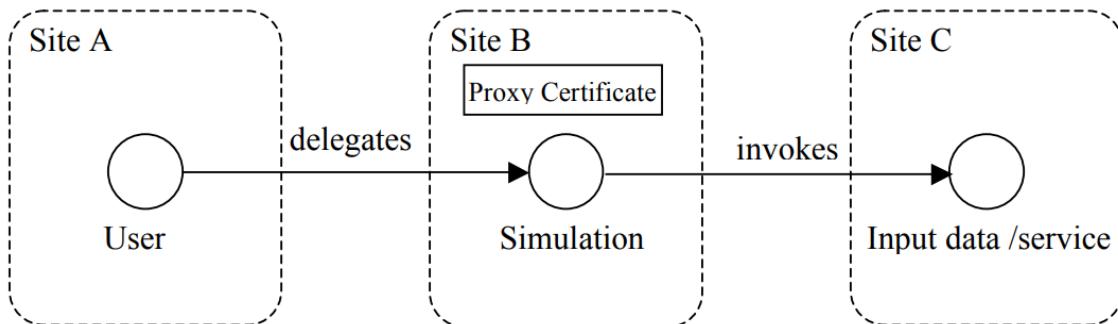


Figure 31: Credential Renewal example

## 18.3. Context

When using Short Lifespan, we need to ensure that long-running jobs are not prematurely aborted due to proxy credential expiry.

## 18.4. Problem

A valid delegation certificate must exist throughout the duration of the delegation, but the proxy certificates may expire before the delegated job finishes. As described in Short Lifespan, the lifetime of the proxy certificate is limited to a short period. However, for a long-lived job with delegation, the proxy certificate may time out before the job finishes. Even if the proxy certificate lifetime was sufficient for the job's computation, the overall job lifetime may still be longer than estimated. For example, if the resource where the job is being processed stops working, the job must be rescheduled and resubmitted. The computation is possibly restarted from the beginning. The jobs can also spend some time in queues waiting for required resources. It is very hard to predict the overall job requiring time.

To solve the delegation certificate expiry issue, a simple solution is to set proxy certificate lifetime very long for possibly long-running services. Given the difficulty of predicting job run-times in practice, this may lead to an over-estimation of the required credential lifetime and increase risk that a credential will be compromised and misused.

To solve this problem, the following forces must be balanced:

- The solution should not interrupt the running transactions already invoked by the grantees.
- A valid proxy certificate must exist throughout the duration of the delegation.

## 18.5. Solution

When the proxy credentials are close to expiring time for jobs still running, the grantors renew the proxy certificate for grantees. A grantor of delegation usually keeps a list of the delegation credentials it issued together with the expiry time. For a possibly long-lived job with delegation, the grantor of delegation is responsible to start the renewal of the corresponding proxy certificate.

The credential renewal will be started when the proxy is nearing expiration (e.g. 1/4 of lifetime remaining). The Renewal Grantor (the grantor in renewal) computes the time when the proxy should be renewed and stores this information in a database. If the renewal succeeds, the information is removed from the database. If it fails, the Renewal Grantor computes a new time to attempt the renewal again, using the method of bisecting intervals of remaining lifetime, with a minimum time interval chosen depends on the context (e.g. five minutes). If all the renewal attempts fail and the proxy expires, it is removed from the database, and the delegation job would stop.

When a proxy is renewed, the Renewal Grantee is updated with the Renewed Proxy Certificate. The Renewal Grantee then is responsible to transport the Renewed Proxy Certificate to the Service (all running services) it invoked on behalf of the Renewal Grantor. If the Renewal Grantor expects the job to be finished before the new proxy certificate expires, it deletes the corresponding entry from the database. Otherwise, it computes the next attempt, updates the database with this information, and retries.

## 18.6. Structure

Figure 32 below illustrates the structure of the Credential Renewal pattern. The delegation grantor attempts renewing the proxy certificate before it expires for a long running job. At the time of attempting, the grantor - called Renewal Grantor in this pattern - requests a renewal to the grantees – called Renewal Grantee. The Renewal Grantee updates its proxy certificate to a Renewed Proxy Certificate and asks the Renewal Grantor to sign. Because public key-pair generation is computationally intensive, the current proxy credentials are reused in the renewal. After the renewal, the Renewal Grantee updates the Service it invoked on the Renewal Grantor's behalf with the Renewed Proxy Certificate. The Service would then verify and use the Renewed Proxy Certificate until it expires.

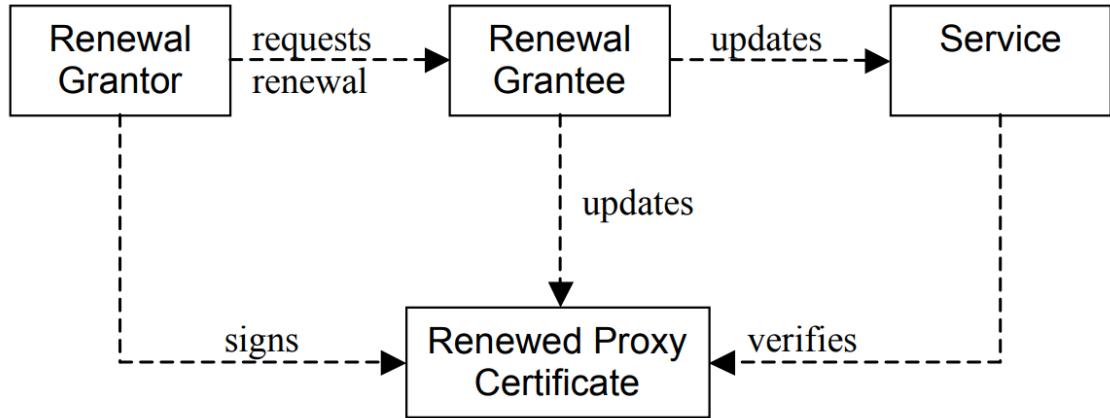


Figure 32: Structure of the Credential Renewal pattern

## 18.7. Dynamics

## 18.8. Implementation

## 18.9. Example Resolved

Before the current proxy certificate expires, the user (Renewal Grantor) at site A requests a renewal to the simulation (Renewal Grantee) at site B. If the renewal succeeds, a new proxy certificate (Renewed Proxy Certificate) is issued to the simulation. The simulation then updates the service at site C with the new certificate.

Figure 33 below illustrates how the example is resolved, using stereotypes of the roles of each component (Renewal Grantor, Renewal Grantee, Renewed Proxy Certificate and Service) indicated in correspondence with the pattern.

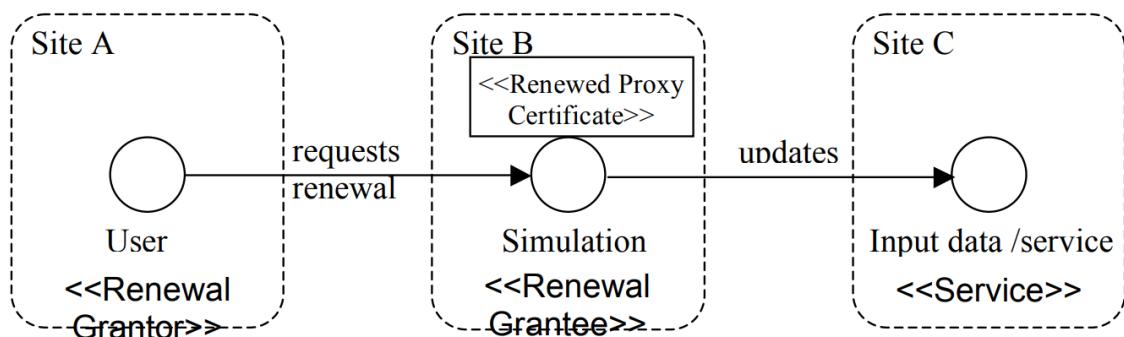


Figure 33: Credential Renewal pattern example resolved (stereotypes indicate roles in the pattern)

## 18.10. Consequences

The following benefits may be expected from applying this pattern:

- Before the old proxy certificate expires, it is renewed and updated, so the transaction already running is not interrupted by the renewal.
- If the renewal does not fail for abnormal reasons (such as failure of the network connection), a valid proxy certificate is always existing throughout the duration of the delegation.

The following liabilities may arise from applying this pattern:

- Performance is negatively affected due to the re-issuing and transmitting the proxy credentials. The longer the transactions are, the bigger the impact is.
- More duties are put on grantors and grantees due to the renewal.
- The infrastructure becomes more complex in order to support renewal.
- The solution depends on the availability of the grantors. In the practice, the user may delegate his rights to a renewal service, but this renewal service must be always available.
- Current practice re-uses the proxy key-pairs, due to the performance degrades with frequent public key generation. It increases the security concerns that the credential may be compromised and misused, especially for long-running jobs.

## **18.11. Known Uses**

The solution to proxy certificate expiry problem was developed first in EU Data Grid project and further evolved in EGEE project [1]. EU Data Grid project was built on Globus Toolkit 2.2. Users delegate short-lived proxy credentials to the grantees acting on their behalf. When the proxy credentials near expiration, a service called Renewal Service retrieves new short-lived proxy credentials from MyProxy on the user's behalf and refreshes the old credentials. MyProxy, a component of Globus Toolkit, is an online certificate repository containing user end entity credentials for issuing proxy certificates on their behalf. In FusionGrid [2], a web service called Credential Manager (CM) is used to renewal proxy certificates by retrieving the user end entity credentials from a MyProxy server CredentialStore Repository, and adding the new proxy certificates to a MyProxy server ProxyStore containing renewable proxies [3].

## **18.12. See Also**

The precondition of this pattern is a delegation relationship being established with Credential Delegation. Short Lifespan is also a context to limit the proxy certificate lifetime so credential renewal is necessary.

## **18.13. References**

- [1] Koufil, D., & Basney, J. (2005, November). A credential renewal service for long-running jobs. In *The 6th IEEE/ACM International Workshop on Grid Computing, 2005*. (pp. 6-pp). IEEE.
- [2] FusionGrid. (n.d.). FusionGrid. <http://www.fusiongrid.org/>
- [3] Thompson, M., & Goode, M. (2006). Use of MyProxy for the FusionGrid. Globus. <http://www.globus.org/toolkit/presentations/MyProxy-GW06FG.ppt>

## **18.14. Sources**

Lu, M., & Weiss, M. (2008). Patterns for grid security. In Mini-PLoP at conference on object-oriented programming systems, languages, and applications, OOPSLA.

# 19. Multilevel Secure Partitions

## 19.1. Intent

Confines execution of a process in a system partition which has been assigned to a specific confidentiality or integrity level. Access from this process to other partitions (processes or data) is restricted according to the rules of a multilevel security model, where processes have sensitivity levels.

## 19.2. Example

ChronOS is now building a web system. The system's Web, Application, and Database Servers are separated because it is clear that the Web Server has a higher exposure to attacks. If the Web server is compromised, ChronOS will be unable to provide some business services to its clients but at least the hacker is denied immediate access to Application and Database servers where the corporate data is stored. ChronOS wants to limit the damage from any one attack and prevent the attacker erasing their steps from the logs. The normal operation of the application requires processes to request and obtain services between the elements of the system.

## 19.3. Context

Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm and usually share data and other resources. The environment can be centralized or distributed. Some processes may be malicious or contain errors.

In multilevel models data and procedures are classified into sensitivity levels and users have access to them according to their clearances. These models have been formalized in three different ways [1]:

- The Bell-La Padula model, intended to control leakage of information between levels.
- The Biba model, which controls data integrity.
- The Lattice model, which generalizes the partially ordered levels of the previous models using the concept of mathematical lattices.

### **The Bell-La Padula confidentiality model**

This model classifies subjects and data into sensitivity levels. Orthogonal to these levels, compartments or categories are defined, which correspond to divisions or groupings within each level. The classification, C, of a data object defines its sensitivity. Similarly, users or subjects in general are given clearance levels. In each level an access matrix may further refine access control. A security level is defined as a pair (classification level, set of categories).

A security level dominates another (denoted as  $\Rightarrow$ ) if and only if its level is greater or equal than the other level and its categories include the other categories. Two properties, known as "no read up" and "no write down" properties, define secure flow of information:

**Simple security (ss) property.** A subject  $s$  may read object  $o$  only if its classification dominates the object's classification, i.e.,  $C(s) \Rightarrow C(o)$ . This is the no read-up property.

**\*-Property.** A subject  $s$  that can read object  $o$  is allowed to write object  $p$  only if the classification of  $p$  dominates the classification of  $o$ , i.e.,  $C(p) \Rightarrow C(o)$ . This is the no write-down property.

This model also includes trusted subjects that are allowed to violate the security model. These are necessary to perform administrative functions (e.g., declassify documents, increase a user's clearance).

### The Biba integrity model

Biba's model classifies the data into integrity levels ( $I$ ) and defines two properties dual to the simple security and \* properties:

**Single integrity property.** Subject  $s$  can modify object  $o$  only if  $I(s) \Rightarrow I(o)$ .

**Integrity \*-property.** If subject  $s$  has read access to object  $o$  with integrity level  $I(o)$ ,  $s$  can write object  $p$  only if  $I(o) \Rightarrow I(p)$ .

The first property establishes that an untrusted subject cannot write to objects of a higher level of integrity, or she would degrade that object.

## 19.4. Problem

Many security attacks propagate through weak parts of a system. After finding an entry point, the malicious software may access a directory or some other unit of the architecture, frequently escalating its power because it will inherit rights from the compromised units. Protection rings can help with this problem, but they can only be applied at the operating system level because of their need for hardware support.

The following forces affect the solution:

- Even if one part of the system is compromised or corrupted, other units will remain unaffected; that is, attacks or errors in a partition at some level should not propagate to other levels.
- The solution should be applicable to all architectural levels of the system, not just the operating system level.
- The solution should be applicable to distributed environments, not just single-processor systems.

## 19.5. Solution

Divide the architecture in such a way that transfers of control or data access from one division to another follows the Bell LaPadula and/or Biba restrictions. Assign functionality to levels according to their sensitivity.

## 19.6. Structure

Figure 34 shows a class diagram for the solution. A Client Process is assigned by the System to a specific Partition according to their function and degree of trust. A process can call other partitions for services according to a set of Transition Rules, based on a multilevel model.

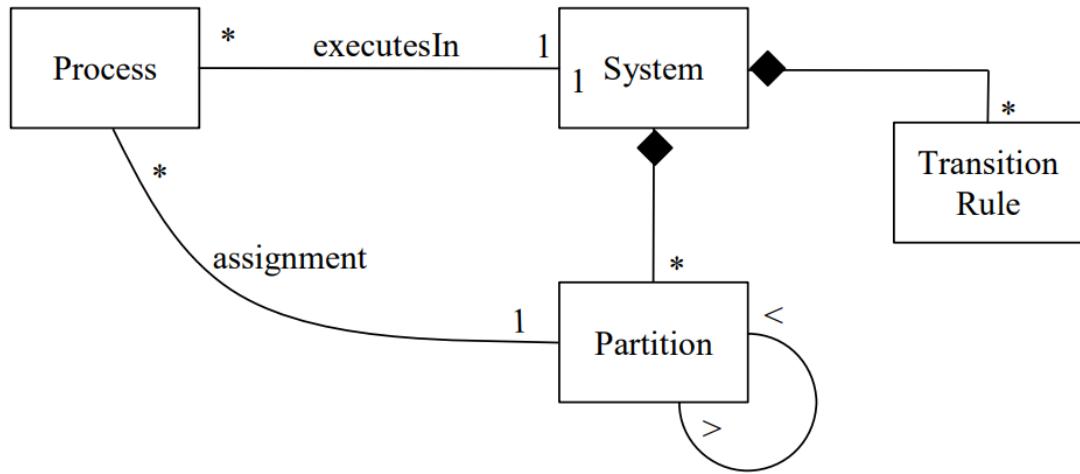


Figure 34: Class diagram for the Multilevel Secure Partitions pattern

## 19.7. Dynamics

Figure 35 shows a sequence diagram for a process requesting a service from a different partition (p2), than the one where it is executing (p1). Requests for services in other partitions follow the rules of the multilevel model.

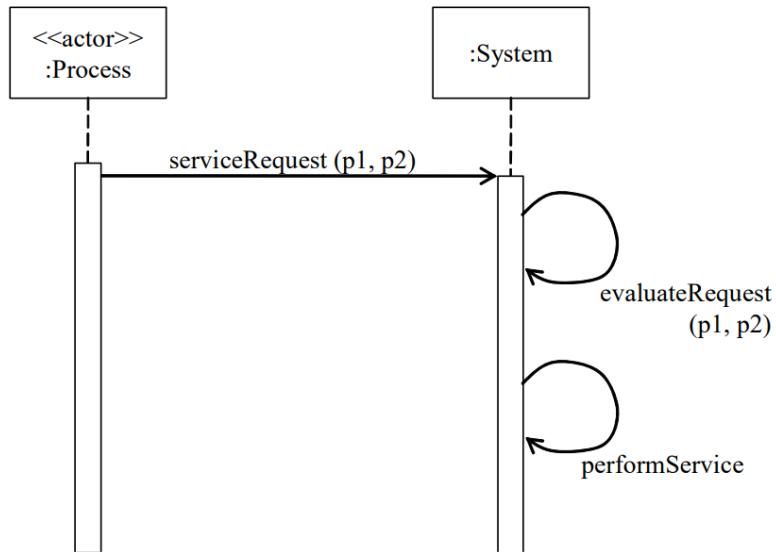


Figure 35: Sequence diagram for requesting a service within a partition

## 19.8. Implementation

The system must support a multilevel access control policy with mandatory restrictions, including data labeling and rule enforcement. Transitions from one partition to another should happen only through Protected Entry Points.

## 19.9. Example Resolved

Figure 36 shows the solution provided by HP to the ChronOS problem [2]. Now, if the web server is compromised, the attacker cannot deface web pages, cannot attack the web application server, cannot erase the log (they are all in different partitions).

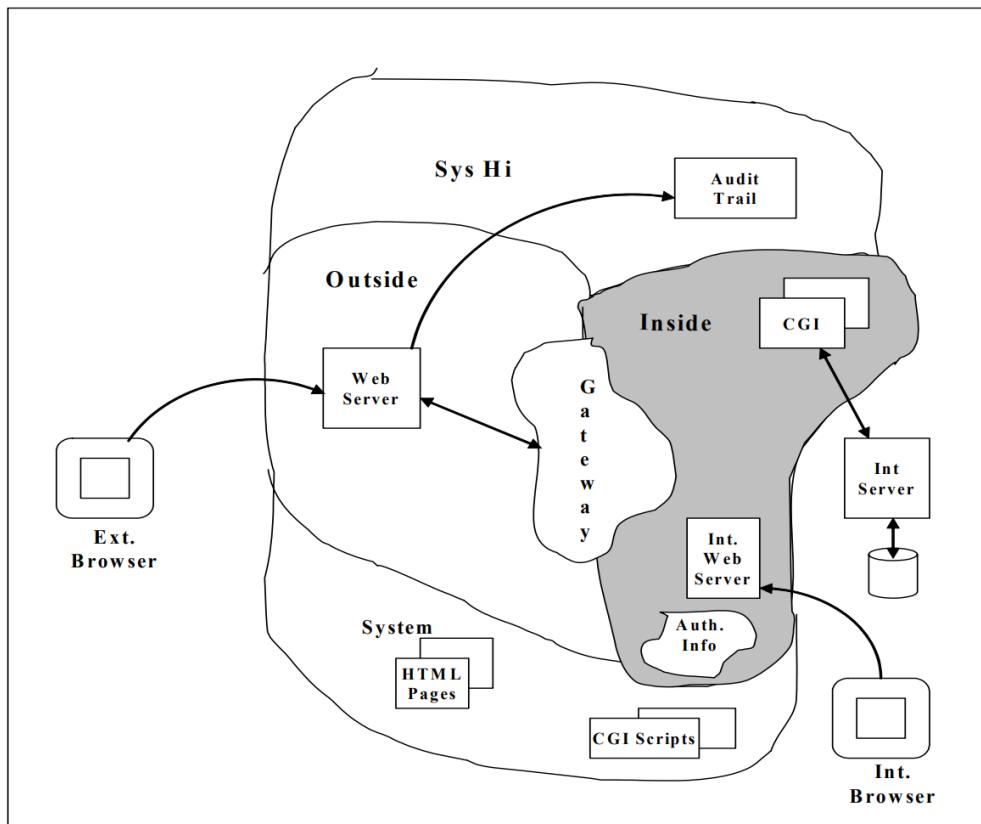


Figure 36: HP's Virtual Vault architecture

## 19.10. Consequences

This pattern has the following advantages:

- A compromised (taken over) or corrupted partition cannot propagate its attack or errors to other partitions.
- The solution does not depend on hardware support and can be applied in any architectural level, from applications to the operating system, to distribution units as in a Service-Oriented Architecture (SOA).
- The solution is particularly suitable for distributed systems because it depends only on local attributes of the procedures and data involved.

- A multilevel model defines the relationship between subject and objects using a lattice model where it can be formally proven that the permitted accesses do not violate security restrictions. This only proves security in an abstract sense and could be affected by implementation details, but it is a good security guideline for the complete system.
- Can lead to a more systematic definition of privileges because the levels can help in structuring them. This also makes administration simpler.

Possible disadvantages include:

- It may not be easy or even possible to place the required functions and data in the appropriate levels. This is particularly true at the application level; most commercial environments are not hierarchical.
- It is not simple to change the levels of existing programs or data. A trusted program is needed to override the rules and move subjects or data to other partitions [1].

## 19.11. Known Uses

- HP's Virtual Vault [2,3,4]—This is a secure platform for Internet applications that includes a trusted HP-UX operating system (a Unix variant), an enterprise server, firewalls, and other protection devices. It was part of a secure server system, the HP Praesidium family of products and appears to be the first use of this pattern. It also reduces the root privileges and controls inheritance of rights in forked threads (See Controlled Inheritance of Rights in [5]).
- HP's restructuring of Unix' Sendmail program and other system programs [6].
- HP's UX-11i, the current version of HP's operating system, also uses this approach [7].

## 19.12. See Also

- Multilevel Security pattern.
- See Protected Entry Points pattern.
- See Protection Rings pattern.
- Protected Address Space (Sandbox) [5]. The MSP pattern can only control the actions from a process with respect to other partitions, the Protected Address Space pattern can define precisely which resources within a partition can be accessed by the process.

## 19.13. References

- [1] Gollmann, D. (2006). Computer security. Wiley. Second Edition.
- [2] Rubin, C., & Arnovitz, D. (1994-1999). *Virtual Vault*. HP Praesidium. Hewlett Packard
- [3] VirtualVault. (1998). Philosophy of Protection. <http://docs.hp.com/en/B5413-90027/B5413-90027.pdf>
- [4] (n.d.). What makes virtual vault secure?. <http://www.docs.hp.com/en/J4255-90011/apas01.html#d0e11429>
- [5] Fernandez, E. B. (2002). Patterns for operating systems access control. *Procs. of PLoP 2002*.

- [6] Zhong, Q., & Edwards, N. (1998). Security control for COTS components. *Computer*, 31(6), 67-73.
- [7] Wikipedia contributors. (n.d.-b). HP-UX Operating system. Wikipedia.  
<https://en.wikipedia.org/wiki/HP-UX>

## 19.14. Sources

Fernandez, E. B., & laRed Martinez, D. (2008, October). Patterns for the secure and reliable execution of processes. In *Proceedings of the 15th Conference on Pattern Languages of Programs* (pp. 1-16).

# **20. Alarm Monitoring**

## **20.1. Intent**

Defines a way to raise events in the system that might require special attention, such as someone tampering with a door.

## **20.2. Example**

Building management wants to raise alarm events when someone opens a door without using the right credentials or when someone tries to use a card that was reported as lost.

## **20.3. Context**

Physical environment with an access control system where we want to be able to raise filtered alarm events and we want to differentiate between alarms that are generated because of a physical violation of the system and alarms generated because of a system rule violation.

## **20.4. Problem**

In an Access control system, there are two types of alarms, physical and logical. Many times, we might want to inform more than one subsystem that a change of state happened in order to generate different types of configurable actions that can vary. Physical alarm inputs are used to monitor various devices. These alarms can be shunted. Logical alarms are used to monitor various system rules. For example, a system may generate an alarm after three invalid tries to get access with the same credentials.

The following forces will affect any solution:

- There are two types of alarms, physical and logical.
- Alarms can be ignored.
- Logical alarms have two possible states, reset and alarm.
- Physical alarms have four possible states: reset, alarm, cut or short. The last two states are known as trouble states, caused by faulty wiring or tampering.
- We may need to know what alarms have been set and when they were reset.
- We need a way to inform interested parties about a change of a state of an alarm.

## **20.5. Solution**

Have an alarm entity that describes the general concepts of an alarm and use generalization to separate logical alarms from physical alarms and their particular characteristics. Add information about the time the alarm occurred. Use the Observer Pattern [1] to advise any interested party of a change of state in an Alarm.

## 20.6. Structure

Figure 37 shows a UML class diagram for the Alarm Monitoring pattern. Abstract class **Alarm** indicates a general class for any type of alarm. The **PhysicalAlarm** class and the **LogicalAlarm** class inherit the **Alarm** class. These classes allow for alarms to be controlled and monitored. By implementing the **AlarmObserver** interface any class (not shown here) interested on this alarm can be added to the list of observers for the alarm and will be advised when a change of state happens. Moreover, we log every alarm activity with a time stamp.

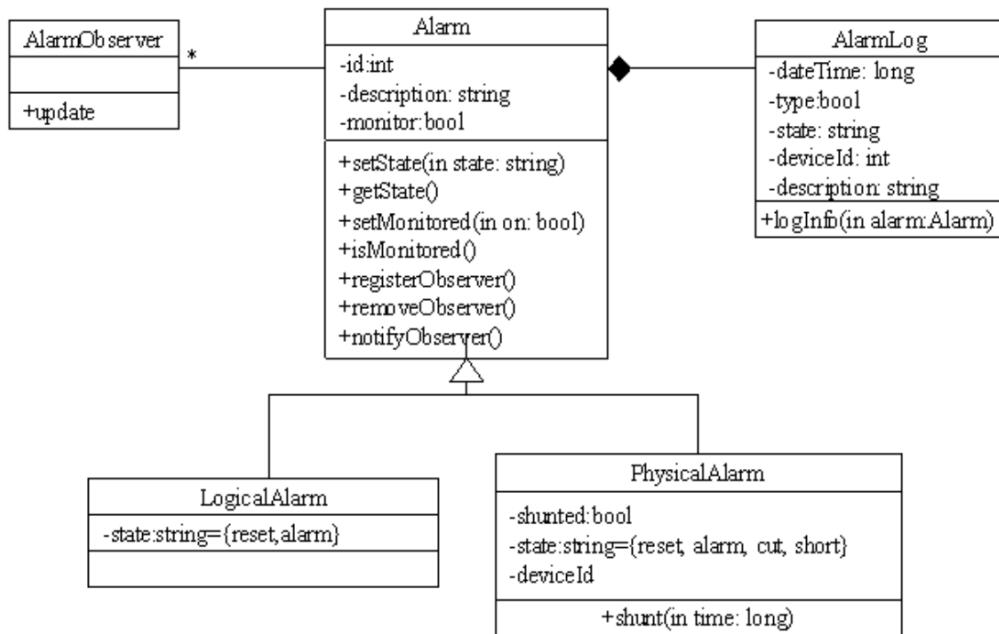


Figure 37: Class Diagram for Alarm Monitoring

## 20.7. Dynamics

Figure 38 shows the main success scenario for the use case “activate an alarm”. The request to set the alarm active is sent by an external actor that detected the need to raise the alarm. The change of state is logged and only if the alarm is being monitored interested parties are advised of the change, otherwise the alarm is ignored.

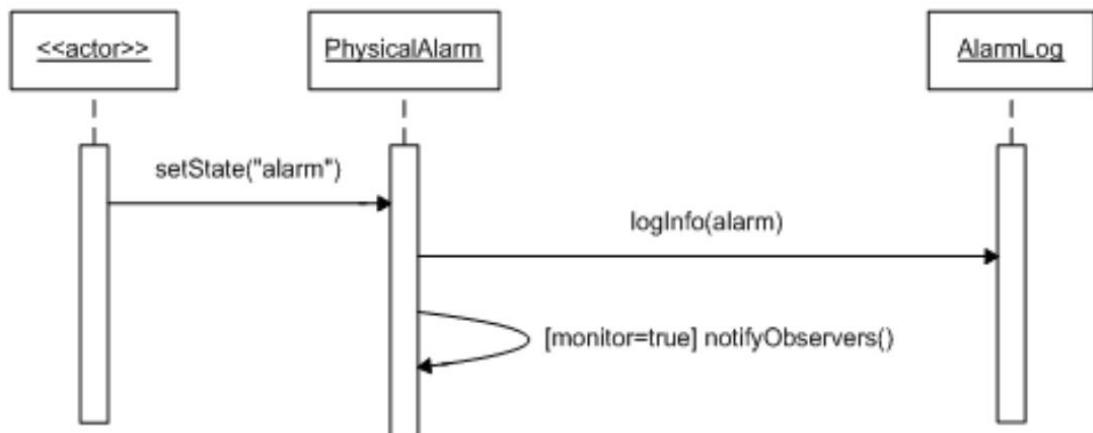


Figure 38: Sequence Diagram for Activating an Alarm

## **20.8. Implementation**

There are many ways to create alarms in a physical access control system. For example, when a card reader is installed on a door, an alarm contact is usually installed as well. The alarm point is used to monitor whether the door was forced or held for too long after a valid access was granted. Logical Alarms can be generated for maximum tries with invalid credential, invalid credential, and communication errors. The call to change the state of an alarm could in turn generate other actions when observers are notified, like displaying a message, activating a siren, etc.

## **20.9. Example Resolved**

Every time someone opens a door without proper permission an alarm can be created and if a person uses a lost badge the system can generate a logical alarm.

## **20.10. Consequences**

Advantages include:

- We can only pay attention to the alarms we are interested in.
- Every alarm change of state is logged, and interested parties are advised.
- This model provides the basic structure for supporting alarms in an access control system.
- We make a distinction between logical and physical alarms, which supports the creation of any alarm independently of the existing hardware.

Disadvantages include:

- The pattern may create overhead for systems that only care about logging the alarm.

## **20.11. Known Uses**

Many commercial Access Control systems have the concept of logical and physical alarms that generate some actions when the states are changed.

## **20.12. See Also**

This pattern is based on the Observer Pattern [1]. The Access Control for Physical Structures [2] complements this pattern by adding the concept of Zones that control Alarms.

## **20.13. References**

[1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

[2] Desouza-Doucet, A. C. (2006). *Controlling access to physical locations*. Florida Atlantic University.

## **20.14. Sources**

Fernandez, E. B., Ballesteros, J., Desouza-Doucet, A. C., & Larrondo-Petrie, M. M. (2007, July). Security patterns for physical access control systems. In *IFIP Annual Conference on Data and Applications Security and Privacy* (pp. 259-274). Springer, Berlin, Heidelberg.

# 21. Relays

## 21.1. Intent

This pattern defines the interactions with electronically controlled switches.

## 21.2. Example

Building management wants to be able to open the main gate for visitors that do not have credentials. Moreover, doors should be opened when someone presents valid credentials.

## 21.3. Context

Physical environment with an access control system where we want to be able turn on/off devices; and lock/unlock doors.

## 21.4. Problem

A relay is an electronically controlled switch. Similar to a light switch on the wall being used to turn on or off a light, a relay can be used to turn on or off other devices. Relays are used to activate the electric door lock, or to activate a variety of other items such as: bells, sirens, turn lights on and off, trip a digital dialer and many other uses. Typically, one relay is used to control the electric strike of doors. The others, usually called auxiliary relays, can be used as needed. When a relay is activated or deactivated, the device wired to it is turned on or off.

The following forces will affect the solution:

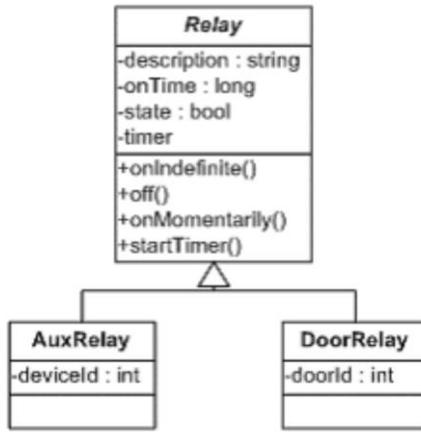
- We need to distinguish between door relays and auxiliary relays.
- Relays can be activated and deactivated.
- Relays can be activated indefinitely or for a defined period of time.
- Relays have two possible states: on and off.

## 21.5. Solution

Have a relay entity that describes the general concepts of a relay and use generalization to separate door relays from auxiliary relays and their particular characteristics.

## 21.6. Structure

Figure 39 shows a UML class diagram for the Relays pattern. Abstract class Relay indicates a general class for any type of relay. The DoorRelay class and the AuxRelay class inherit the Relay class. These classes allow for relays to be controlled.



*Figure 39: Class Diagram for Relays*

## 21.7. Dynamics

The sequence diagram is trivial. The request to set the relay active is sent by an external actor. The relay is activated for the previously defined “on time.” If the relay was already active the timer is restarted. Once the timer expires the relay is turned off.

## 21.8. Implementation

Relays could be activated or deactivated by a variety of events. An alarm input, a valid credential, an egress button being pushed, or a time event, can all activate a relay.

## 21.9. Example Resolved

Doors that have relays defined can be opened when a valid credential is presented. Also the main gate can be assigned an auxiliary relay that can be activated at will.

## 21.10. Consequences

Advantages include:

- We can distinguish between auxiliary and door relays.
- We can activate relays for a predefined amount of time.
- This model provides the basic structure for supporting relays in an access control system.

A disadvantage is:

- We might want the same kind of functionality for devices that are not exactly door or aux relays.

## 21.11. Known Uses

Many commercial Access Control systems have the concept of relay management and distinction between door and aux relays.

## **21.12. See Also**

The Access Control to Physical Structures pattern complements this pattern by adding the concept of Zones that control Relays.

## **21.13. References**

- Fernandez, E. B., Ballesteros, J., Desouza-Doucet, A. C., & Larrondo-Petrie, M. M. (2007, July). Security patterns for physical access control systems. In *IFIP Annual Conference on Data and Applications Security and Privacy* (pp. 259-274). Springer, Berlin, Heidelberg.

## **21.14. Sources**

# 22. Access Control to Physical Structures

## 22.1. Intent

Applies authentication and authorization to the control of access to physical units including alarm monitoring, relays, and time schedules that can control when things will happen.

## 22.2. Example

Building management wants to put in place an access control system to control access to certain zones and to control who can access the zones. They need to deny all access to certain zones after 5pm. They want to generate alarms when someone tries to access a zone for which they do not have permission and start monitoring alarms for all the exterior doors at 8pm. Moreover, they want to turn on the main door light at 7pm.

## 22.3. Context

Physical environment with access control system where we need to control access and turn on/off devices based on time constraints.

## 22.4. Problem

We need a way to enforce business rules in an access control system that take effect at a given time and day of the week. For example, a user may have different access needs on different days, as in the following example:

- 08:30 - 17:00 - Monday, Wednesday, Friday - Standard Hours
- 08:30 - 19:00 - Tuesday, Thursday - Stays Late 2 Days a Week
- 08:00 - 12:00 - half a day on Saturday.

The following forces affect the solution:

- We need a way to automatically cancel access for everyone to some areas of a building at given time and day of the week. Some users might have access to some areas only during a time range of the day.
- We need to provide a way to automatically activate devices based on the time and day of the week.
- We need a way to represent a day of the week and time.
- This pattern expands the Access Control for Physical Structures Pattern in [1]. Therefore, all the forces presented in that pattern are present in this pattern as well.
- We need to restrict access to some users depending on the identity or other characteristics of the visitor.
- The boundaries of the unit of access must be clearly defined.
- There is a variety of users such as employees, contractors, repairpersons, etc., that require access to different parts of a building.
- Since some units will be normally closed, it is necessary to create policies and plans in case of an emergency, such as earthquake or fire.

- We may need to keep track of who entered each unit and when

## 22.5. Solution

Define the structure of an access control system using the Role Based Access Control pattern. Integrate the Alarms Monitoring and Relays patterns and introduce the concept of a time schedule to control when things can/must happen. Time Schedules have two uses; one is to control access times and the other is to configure automatic actions.

## 22.6. Structure

Figure 40 shows a UML class diagram for the Access Control to Physical Structures pattern. We can see how the Alarms Monitoring pattern can be integrated so that Zones can control Alarms. We also use the Relays pattern so that each door has its own Door Relay, and the Aux Relays can be used to turn on/off devices. Zones can control Relays.

We introduce a TimeSchedule class that consists of a few time intervals. A Time Interval consists of a start time, a stop time, and selected day of the week. When the system clock activates a Time Schedule, it can automatically perform some actions. Relays can be turned on and off, alarm zones can be activated or deactivated, and doors can be unlocked. To control access times, roles are combined with Time Schedule to determine both where and when a user can gain access to zones, and we also need to be able to assign a Time Schedule for the entire zone that applies to all users.

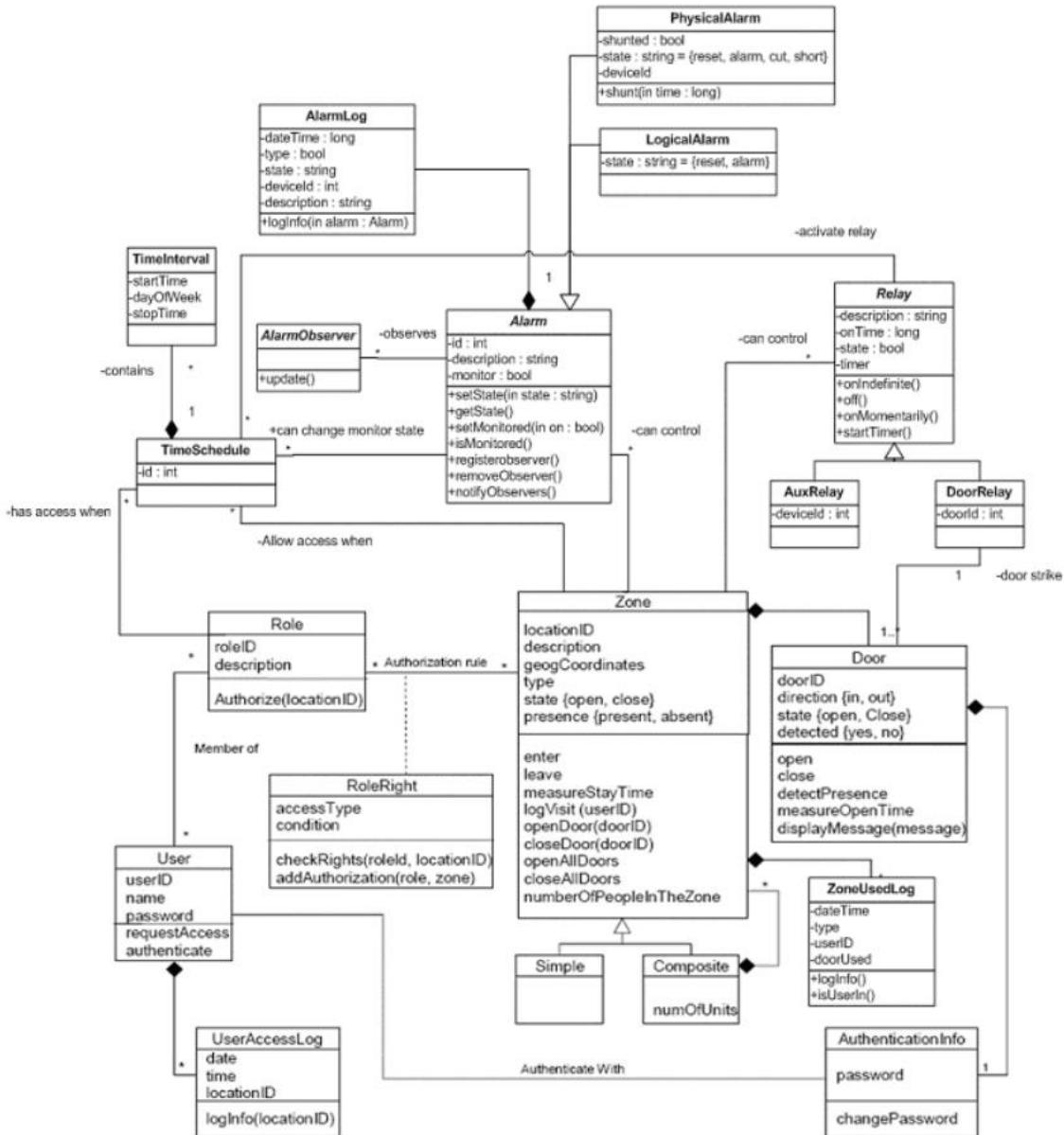


Figure 40: Class Diagram for Access Control to Physical Structures

## 22.7. Dynamics

Figure 41 shows a main success scenario for the use case “enter a zone”. When the control is passed to the zone, the Zone and the Role will consider their Time Schedules before authorizing a person. Also, when the Zone opens or closes a door, the Door calls its Relay to perform the action. Other use cases include: “zone access denied by zone time schedule”, “role access denied by time schedule”, and “activate alarm by role access denied”; they are not shown for conciseness.

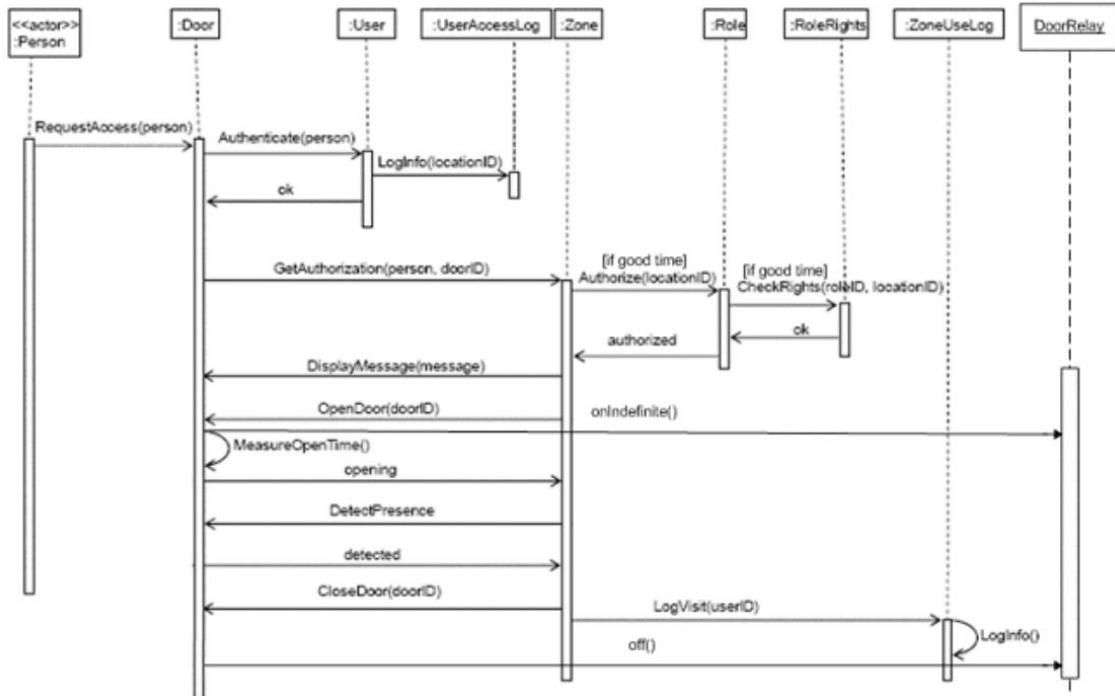


Figure 41: Sequence Diagram for Entering a Zone

## 22.8. Implementation

Access control systems use centralized processing, distributed processing, or hybrid arrangement. The system architecture should be taken into consideration when designing an access control system, since it can have a significant effect upon operation during a catastrophic system failure.

**Centralized Processing.** In computer dependent processing systems, all events are gathered by the field panels, and are then sent to the computer for processing. For example, if a credential is presented at a door, the door sends the credentials to the central computer or processor. The computer checks the credential against its programming and determines if it is allowed through that door at that time. If valid the computer sends a command back to the panel to release the door. In these systems if the computer goes down, or if communications between the panel and computer is lost, the system can no longer function, to verify proper access, and to process alarms.

**Distributed Processing.** In distributed processing systems the database is loaded to the field panel. All decisions are made at the field panel and are passed to the computer or logging printer for storage. In these systems if communications are lost, access control continues uninhibited. Furthermore, the events can sometimes be stored in the panels, and can be sent up to the computer once communications are restored. Due to their architecture, systems which employ distributed processing generally offer better reliability, and faster response than systems that rely on central computers for all decision making.

## 22.9. Example Resolved

Building management can configure time schedules and assign them to Zones and Roles, that a way a Zone could have a time schedule from 8-5pm. Moreover, time schedules can be

assigned to Relays and Alarms so that they can be activated and monitored respectively when the system clock activates these time schedules.

## 22.10. Consequences

Advantages include:

- We can activate relays for a predefined amount of time.
- Every alarm change of state is logged, and interested parties are advised.

A disadvantage is:

- We might want the same kind of functionality for devices that are not exactly door or aux relays.
- The pattern may create overhead for systems.

## 22.11. Known Uses

Many commercial or institutional buildings control access to restricted units using the concepts presented here. This model corresponds to an architecture that is seen in commercial products, such as: Secure Perfect, Picture Perfect and Diamond II. BACnet is a standard that includes access control to buildings and incorporates RBAC, zones, and schedules [2].

## 22.12. See Also

This pattern is a combination of the Role based access control (or another suitable authorization model), the Access Control for Physical Structures pattern [1], the Alarm Monitoring pattern, and the Relays pattern.

## 22.13. References

[1] Desouza-Doucet, A. C. (2006). *Controlling access to physical locations*. Florida Atlantic University.

[2] Ritter, D., Isler, B., Mundt, H., & Treado, S. (2006). Access Control In BACnet®. *Ashrae Journal*, 48(11), B26.

## 22.14. Sources

Fernandez, E. B., Ballesteros, J., Desouza-Doucet, A. C., & Larrondo-Petrie, M. M. (2007, July). Security patterns for physical access control systems. In *IFIP Annual Conference on Data and Applications Security and Privacy* (pp. 259-274). Springer, Berlin, Heidelberg.

# 23. Discretionary Access Control

## 23.1. Intent

The DAC pattern enforces access control based on user identities and the ownership of objects. The owner of an object may grant permission to another user to access the object, and the granted user may further delegate the permission to a third person.

## 23.2. Example

Consider an environment where access control is solely managed by the security administrator. A problem in such an environment is that it requires much effort for the administrator to maintain access control for every single user and to deal with daily-basis requests of permission changes. Another problem related to confidentiality is that the administrator may give unreasoned permission to a person who is not supposed to be authorized. For example, in a medical care system, access to patient information should be kept confidential and limited to only the doctor who handles the case, or other doctors who have permission given by the handling doctor. If the security administrator receives a fallacious request for permission given to a clerk, obviously it should not be allowed. In regard to availability, such an environment may cause a situation where no one can access information. For example, in the medical care system above, if DoctorA who handles CaseFile1 has left for a vacation, without making a permission request for other doctors to access the file, no one can access the file in case of emergency. Figure 42 illustrates these problems.

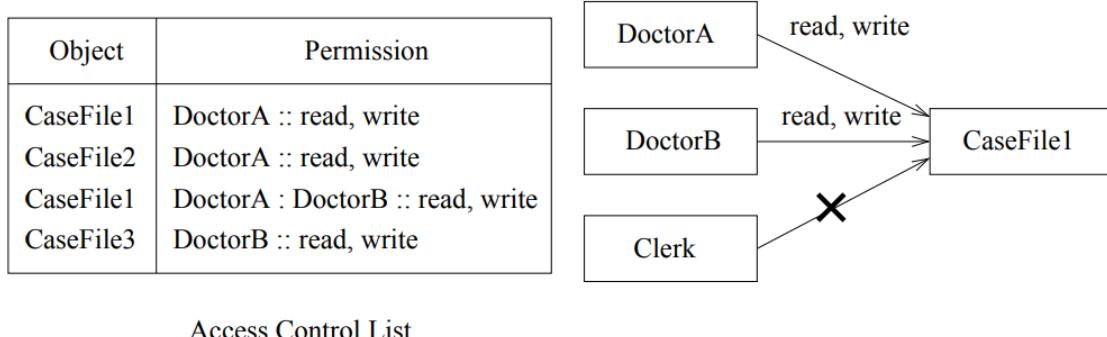


Figure 42: A Motivation Example

## 23.3. Context

Development of access control systems that allow user-controlled administration of access rights to objects.

## 23.4. Problem

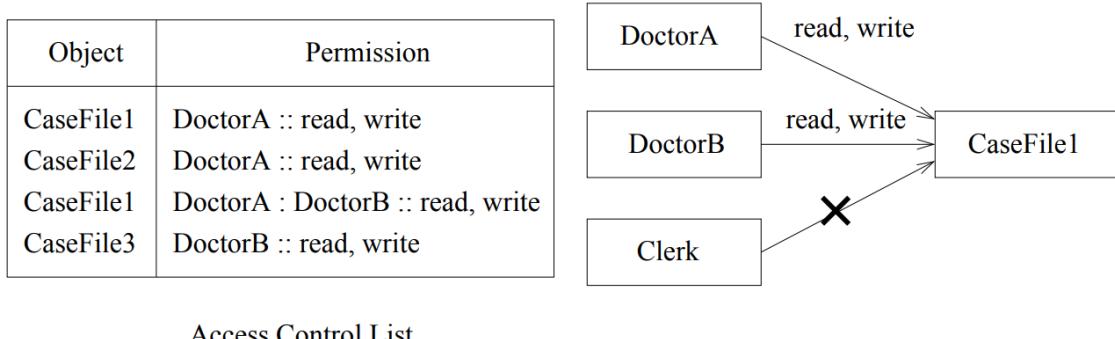
In an environment where access control is managed solely by administrators, unreasoned permission may be given to a person who should not be authorized, or no access may be allowed for anyone. In such cases, it may be desired to leave access control decisions to the discretion of the owner of the respective resources. Use the DAC pattern:

- Where users own objects.
- When permission delegation is necessary.
- When a resolution for conflicting privileges is needed. For instance, a user may be allowed to access an object as a member of a group, but not allowed with individual permissions.
- When a security mechanism is needed in a heterogeneous environment for controlling access to different kinds of resources.
- Where multi-user relational database is used.

### 23.5. Solution

The DAC pattern can address the above problems by using the concept of “permission delegation” which allows a user of an object to give away permission to other users to access the object at her/his discretion without the intervention of the administrator. Using the DAC pattern, the burden on the administrator is shared with the users of objects who are capable of delegating permission. The DAC pattern mitigates the confidentiality problem above by granting permission directly to related people in the area. Also, the availability problem above can be addressed by delegating permission at the discretion of object users.

Figure 43 shows an Access Control List (ACL) that implements the DAC pattern. In the ACL, a delegating user is represented in `user::rw` and a named user to which permission is delegated is represented in `user:namedUser::rw`. For example, DoctorA is a user of CaseFile1, and has read and write access to the file, and DoctorB is a named user who is granted access to CaseFile1 by DoctorA and has read and write access to the file.



*Figure 43: DAC Solution*

An inconsistency between the permission given as an individual and the permission given as a member of a group to which the user belongs can be resolved by evaluating permission in an order. The evaluation stops either when all requested access rights have been granted by one or more permission entries, or when any one of the requested access rights has been denied by one of the permission entries.

### 23.6. Structure

Figure 44 shows the solution structure of the DAC pattern.

- User represents a user or group who has access to an object, or a named user or group who are granted access to the object by the user or group. The owner or owning group

of an object has full access to the object and can grant or revoke permission to other users or groups at their discretion.

- Object represents any information resource (e.g., files, databases) to be protected in the system.
- Operation represents an action invoked by a user.
- Subject represents a process acting on behalf of a user in a computer-based system. It could also be another computer system, a node, or a set of attributes.
- Permission represents an authorization to carry out an action on the system. In general, Access Control Lists (ACLs) are used to describe DAC policies for its ease in reviewing. An ACL shows permissions in terms of objects, users, and access rights.
- ReferenceMonitor checks permission for an access request based on DAC policies. If the user has permission to the object, the requested operation may be performed.

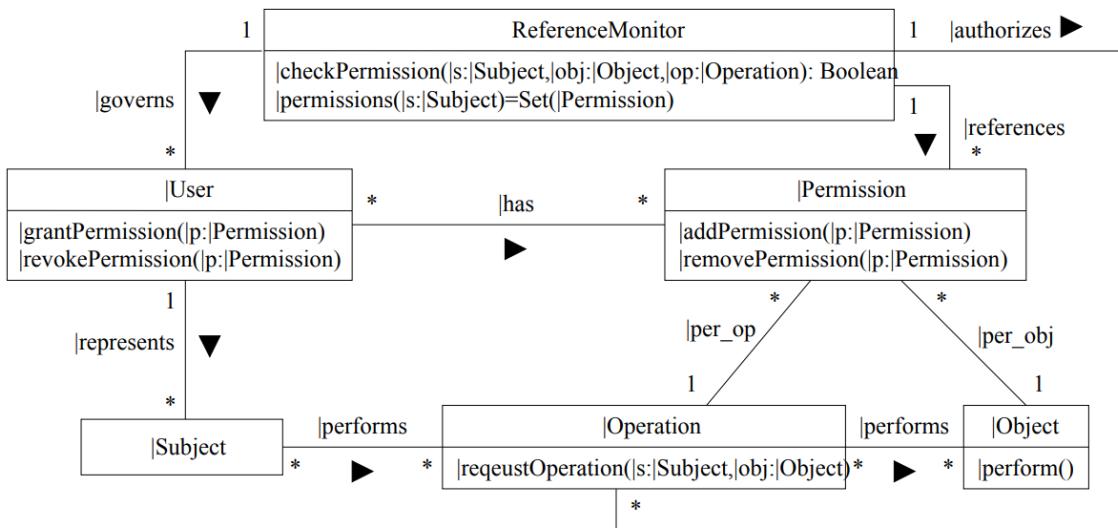


Figure 44: DAC Solution Structure

### 23.7. Dynamics

Figure 45 shows the collaboration of the DAC pattern for requesting an operation. When an operation is requested for an object, the reference monitor intercepts the request and checks for permission based on DAC policies which is typically described in ACLs. If permission is found, the operation is performed on the object, otherwise, the request is denied.

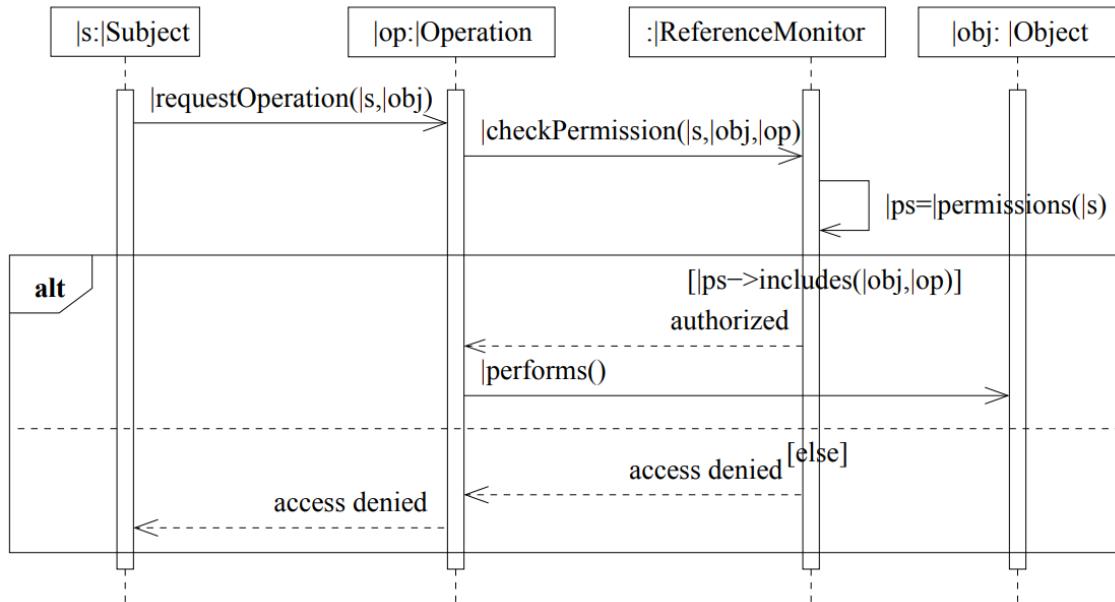


Figure 45: DAC Collaboration

## 23.8. Variants

Based on the underlying DAC concepts above, there are several variants [1] which are different by the degree of the strictness in owner's discretion. DAC variants can be categorized into strict DAC, liberal DAC, and DAC with change of ownership. Strict DAC is the strictest form in which only the owner of an object can grant access to the object. Liberal DAC allows the owner to delegate access granting authority to other users. Liberal DAC can be further divided into one level grant, two-level grant and multilevel grant, depending on the level at which access granting authority can be passed on. DAC with change of ownership allows the owner to delegate ownership to other users.

## 23.9. Implementation

## 23.10. Example Resolved

## 23.11. Consequences

The DAC pattern has the following advantages:

- Users can self-manage access privileges.
- The burden of security administrators is significantly reduced, as resource users and administrators jointly manage permission.
- Per-user granularity for individual access decisions as well as coarse-grained access for groups are supported.
- It is easy to change privileges.

- Supporting new privileges is easy.

The DAC pattern has the following disadvantages:

- It is not appropriate for multilayered systems where information flow is restricted.
- There is no mechanism for restricting rights other than revoking the privilege.
- It becomes quickly complicated and difficult to maintain access rights as the number of users and resources increases.
- It is difficult to judge the “reasonable rights” for a user or group.
- Inconsistencies in policies are possible due to individual delegation of permission.
- Access may be given to users that are unknown to the owner of the object. This is possible since the user granted authority by the owner can give away access to other users.

## 23.12. Known Uses

Standard Oracle9i [2] uses the DAC pattern to mediate user access to data via database privileges such as SELECT, INSERT, UPDATE and DELETE. The TOE [3], a sensitive data protection product developed by The Common Criteria Evaluation and Validation Scheme (CCEVS), uses the DAC pattern to mediate access to cryptographic keys to prevent unauthorized access. Windows NT implements the DAC pattern to control generic access rights such as No Access, Read, Change, and Full Control for different types of groups (e.g., Everyone, Interactive, Network, Owner).

## 23.13. See Also

The Authorization pattern which addresses accessing objects by subjects. The Authorization pattern has the concept of delegation as in the DAC pattern. However, unlike the DAC pattern, the access request may need not specify a particular object in the rule. It may be implied by the existing objects being protected.

## 23.14. References

- [1] Sandhu, R., & Munawer, Q. (1998, October). How to do discretionary access control using roles. In *Proceedings of the third ACM workshop on Role-based access control* (pp. 47-54).
- [2] Oracle9i. (n.d.) <http://www.oracle.com/technology/products/oracle9i/datasheets/ols/OLS9iR2ds.html>.
- [3] TOE. (n.d.) <http://niap.nist.gov/cc/-scheme/st/ST VID4048.html>.

## 23.15. Source

Kim, D. K., Mehta, P., & Gokhale, P. (2006, October). Describing access control models as design patterns using roles. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-10).

# 24. Mandatory Access Control

## 24.1. Intent

The MAC pattern governs access based on the security level of subjects (e.g., users) and objects (e.g., data). Access to an object is granted only if the security levels of the subject and the object satisfy certain constraints. The MAC pattern is also known as multilevel security model and lattice-based access control.

## 24.2. Example

The MAC pattern addresses the following two problems in the DAC pattern. First, there is nothing that prevents a user who is granted read access to a file by the owner of the file from copying the content of the file and granting read access to other users. For example, consider Figure 46. Let's suppose John wants to access File1 which he does not have permission to access. Of course, a DAC system will not allow him to access the file since he has no permission. However, if there is a third person who has read access to File1 granted by (Jane) who is the owner of File1, and the person grants John read access to File1, then John can read File1 without Jane being aware of it. Secondly, a user who has write access to a file may write a Trojan horse program into the file to copy the contents of the file. A Trojan horse program disguises as if it is a utility program while copying file contents. In Figure 46, the Trojan horse program copies the contents of File1 into File2 which John can access.

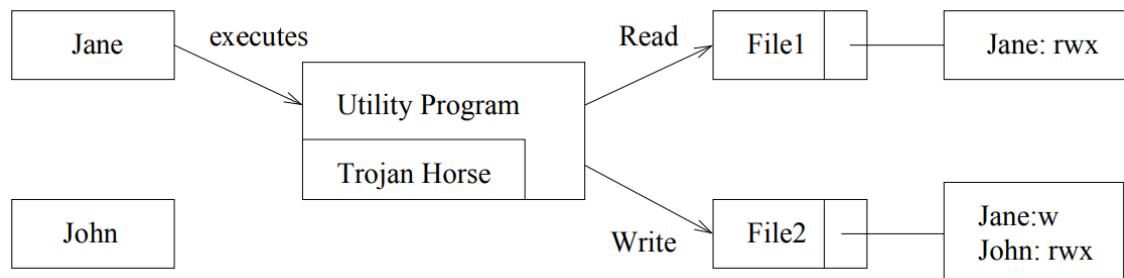


Figure 46: Trojan Horse Problem in the DAC Pattern

## 24.3. Context

Development of access control systems that handle classified objects and need to limit users' actions according to a hierarchy of classifications.

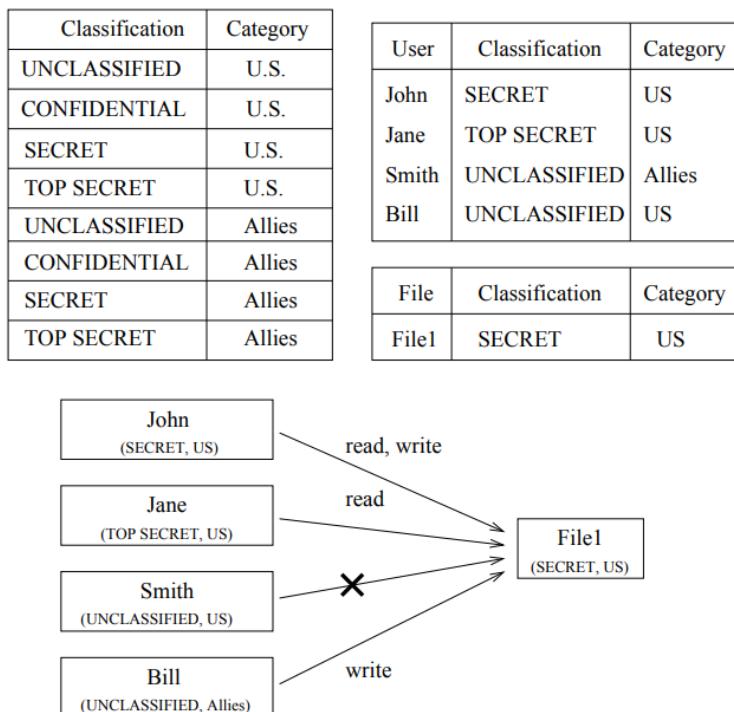
## 24.4. Problem

The MAC pattern can solve the above problems with the DAC pattern in a multi-layered environment (e.g., military and government systems) by assigning security levels to users and objects. In this sense, the solution of the DAC pattern can be considered as a problem of the MAC pattern. That is, if a DAC system is deployed in a multi-layered environment, the MAC pattern can be applied to improve the confidentiality of the system. Use the MAC pattern:

- Where the environment is multi-layered. For example, in the military domain, users and files are classified into distinct levels of hierarchy (e.g., Unclassified, Public, Secret, Top Secret), and user access to files is restricted based on the classification.
- When security policies need to be defined centrally. Access control decisions are to be imposed by a mediator (e.g., security administrator), and users should not be able to manipulate them.

## 24.5. Solution

The MAC pattern solves the above issues in a classified environment by assigning security levels to users and objects. In the MAC pattern, a user can read a file, but cannot write to the file if the user's security level is higher than or equal to the file's security level. A user can write to a file but cannot read the file if the user's security level is lower than or equals to the file's security level. For example, consider the military domain where documents are classified and categorized as shown in Figure 47.



*Figure 47: A MAX Example*

In the diagram, John has read and write access to File1 since his classification and category are same as that of File1. However, Jane can only read File1, but cannot write to the file because the classification of File1 is lower than that of Jane. Smith cannot read and write to File1 because his category is different from the category of File1. Bill can write to File1 but cannot read the file because the classification of the file dominates his classification.

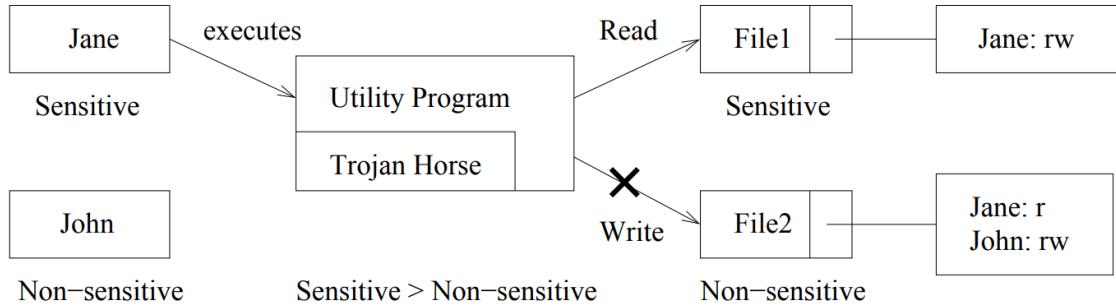


Figure 48: Trojan Horse Solution in the MAC Pattern

Using the MAC pattern, the Trojan horse problem in Figure 46 can be resolved as shown in Figure 48. In Figure 48, the Trojan horse program running on behalf of Jane can read File1 since the security level of Jane is equal to that of File1. However, it cannot write to File2 because Jane's security level is lower than the security level of File2. Thus, John is not able to access File1.

## 24.6. Structure

Figure 49 shows the solution structure of the MAC pattern [1].

- User represents a user or a group of users who interacts with the system. A user is assigned a hierarchical security level (e.g., SECRET, CONFIDENTIAL) and non-hierarchical category (e.g., U.S., Allies) to which the user belongs. A user may have multiple login IDs which can be activated simultaneously. A user also may create and delete one or more subjects.
- Subject represents a computer process that acts on behalf of a user to request an operation on an object. For instance, an ATM machine being used by a user can be viewed as a subject. A subject may be given the same security level as the user or any level below the user's security level.
- Object represents any information resource (e.g., files, databases) in the system that can be accessed by the user. Similar to users, an object is assigned a hierarchical security level and a non-hierarchical category to which the object belongs.
- Operation is an action being performed on an object invoked by a subject.
- SecurityLevel represents a sensitivity assigned to users (subjects) and objects. A security level consists of a classification and a category. While classifications are hierarchical, categories are non-hierarchical.
- ReferenceMonitor checks accessibility based on the following constraints.
  - Simple security property - A subject S is allowed read access to an object O only if  $L(S) \geq L(O)$ .
  - Star property - A subject S is allowed write access to an object O only if  $L(S) \leq L(O)$ .

Access is allowed when both the constraints are satisfied. Access is checked only if the user is in the same category as that of the object. With the categories matched, the accessibility of the user for the object is determined by the dominance relations of classifications in the above constraints.

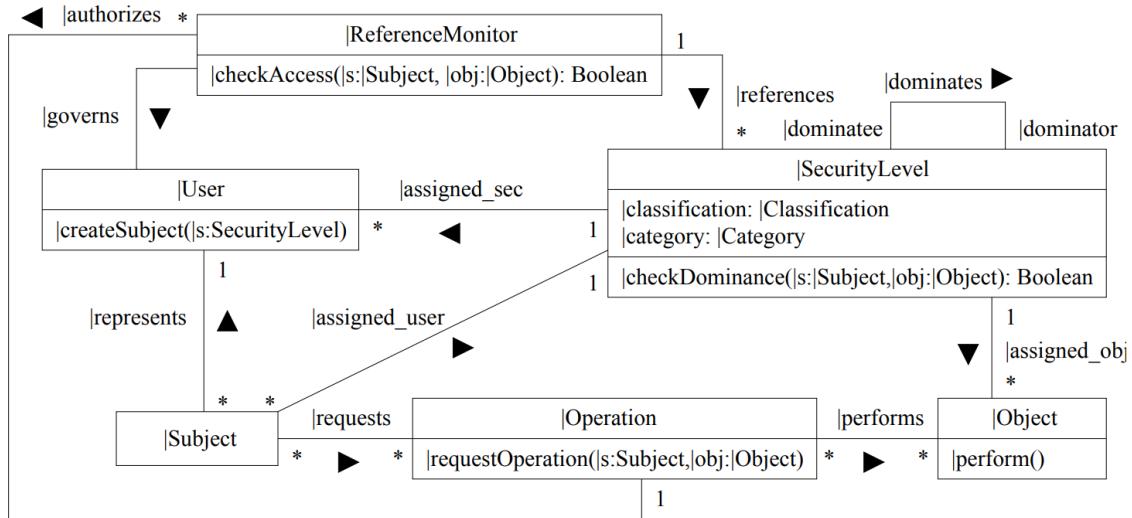


Figure 49: MAC Solution Structure

## 24.7. Dynamics

Figure 50 shows the collaboration for requesting an operation. The diagram describes that a subject requests an operation on an object, and the request is intercepted by the reference monitor to check authorization by enforcing the simple security property and star property. The request is performed on the object if it is authorized, otherwise it is denied.

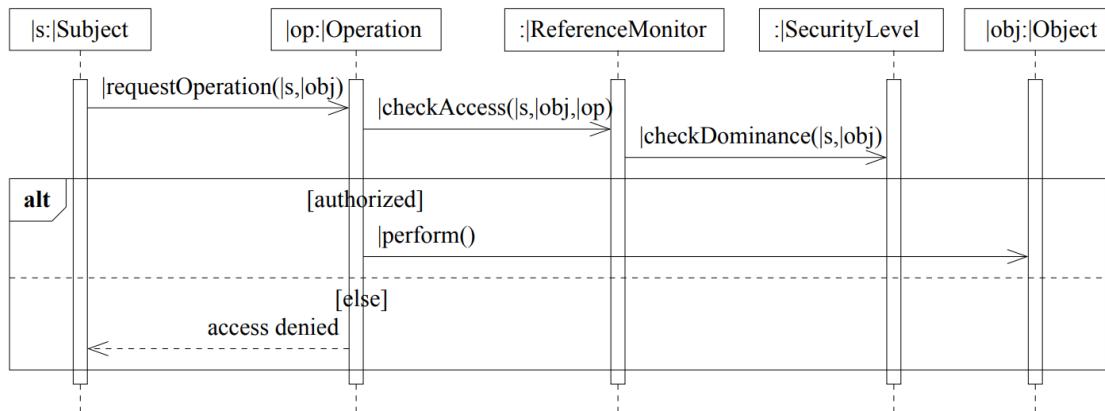


Figure 50: MAC Colloboration

## 24.8. Variants

The Biba Integrity model [2] can be viewed as a variant of the MAC pattern, emphasizing on integrity rather than confidentiality. Similar to the MAC pattern, Biba model has Simple Integrity property and Integrity Star property which are defined as follows:

- Simple integrity property - A subject S is allowed read access to an object O only if  $L(O) \geq L(S)$ .
- Integrity star property - A subject S is allowed write access to an object O only if  $L(O) \leq L(S)$ .

## **24.9. Implementation**

## **24.10. Example Resolved**

## **24.11. Consequences**

The MAC pattern has the following advantages:

- MAC systems are secure to Trojan horse attacks.
- The assignment of a classification and category to users and objects is centralized by a mediator.
- The MAC pattern facilitates enforcing access control policies based on security levels.

The MAC pattern has the following disadvantages:

- Introducing a new object or user requires a careful assignment of a classification and category.
- The mediator who assigns classifications to users and objects should be a trusted person.

## **24.12. Known Uses**

Security-Enhanced Linux (SELinux) kernel [3] developed by a collaboration of NSA, MITRE Corporation, NAI labs and Secure Computing Corporation (SCC) enforces the MAC pattern to implement a flexible and fine-grained MAC architecture called Flask which operates independently of the traditional Linux access control mechanisms. TrustedBSD [4] developed by the FreeBSD Foundation provides a set of trusted operating system extensions to the FreeBSD operating system which is an advanced operating system for x86, amd64 and IA-64 compatible architectures. TrustedBSD contains modules that implement MLS (Multi-Level Security) and fixed-label Biba integrity policies which is a variant of the MAC pattern. GeSWall (General Systems Wall) [5] is the Windows security project developed by GentleSecurity. GeSWall implements the MAC pattern to provide OS integrity and data confidentiality transparent and invisible to user.

## **24.13. See Also**

The Biba's Integrity model [2] which addresses integrity issues where access is determined by the integrity levels of subjects and objects, rather than confidentiality. The Chinese Wall model [6] which is similar to the MAC pattern in that it also defines read and write constraints where the write constraint considers the Trojan horse problem. However, unlike the MAC pattern, the Chinese Wall model does not distinguish between users and subjects. Subjects include both users and processes acting on behalf of the user.

## **24.14. References**

- [1] Kim, D. K., & Gokhale, P. (2006, September). A pattern-based technique for developing UML models of access control systems. In *30th Annual International Computer Software and Applications Conference (COMPSAC'06)* (Vol. 1, pp. 317-324). IEEE.
- [2] Biba, K. J. (1977). *Integrity considerations for secure computer systems*. MITRE CORP BEDFORD MA.
- [3] SELinux. (n.d.). <http://www.nsa.gov/selinux/>.
- [4]. TrustedBSD. (n.d.). TrustedBSD Project. <http://www.trustedbsd.org/>.
- [5] General System Wall. (n.d.). <http://www.securesize.com/GeSWall/>.
- [6] Brewer, D. F., & Nash, M. J. (1989, May). The Chinese Wall Security Policy. In *IEEE symposium on security and privacy* (Vol. 1989, p. 206).

## **24.15. Source**

Kim, D. K., Mehta, P., & Gokhale, P. (2006, October). Describing access control models as design patterns using roles. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-10).

# 25. Anonymity Set

## 25.1. Intent

Hide the data by mixing it with data from other sources.

## 25.2. Example

**The Athenian Mistake.** In a message communication scenario, the content of the message is not always important. Merely the fact that a sender is sending a message can reveal important information. Suppose there are two regions Athens and Sparta, that are going through troubled times. The threat that one can launch a preemptive attack on the other is imminent. The Athenian army hired a veteran cryptographer who devises an unbreakable cipher. The intelligence branch of Sparta has not been able to decrypt this cipher scheme, but they have under-cover probes that let them know who is sending message to whom, a correlation that the Athenian army is not choosing to conceal.

Deep into one night, Athens decides to launch the attack the next morning. Suddenly there is a fury of messages passing among the chain of command of the Athenian army. Spartan intelligence picks up the information that suddenly the Athenian Generals have become active late into the night. They mobilize their army that night. Athens was not prepared for the counterstrike. They lose the battle.

**Protected Health Information.** Health Insurance Portability and Accountability Act (HIPAA) [1] of 1996 defines the appropriate way to handle Protected Health Information (PHI). For research purpose, Cure Clinic is releasing its PHI about cancer victims to Acme Laboratories. Let us suppose that Cure Clinic was keeping the patients' name, birth date, sex, zip code and diagnostics record. To protect privacy, Cure Clinic does not release the name of the patients. The birth date, sex, zip code and diagnostics records are released. Acme Laboratories is doing the research on the probability of cancer attack on a particular age group and sexual orientation over the people of a particular locality. So, all the data fields that are released are important for the research. Mallory is a malicious worker at Acme Laboratories who wants to unravel private information from this data. Mallory goes to the city council of a particular area and gets a voter list from them. The two lists are matched for age, sex, and locality. Mallory finds the name and address information from the voter registration data and the health information from the patient health data.

## 25.3. Context

You are designing a system to protect the privacy of the users. This system will maintain sender and/or recipient anonymity in a messaging scenario. Although this is an important application area, the context is not limited to messaging only. The context also entails other scenarios like anonymity in a location tracking system, anonymous voting in an electronic voting system, or anonymity preserving data sharing in a data publishing system.

## **25.4. Problem**

It is difficult to ensure sender and recipient anonymity during message communication. The only concern of traditional security approaches is to protect data content. This does not hide the message path from the sender to the receiver and thus the anonymity is compromised.

In an electronic voting system or an online voting system, the system should protect the privacy of the voters by not revealing their vote.

Similarly, a user may want to hide his information from a location tracking system. This may be because the location tracking device is offering some context-aware service based on user location, but the user is not interested at the moment, or may be because the user is not trusting the location tracking system or may be because the user does not want to reveal his private location information at all.

When private datasets are released, the private data about the subjects may be exposed. The released dataset has to be sufficiently rich in order to be useful, but it also should protect the privacy of the entities.

In all the cases, the general problem is to ensure the anonymity. How can the anonymity of an entity or a personal information be retained?

The forces that need to be considered when choosing to use this pattern are as follows.

- User Count. The number of users using an anonymous messaging system may vary with time. This fluctuation may depend on operational hours, user interest, etc. If the user flow is low, the solution does not work because there are not enough candidates to create the anonymity set.
- User Friendliness. Users should be able to use the privacy enabling mechanisms with minimal alteration of their primary task. If the users have to adapt a lot to achieve anonymity, they may start judging where they should have anonymity. This way, a user's misjudgment can sometimes reveal private information.
- Data Usability. An anonymous data set has to be usable. One extreme of achieving anonymity is not to release any data, but obviously this is not a usable scheme.
- Performance. The privacy retaining operations should not become a performance bottleneck (i.e. latency, bandwidth etc.). For anonymous messaging, the system should be usable in low latency usage scenarios, like web browsing.
- Law Enforcement. Law enforcement agencies might require that the anonymity solutions sometimes lift their anonymity cover to investigate on crime suspects. This would prevent a malicious user from abusing anonymity

## **25.5. Solution**

Mix the private information with other information so that the private information is not distinguishable from other information. Create a set of equally probable information and hide the user information by making it a part of the set. This set is called the anonymity set. If the size of the anonymity set is large, it will ensure stronger privacy.

For message communication, create an abstract mechanism between the message sender and the recipient that hides the correlation between the sender and the recipient. This can be done by network intermediaries called mix networks that mix the message coming from one

source with messages coming from different other sources. Once the data packet from the sender passes through one of these filters, it is indistinguishable from other packets (i.e. sender anonymity). The anonymity set is the set of messages from different sources. Figure 51 illustrates this solution. Alice's data is collected at the mix node and mixed with Bob and Carol's data.

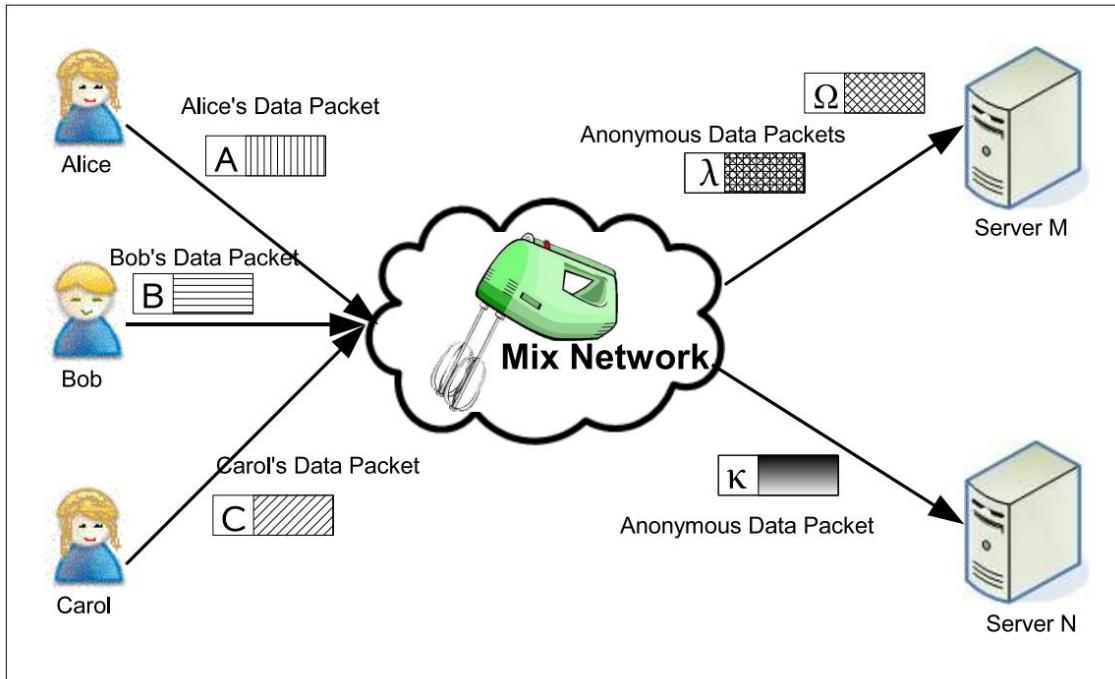


Figure 51: Sender anonymity in a mix network

For recipient anonymity in message communication, broadcast the message to all users or send the message to a message pool instead of one single recipient. The recipient will view the message like everyone else, but an adversary will not be able to tell who that message is for.

Use the same idea for sender anonymity in a location monitoring system. Install the abstract obfuscation mechanism in a region. Agents are identified by pseudonyms in the location tracking system. Once an agent enters the region where the obfuscation mechanism is installed, he is given a different pseudonym. If there are multiple agents in the obfuscation region and all of them adopt a different pseudonym upon entry, then the agents in the region create the anonymity set at a particular point of time. Once the agents come out of the region, their new pseudonyms cannot be correlated with the old pseudonyms.

In an anonymous voting system, a voter's vote is passed through an obfuscation mechanism where it is mixed with other votes and the generated outcome leaves no trace that can be used to link the voter with the vote.

When releasing private data sets for public use, change the specific values of attributes that might reveal private information to more generalized values. If the dataset has a specific gender information, change the values of gender (male/female) to more general values (person). Also partition the attribute domain space and provide partitioned information rather than exact information. For example, if the dataset has specific age information, create age groups, and convert the dataset such that the specific ages are replaced with age groups. What

this mechanism is doing is that it is creating an anonymity set so that one row in the dataset becomes indistinguishable from another.

## 25.6. Structure

## 25.7. Dynamics

## 25.8. Implementation

**Size of the Anonymity Set.** The size of the anonymity set will determine how good the obfuscation will be. If the set is small, then correlation between input and output of the obfuscation mechanism can be determined with higher probability. If the anonymity set has only one element, then the privacy is provably compromised.

**Latency.** In a messaging scenario, the mixing mechanism might stall the data traffic to wait for enough data packets to arrive so that the mixing can be done effectively. This means that the latency of the data flow increases, which might make it unusable in a low latency messaging scenario like web browsing. Different strategies can be taken to counter the latency issue. The required degree of privacy is scenario specific and based on that the designer can identify the trade-off between privacy and performance.

**Usability of Information.** In the case of dataset release, absolute data obfuscation is possible by replacing all the specific attribute values with more general values. But this way the datasets may not be useful. For example, if a research is interested in the impact of sexual orientation on cancer attacks and the dataset has been anonymized in a way that all the gender values are replaced with a more general 'person' value, then this dataset becomes useless for the research purpose. So, an anonymity protection mechanism that retains the usability of data is required.

## 25.9. Example Resolved

## 25.10. Consequences

The pattern has the following benefits.

- Privacy. The obfuscation mechanism ensures that private information is not easily compromised. Not all mechanisms provide absolute privacy, but they ensure that the attacker will have to do more work to break into the system's sensitive information.
- Freedom from User Profiling. Business entities are interested in user profiling to make smart advertisements. Users may not want to be bothered by these marketing suggestions. An anonymous user is free from such user annoying sales mechanisms.

- Minimal user involvement. The users do not have to modify their normal activities to get anonymity service. The service is provided by proxies resident at the user end and the intermediaries in the network. Usability is improved because of this transparency.

The pattern has the following liabilities.

- Performance. In the messaging domain, when the system is waiting for enough probable suspects to arrive to mix with incoming traffic, the users experience increased latency. Cover traffic can be used to create dummy probable suspects. But maintenance of a cover traffic flow is expensive in the bandwidth. Also, the mix nodes might employ a batched transaction strategy that causes flush traffic out of the anonymity nodes. In that case, bandwidth becomes a big factor.  
For data anonymization, it has been proven that general data obfuscation mechanism is NP-Hard [2]. In a location anonymity system, adding effective obfuscation mechanism (by introducing cover traffic) is very computation extensive.
- Usability of Information. Too much data obfuscation can undermine the usefulness of data. In the case of private dataset publishing, if all the attributes of the dataset are anonymized such that they retain privacy, the resultant dataset may not be useful at all. Queries granularity (that may be important for the research for which the dataset was made public initially) cannot be served.
- Abuse of Privacy. Anonymity systems are open to abuse by malicious users. An anonymous sender might be encouraged to send a hate mail in a public forum showing his ethnic bias. Sensitive information about a person can be posted anonymously to commit a smear attack. Terrorists might want to use the anonymity mechanism to communicate between themselves. Strong privacy guarantee for the end user makes the task of crime-fighting very difficult.

## 25.11. Known Uses

The Mix based networks [3] are based on the idea of mixing the incoming data traffic from one user with the data traffic coming from other users. Each mix has a public key which is used by the message senders to encrypt the message between the user and the mix. The mix accumulates these messages, decrypts them, optionally re-encrypts the messages and delivers them to the subsequent node. If there is a sufficient amount of input data packet from different sources, the mix ensures that the sources cannot be linked with the data packets once the packets come out of the mix. Users should not trust only one mix. Instead, they should send their data through a cascade of mixes. In this way, weak anonymity is preserved even if some of the mixes are honest (i.e. not run by an adversary). The first widespread public implementation of mixes was produced by contributors of the Cypherpunks mailing list [4]. Then Mixmaster [5] and Babel [6] were based on the mix network idea to send anonymous emails. These systems are called remailers. Mixmaster was a Type II remailer (Cypherpunks remailers were Type I remailers). Mixminion [7] is a Type III remailer that addresses the problems of previous generations of remailers like the sophisticated flood and trickle attacks. The types associated with the remailers generate different generations of these remailers.

Onion Routing [8,9] systems are based on mixes, but they leverage the idea of mixes and add layered encryption. Onion Routing systems have better latency values than mix networks and therefore are more applicable in a web browsing scenario. Crowds [10] is another system that provides low-latency data anonymization. Its approach is different from onion routing or mix

schemes. Here, every node in the network is similar and every node can either forward the data packet to another node in the network or send the request to the end server based on a probabilistic coin toss. Thus, every node is a probable suspect of being the originator. In this way, Crowds provides plausible deniability for the message sender.

Hordes [11] uses multicast routing where every responder gets every message. This provides recipient anonymity.

The Votegrity system based on by Chaum's secret-ballot receipts [12] and the VoteHere system based on Neff's secret shuffle algorithm [13] use the concept of mix networks for mixing the votes.

Mix zone [14, 15] is the same concept of mix networks taken into the domain of location anonymity. The mix zone concept was added with the Active Bat [16] system to provide location anonymity.

The principle of k-Anonymity [17] was introduced by Latanya Sweeney for publishing of secret data. The sensitive information in a dataset is obfuscated by replacing them with a more general information.

## 25.12. See Also

MORPHED REPRESENTATION pattern is used with ANONYMITY SET pattern to hide the correlation between incoming and outgoing traffics.

## 25.13. References

- [1] US Department of Homeland Health Services Office for Civil Rights. (May 2003). Summary of the HIPAA privacy rule.
- [2] Ryan, A. M., & Williams, R. (2003). General k-anonymization is hard. In *In Proc. of PODS'04*.
- [3] Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 84-90.
- [4] APAS Anonymous Remailer Use [FAQ 3/8]: Remailer Basics. (2001, December). Computer Cryptology. <http://www.faqs.org/faqs/privacy/anon-server/faq/use/part3/>
- [5] Cotrell, L. (1995). Mixmaster & remailer attacks. <http://riot.eu.org/anon/doc/remailer-essay.html>.
- [6] Gulcu, C., & Tsudik, G. (1996, February). Mixing E-mail with Babel. In *Proceedings of Internet Society Symposium on Network and Distributed Systems Security* (pp. 2-16). IEEE.
- [7] Danezis, G., Dingledine, R., & Mathewson, N. (2003, May). Mixminion: Design of a type III anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003*. (pp. 2-15). IEEE.
- [8] Goldschlag, D. M., Reed, M. G., & Syverson, P. F. (1996, May). Hiding routing information. In *International workshop on information hiding* (pp. 137-150). Springer, Berlin, Heidelberg.

- [9] Syverson, P. F., Goldschlag, D. M., & Reed, M. G. (1997, May). Anonymous connections and onion routing. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)* (pp. 44-54). IEEE.
- [10] Reiter, M. K., & Rubin, A. D. (1998). Crowds: Anonymity for web transactions. *ACM transactions on information and system security (TISSEC)*, 1(1), 66-92.
- [11] Shields, C., & Levine, B. N. (2000, November). A protocol for anonymous communication over the internet. In *Proceedings of the 7th ACM Conference on Computer and Communications Security* (pp. 33-42).
- [12] Chaum, D. (2004). Secret-ballot receipts: True voter-verifiable elections. *IEEE security & privacy*, 2(1), 38-47.
- [13] Neff, C. A. (2001, November). A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security* (pp. 116-125).
- [14] Beresford, A. R., & Stajano, F. (2004, March). Mix zones: User privacy in location-aware services. In *IEEE Annual conference on pervasive computing and communications workshops, 2004. Proceedings of the Second* (pp. 127-131). IEEE.
- [15] Beresford, A. R., & Stajano, F. (2003). Location privacy in pervasive computing. *IEEE Pervasive computing*, 2(1), 46-55.
- [16] Ward, A., Jones, A., & Hopper, A. (1997). A new location technique for the active office. *IEEE Personal communications*, 4(5), 42-47.
- [17] Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05), 557-570.

## 25.14. Sources

- Hafiz, M. (2006, October). A collection of privacy design patterns. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-13).

# 26. Morphed Representation

## 26.1. Intent

Change the representation of the data when it is passing through an anonymity providing node so that outgoing data cannot be linked with incoming data.

## 26.2. Example

The unsuccessful mix. Alice, Bob, and Carol are using a mix-based system to communicate over the Internet. Alice sends her data through a node in the network where it gets mixed with the data coming from other sources (e.g. Bob, Carol, etc.). Figure 52 shows that Mallory is a passive observer of the mix network, and she wants to find out all the correspondences of Alice.

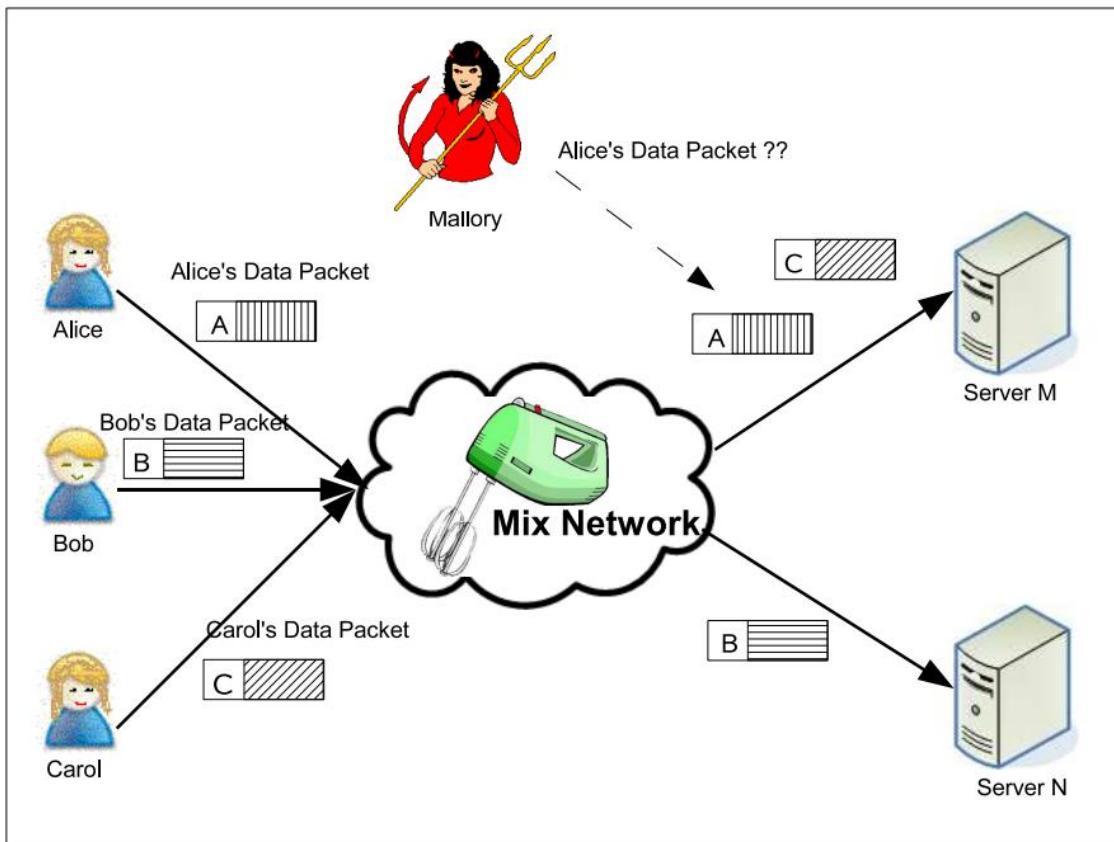


Figure 52: The unsuccessful Mix network

Alice encrypts her data with the public key of the recipient (in this case server M) to keep it confidential. The mix network receives Alice's packet, waits for other packets to arrive, and then releases a bunch of packets together. However, the incoming and outgoing packets have the same data fields, hence it is easy for Mallory to find out who is sending which packet. Mallory can profile everyone's messaging habit very easily.

## **26.3. Context**

You are designing a mix-based system to protect the privacy of the users. You want to have sender anonymity and sender and receiver unlinkability for the communicating parties. The mix-based system can be a mix-based filter in the Internet messaging domain that is used for email messaging or web browsing.

## **26.4. Problem**

Mix networks combine the data from a sender with the data coming from multiple other sources and send them together. The incoming data packets carry the data content. They also have meta characteristics associated like the time of packet generation, ingress order of packets etc. The mix network obfuscates these meta characteristics by adding random delays to the ingress packets, or by batching a number of ingress packets before releasing the packets. However, if the mix concepts do not obfuscate the data content, the incoming and outgoing data packets have the same representation, and they can be correlated trivially. This compromises sender anonymity as the packets can be linked to the packet generator if an adversary has enough capability to trace the packet path back to the sender. Also, the packets can be traced to the recipient and sender and receiver unlinkability is compromised.

How can the representation of the data be obfuscated?

This pattern addresses the following forces.

- **Packet Characteristics.** The size and content of the data packet separates one packet from another. It is highly improbable that the size and content of two packets would be the same because timestamps are associated with packets. Even if the data inside the packet is protected by encryption, the encrypted content and size of packets reveal the correlation of the outgoing packets with the incoming packets.
- **Scalability.** The Mix networks should be scalable. A PKI infrastructure should be established between the participating nodes (i.e. mix nodes and end nodes) such that symmetric encryption keys can be exchanged during data transfer.
- **Confidentiality of Data.** Data flow in the network is encrypted to retain confidentiality of content.
- **Data Corruption.** The mix nodes should not change the data content, only change the representation of the data content to achieve unlinkability.
- **Type of Adversary.** A global, passive adversary monitors the data traffic in the network and does not manipulate the data content. Active adversaries may control the mix nodes. Mix nodes can also be controlled by passive (or semi-honest) adversaries that adhere to the mix protocol but only monitor the data content passing through the node.
- **Performance.** The privacy retaining operations should not become a performance bottleneck. For anonymous messaging, the system should be usable in low latency usage scenarios, like web browsing.

## **26.5. Solution**

Change the representation of the incoming data packet such that the outgoing packets look different from incoming packets. The incoming packets are encrypted using a key shared

between the sending node and the mix node. At the mix node, decrypt the packet and then re-encrypt it with the key shared between the mix node and the subsequent node.

Figure 53 shows the message transfer between Alice and Server N. Alice encrypts her packet with a shared key between her and the mix node. The mix node decrypts the packet and then re-encrypts it with the shared key between the mix and server M. Mallory is monitoring the network and she cannot identify the packets because the packet does not look the same.

Use symmetric keys for encryption to avoid the expensive primary key operations. The nodes set up a key share with all its neighbors during the setup phase.

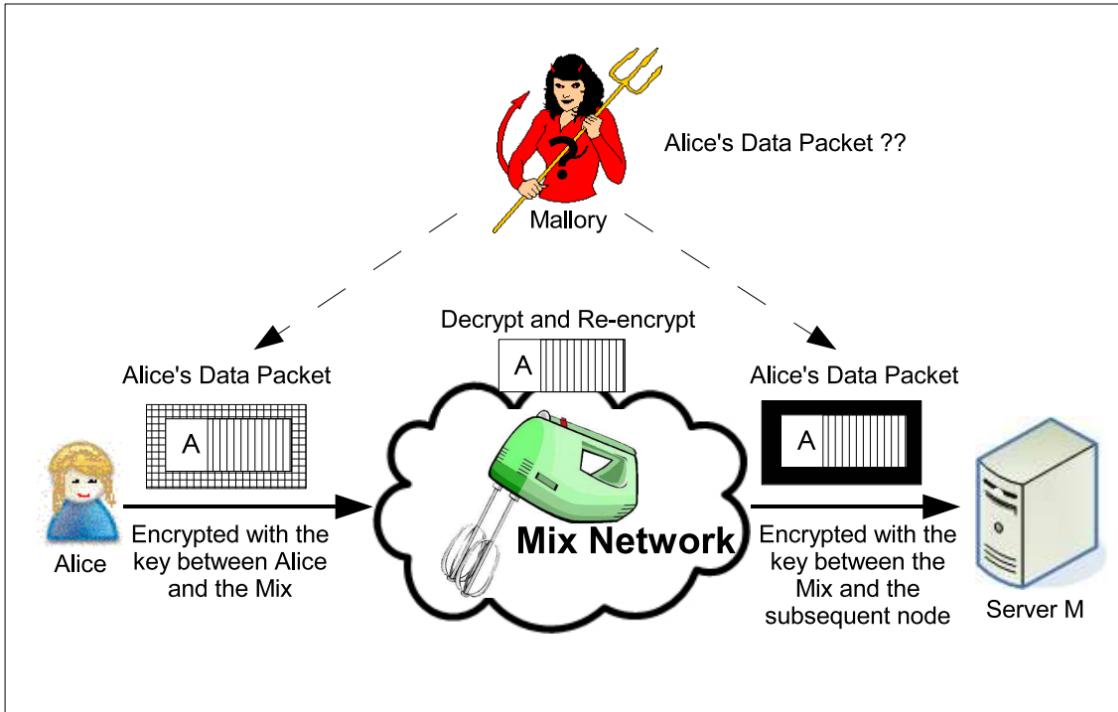


Figure 53: Data packet morphing at a Mix node

## 26.6. Design Issues

**Key Sharing.** The nodes in the network have to establish a symmetric key share with their neighbors. This symmetric key share can be established using public key certificates. However, the deployment of a global PKI infrastructure is an additional overhead for the scheme to be successful. To avoid the use of public key based key share establishment, lightweight secret sharing schemes like Diff e-Hellman key exchange [1] can be used.

**End-to-end encryption.** Since the packets are decrypted and re-encrypted in the mix nodes, confidentiality might be compromised if the data is in plaintext after the decryption. In that case even a semi-honest adversary running the mix node can compromise the privacy of sender and recipient and the confidentiality of the data. To avoid this, data has to be encrypted end-to-end. The sender encrypts the plaintext content with the public key of the ultimate recipient and then uses the symmetric key share to route it through the intermediaries.

**Size of Neighbor Set.** The scheme depends on all the nodes keeping a symmetric key share with their neighbors. A large list of neighbors (i.e. a large anonymity set) would ensure better

anonymity because the node has many options to choose from for the next hop. However, a large list would add maintenance overhead of key share tables.

## **26.7. Structure**

## **26.8. Dynamics**

## **26.9. Implementation**

## **26.10. Example Resolved**

## **26.11. Consequences**

The pattern has the following benefits.

- Privacy. The sender enjoys improved privacy because the representation of the data changes at every intermediate node. A single adversary can break the anonymity if he can observe the network globally which is fundamentally infeasible. The mechanism is also safe from colluding adversaries unless they are distributed globally and control the whole network. As long as there is one honest mix, it will obfuscate the correlation between input and output data traffic.

The pattern has the following liabilities.

- Performance Overhead. Performance overhead comes from two things - overhead of creating symmetric key shares and overhead of cryptographic operations at each mix node. The key sharing is often done beforehand to avoid the overhead during data transfer. There are several trade-offs to consider determining the lifetime of the symmetric key share. If the symmetric keys are used for a long time, then the system becomes vulnerable to brute force attack on the key. If the keys have a short lifetime, then the key setup overhead would be considerably high. Public keys can be durable, but they would involve a high computational overhead.
- Denial of Service. The active adversaries controlling the mix can drop the packets and create a denial-of-service scenario. Without the presence of a network management component, it would be very difficult to find the misbehaving node.

## **26.12. Known Uses**

The Mix based networks [2] are based on the idea of mixing the incoming data traffic from one user with the data traffic coming from other users. To hide the correlation, the incoming data

in the mix network is decrypted and then re-encrypted so that the egress traffic and the ingress traffic cannot be matched. Remailers based on the mix network principle, e.g. the Cypherpunks mailing list [3], Mixmaster [4], Babel [5], Mixminion [6] etc., follow this pattern to hide the correlation between incoming and outgoing packets.

Onion Routing [7, 8] systems are based on the concept of mixes, but they have better latency values than mix networks and are therefore more applicable in a web browsing scenario. Private web browsing systems for peer-to-peer communication that provide anonymity following this pattern include Morphmix [9], Tarzan [10] etc.

## 26.13. See Also

MORPHED REPRESENTATION is used with ANONYMITY SET to hide the correlation between incoming and outgoing traffic. Sometimes the data traffic is encrypted with LAYERED ENCRYPTION so that MORPHED REPRESENTATION does not compromise data confidentiality.

## 26.14. References

- [1] Diffie, W., & Hellman, M. E. (1976). New Directions in Cryptography. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 22(6).
- [2] Chaum, D. L. (1981). Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 84-90.
- [3] APAS Anonymous Remailer Use [FAQ 3/8]: Remailer Basics. (2001, December). Computer Cryptology. <http://www.faqs.org/faqs/privacy/anon-server/faq/use/part3/>
- [4] Cotrell, L. (1995). Mixmaster & remailer attacks. <http://riot.eu.org/anon/doc/remailer-essay.html>.
- [5] Gulcu, C., & Tsudik, G. (1996, February). Mixing E-mail with Babel. In *Proceedings of Internet Society Symposium on Network and Distributed Systems Security* (pp. 2-16). IEEE.
- [6] Danezis, G., Dingledine, R., & Mathewson, N. (2003, May). Mixminion: Design of a type III anonymous remailer protocol. In *2003 Symposium on Security and Privacy, 2003*. (pp. 2-15). IEEE.
- [7] Goldschlag, D. M., Reed, M. G., & Syverson, P. F. (1996, May). Hiding routing information. In *International workshop on information hiding* (pp. 137-150). Springer, Berlin, Heidelberg.
- [8] Syverson, P. F., Goldschlag, D. M., & Reed, M. G. (1997, May). Anonymous connections and onion routing. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)* (pp. 44-54). IEEE.
- [9] Rennhard, M., & Plattner, B. (2002, November). Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society* (pp. 91-102).
- [10] Freedman, M. J., & Morris, R. (2002, November). Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (pp. 193-206).

## **26.15. Source**

Hafiz, M. (2006, October). A collection of privacy design patterns. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-13).

# 27. Hidden Metadata

## 27.1. Intent

Hide the meta information associated with data content that reveal information about sensitive data content.

## 27.2. Example

Exposure. Alice is suffering from a medical condition, and she wants to find some information about it. She visits the website [www.dishonest-medical-website.org](http://www.dishonest-medical-website.org) that is controlled by Mallory (figure 54). While Alice is visiting the website, Mallory secretly gathers Alice's email address, geographical location, computer type, operating system, web browser, previous web site visited etc. Mallory is gathering these information about Alice even if Alice does not accept cookies, which is primarily used for profiling browser behavior. Mallory analyzes the `HTTP_USER_AGENT`, `REMOTE_HOST`, and `HTTP_REFERER` variables, which almost all web browsers provide to each site visited as part of the HTTP protocol.

`HTTP_USER_AGENT` reveals the user's browser software, which the remote web site could use to generate web pages specifically tailored to the browser's capabilities. However, both Netscape and IE also include the user's computer type and operating system as part of this variable.

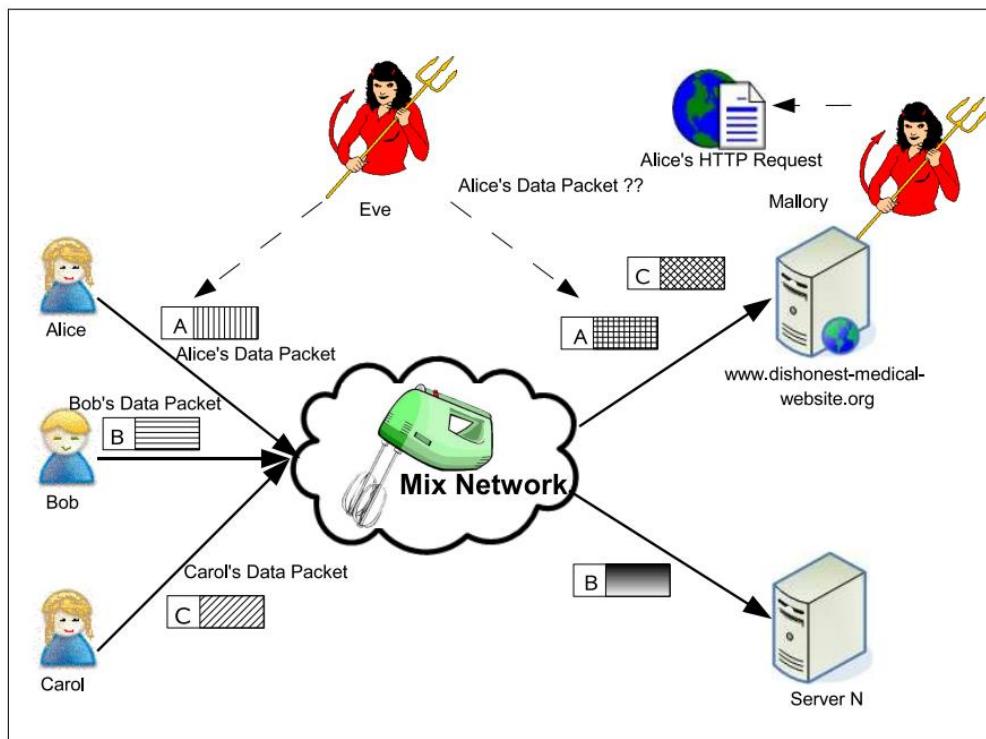


Figure 54: Compromised Anonymity by Header Matching

The `REMOTE_HOST` variable reveals the Internet address of the computer making the request for a web page. Let us assume that Alice is running a single-user workstation. The computer's identity may be the key to an enormous source of personal information. Using the Unix 'finger'

command, Mallory can identify the Alice's full name, email address, and even phone numbers. Even if Alice is accessing the web via a large commercial provider such as AOL, or from behind a corporate firewall, the REMOTE\_HOST field reveals that the user is an AOL member or an employee of that particular company. People who access the web via local Internet service providers (ISPs) reveal the identity of that ISP, which in turn reveals their geographic location. Mallory then performs a 'whois' lookup from the InterNIC database to find and report the physical address associated with the user's Internet host.

The HTTP\_REFERER variable reveals the previous page visited by Alice. All of these information's are provided without Alice's consent or knowledge and thus is a threat to Alice's privacy. Mallory also combines the data with another publicly accessible database such as a phone directory, marketing data, voter registration list, etc. She gains a significant amount of personal data on every visitor to her pages. All of this information-gathering is accomplished without Alice's authorization or awareness.

Other than the application layer protocol attacks, the data packets carrying the request from Alice to the end host are also vulnerable to inspection by a global passive eavesdropper Eve. These packets contain the IP addresses for routing. Figure 54 shows that Alice is sending the packets through a Mix network along with other users Bob and Carol. The input data packets to the mix are encrypted. The mix decrypts the data, re-encrypts it and sends them out after adopting a mix strategy. However, the IP addresses associated with the data packets are needed for routing. Eve does not have access to the data content and because of the mix network's mixing and morphing mechanism she cannot correlate the input and output packets based on data content. But with the use of the headers, she can easily identify which message is coming from Alice.

**In and out.** Alice and Bob are in a location tracking system where they want to anonymize the information of their whereabouts. The system has regions called the mix zone that work on the principle of mix network. Once multiple agents enter the region and then come out it should be impossible to identify the agents. The system has a handle for all the agents. The handle acts as a pseudonym for the agent. Suppose Alice has the pseudonym agent12345 and Bob has the pseudonym agent12346. Now agent12345 and agent12346 enter the mix zone and come out. However, they still retain their pseudonym and therefore are trivially identified.

### 27.3. Context

You are designing a system to protect the privacy of the users. This system will maintain sender anonymity in a messaging scenario. Although this is an important application area, the context is not limited to messaging only. The context also entails other scenarios like anonymity in a location tracking system or anonymity preserving data sharing in a data publishing system.

### 27.4. Problem

Any metadata associated with the data traffic in the network reveals information about the originator of that data packet. The packet headers are used for information routing in forward and reverse direction. The information in the packet headers like IP addresses reveal private information about sender's identity and their location.

In the spatial mix zone-based location anonymity system, the agents are identified by their pseudonyms. If the pseudonym of an agent entering the mix zone and the agent exiting the mix zone remains the same, a location monitoring system can successfully correlate the outgoing agent with the incoming agent.

When private databases are shared for public use, obfuscation mechanisms like k-Anonymity [1] are applied on the database to create anonymity set. This anonymity set is compromised if information can be revealed by using meta-information associated with the data. For example, the sort order of a table in the database can be used to compromise private information. If there is a patient in a medical database whose last name starts with Z, and the patient database is sorted by last name then that person's information should appear in the last portion of the database. If the sort order is retained, the data obfuscation may be unsuccessful.

How can the meta-data associated with the data content be hidden?

The forces that need to be considered when choosing to use this pattern are as follows.

- Anonymity Service. Users might be sensitive about some of their Internet browsing behavior and do not want these behaviors to be associated with their profile. For some other browsing activities, users might be apathetic to the fact that they are being profiled. In some cases, users might be willing to be profiled over a long time so that they can get customized service experience.
- Routing. Packet headers are used for routing. Stripping off packet headers would make it difficult to route the request. Also, the response from the recipient has to be sent back to the request originator. If the header is irretrievably tampered, then the response cannot be routed back to the originator. Encrypting packet headers do not work either because, the intermediate mixes have to access the packet headers for routing. If a mix is controlled by an adversary, the plaintext header information at that node would reveal sender identity.
- Performance. The system should not have complex operations that add to the latency in a messaging system. The system should be applicable in a low-latency messaging domain like web browsing.
- Type of Adversary. A global, passive adversary monitors the data traffic in the network and do not manipulate the data content. Active adversaries may control the mix nodes and manipulate the through traffic. Mix nodes can also be controlled by passive (or semi-honest) adversaries that adhere to the mix protocol but only monitor the data content passing through the node. The adversary has access to other third-party data sources that he can use to infer information.
- User Consent. Most of the web browsing information are gathered without users' consent. The technology underlying web browsing makes it possible for web sites to collect varying amounts of personal information about each user. Although it is important that the consumers have the right to be informed about the privacy and security consequences of an online transaction before entering into one, current technology does not provide any mechanism to enforce this user right.
- Payment for Privileged Service. Users are sometimes willing to pay for privileged anonymity service for their sensitive data. The privileged service would involve lower latency in data transmission and better anonymity.

- Law Enforcement. Anonymity can be abused by malicious users and to thwart that law enforcement agencies might require that the anonymity solutions sometimes lift their anonymity cover to investigate on crime suspects.

## 27.5. Solution

Obfuscate the metadata associated with the data. For the web browsing domain, create a middleman between the request sender and the recipient that strips off identity-revealing meta-data from the packet headers. The sender submits the request to the middleman that acts as a proxy. It submits the request to the recipient on behalf of the sender but removes the values from the tags like `HTTP_USER_AGENT` and `HTTP_REFERER` from the header. For email messaging, use a remailer that strips identifying header information from outbound email messages. Hide the metadata that do not hamper message routing, or the service provided by the end server upon receiving the message.

For the location anonymity domain, strip off the pseudonym associated with an agent when he enters the mix zone. Assign a different pseudonym to the agent when he comes out of the mix zone.

For privacy preserving data sharing, scramble the sorting order of the data in the table. Remove any other meta-information in the data that can compromise the privacy.

## 27.6. Structure

## 27.7. Dynamics

## 27.8. Implementation

**Storage of State Information.** The anonymizer should keep track of the changes it has made to the sender's request header. This is needed to forward the response back to the sender. This can be stored in a table-based storage with hashed key, or a database if the anonymizer controls large amount of traffic. The anonymizer proxy should also keep the states to prevent replay attacks.

**Traffic Analysis of Anonymizing Proxy.** The ingress and egress traffic of anonymizing proxy can be monitored, and privacy can be compromised if the anonymity set is small. The anonymizer can adopt mix technologies to prevent against these attacks. Also, the sender might submit traffic to the anonymizer through a mix network or onion routing portal to achieve stronger anonymity guarantee.

**Trust Relationship with the Anonymizer.** The users should establish a trust relationship with the anonymizing proxy. A malicious proxy could in principle track its users' browsing patterns and make unscrupulous use of that information. The trust establishment can be achieved with a legal contract or can be done dynamically using explicit trust negotiation protocols.

**Bandwidth Requirement.** The anonymizing proxy has to handle hundreds of thousands of page requests. For each request, the anonymizing proxy has to fetch, process, and forward a web page from elsewhere on the net. A subscription mechanism can be introduced to create users of various privilege levels and the requests can be prioritized based on that.

**Direct Sender-recipient Link.** Many ActiveX controls require direct linkage between the sender and the recipient. For example, RealAudio goes around the proxy by establishing their own direct net connections. Recent Ajaxian applications also require direct connection between the browser and the server. The link-rewriting mechanism of anonymizer proxy cannot provide anonymity for browsing these pages with active controls.

## 27.9. Example Resolved

## 27.10. Consequences

The pattern has the following benefits.

- Privacy. The anonymizing proxy provides an alternative for privacy. It does not depend on costly cryptographic operations like a mix network. Also mix networks require wide-scale deployment, an issue that is not relevant to the architecture of anonymizing proxy. The anonymity service is offered transparently by the anonymization proxy, and the users do not have to modify their normal behavior to use the service.
- Business Incentive for running an Anonymizer. Business venture can be established to provide anonymity services like anonymizer, because it does not rely on wide-spread deployment of services. In a mix network, the mix node has to communicate with other nodes beyond the organizational boundary. The anonymizer can be deployed independently and the success of the it depends on the reputation of the authority running the anonymity service. Moreover, the anonymizer can provide privileged service based on subscription. A client using free service would experience higher latency than the paid service.

The pattern has the following liabilities.

- Performance Overhead. Heavy traffic can throttle beyond the bandwidth limit of the anonymizer creating a DoS scenario. The storage and maintenance of meta-information of anonymized packets and packet processing cost has severe performance overhead.
- Single Point of Failure. The anonymizing proxy is a single point of failure. For a mix network, a passive adversary has to monitor different parts of the network. On the other hand, for an anonymizing proxy monitoring the ingress and egress paths is sufficient for the attacker.
- Forced Compromise of Privacy. An anonymizing proxy may keep track of the obfuscations it is making on incoming data to generate outgoing data traffic. This is especially necessary because the response traffic has to be routed in the reverse direction. Maintainer of an anonymizing proxy can be forced by law enforcement authorities to divulge this information, thereby undermining the anonymity of the

proxy users. This can also lead to extortion from influential organizations. One of the first remailers built on this concept (the Penet remailer, developed in 1993) came under attack several times from different organizations. In 1995, the Church of Scientology filed a lawsuit against Johan Helsingius (the creator and maintainer of the Penet remailer) to disclose the identity of an anonymous user, who posted a stolen file anonymously in the alt.religion.scientology newsgroup. The file was stolen from the Church's internal server. The Church's initial claim was to reveal the identity of all the users of the remailer (about 300,000 in that time), but in the end they settled with the disclosure of the person responsible for the post. The identity of the anonymous user, who was posting under the pseudonym "-AB-" and the anonymous ID an144108@anon.penet.fi, was revealed to be Tom Rummelhart, a system administrator of the Church of Scientology's INCOMM computer system.

Johan Helsingius was also contacted by the government of Singapore as part of an effort to discover who was posting messages criticizing the government in the newsgroup soc.culture.singapore. This time Johan did not have to compromise the identity of the user because the Finnish law did not rule the posting as a crime.

Then in September 1996, Church of Scientology sued Grady Ward [2] under the suspicion that he posted secret files under the posting title "Scamizdat" in the Penet remailer and forced Johan Helsingius to disclose the identity of two users, an498608@anon.penet.fi and an545430@anon.penet.fi, posting under the handle "DarkDemonStalker". Johan decided to close the remailer in September 1996. The stories of these attacks on the Penet remailer have been written in many newspaper and online articles [3, 4, 5, 6]. Ironically, the Church extorted the information of the anonymous post, but it turned out to be anonymized by another anonymous remailer, the alpha.c2.org nymserver. alpha. c2.org was a more advanced and more secure remailer that obfuscated the mapping of the input and the output, and hence the Church could not get the conviction they were after.

## 27.11. Known Uses

The Penet remailer (anon.penet.fi) [7] was a pseudonymous remailer operated by Johan Helsingius of Finland from 1993 to 1996. The concept of this remailer is to provide a portal that stores pseudonyms for users. The users send messages hiding behind the pseudonym. By stripping the user's name and assigning a pseudonym, the system provides sender anonymity through pseudonymity. Moreover, recipient anonymity can be achieved if the recipient of the mail is also a user behind a pseudonym. Because the users always use one pseudonym, it has the advantage that the users can create a reputation by using the pseudonym for a long time. But repeated use of the pseudonym means that privacy can be weakened by long term salvage of context information from a user's correspondence.

The Anonymizer provides a technological means for preserving a user's privacy when surfing the web. A third-party web site (<http://www.anonymizer.com>) is set up to act as a middleman between the sender and the recipient. When the client wants to visit a web site, say the Google web site ([www.google.com](http://www.google.com)), he does not send the request directly to the Google server. Instead, it directs the request through the anonymizer proxy by using the URL <http://www.anonymizer.com:8080/www.google.com>. The Anonymizer then connects to

google.com without revealing any information about the user who requested the information, and forwards the information received from Google to the user.

The first version of the Anonymizer was based on the public-domain CERN proxy server, but with several modifications to preserve anonymity:

- It does not forward the source IP address of the end-user.
- It eliminates revealing information about the user's machine configuration from the "User-Agent" MIME header, user's name from the "From" MIME header, and previously visited site name from the "Referer" MIME header.
- It does not forward the user's email address to serve as a password for FTP transactions.
- It filters out Java applets and JavaScript scripts which may compromise anonymity.
- It filters out all "magic cookies" which may compromise anonymity.
- It gives positive feedback to the user by displaying an Anonymizer header on the page and adding the word "[Anonymized]" to the page's title.

The Anonymizer provides an easy-to-use interface which allows users to bypass the configuration procedure normally associated with using a proxy. Users access the service simply with extended URLs, such as <http://www.anonymizer.com:8080/www.google.com/>. The interface is flexible, and the users can freely switch to their regular browsing behavior when anonymity is not required.

There are various other anonymity providing services that are built on the principle of anonymizer, e.g. iProxy and the Lucent Personalized Web Assistant (LPWA). LPWA does not offer the Anonymizer's page-rewriting mechanism which enables users to easily change between anonymized and non-anonymized browsing. However, it does provide an additional feature, support for anonymous authentication and registration at web sites which provide personalized services.

Mix zone [11, 12] is the same concept of mix networks taken into the domain of location anonymity. Agents are identified by pseudonyms in the mix zone. When the pseudonymous agent enters into the mix zone, his pseudonym is changed, so that once he comes out of the zone, his identity cannot be correlated with that of the entering agent. The mix zone concept was added with the Active Bat [13] system to provide location anonymity.

The principle of k-Anonymity [1] was introduced by Latanya Sweeney for publishing of secret data. The sensitive information in a dataset is obfuscated by replacing them with a more general information. The sort order is also altered to obfuscate meta-information.

## 27.12. See Also

## 27.13. References

[1] Sweeney, L. (2002). k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05), 557-570.

[2] Newman, R. (1996, July 24). The Church of Scientology vs. Grady Ward. XS4ALL.  
<http://www.xs4all.nl/~kspaink/cos/rnewman/grady/home.html>

- [3] Post, D. G. (1996). The first Internet war-The state of nature and the first Internet war: Scientology, its critics, anarchy, and law in Cyberspace. *Reason Magazine*.
- [4] Newman, R. (1997, Mar 23). The Church of Scientology vs. anon.penetr.fi - Julf Helsingius voluntarily closes his remailer after Finnish court orders him to turn over a user name to Scientology. XS4ALL. <http://www.xs4all.nl/~kspaink/cos/rnewman/anon/penet.html>
- [5] EFF. (1996, Sept 23). Johan Helsingius gets injunction in Scientology case: Privacy protection of anonymous messages still unclear. EFF. [http://www.eff.org/Privacy/Anonymity/960923\\_penet\\_injunction.announce](http://www.eff.org/Privacy/Anonymity/960923_penet_injunction.announce)
- [6] Helsingius, J. (1996, Aug 30). Johan Helsingius closes his Internet remailer. Cyberpass. [http://www.cyberpass.net/security/penet\\_press-release.html](http://www.cyberpass.net/security/penet_press-release.html)
- [7] Helsingius, J. J. (1995). The anon.penetr.fi anonymous server.
- [11] Beresford, A. R., & Stajano, F. (2004, March). Mix zones: User privacy in location-aware services. In *IEEE Annual conference on pervasive computing and communications workshops, 2004. Proceedings of the Second* (pp. 127-131). IEEE.
- [12] Beresford, A. R., & Stajano, F. (2003). Location privacy in pervasive computing. *IEEE Pervasive computing*, 2(1), 46-55.
- [13] Ward, A., Jones, A., & Hopper, A. (1997). A new location technique for the active office. *IEEE Personal communications*, 4(5), 42-47.

## **27.14. Source**

- Hafiz, M. (2006, October). A collection of privacy design patterns. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-13).

# 28. Layered Encryption

## 28.1. Intent

Use a sender-initiated packet routing scheme and encrypt the data packets in multiple layers so that the intermediaries only have access to a particular layer and use that information to route the packet to the next hop.

## 28.2. Example

Ghost in the Machine. Alice, Bob, and Carol are using a mix-based system to communicate over the Internet. Alice sends her data through a node in the network where it gets mixed with the data coming from other sources (e.g. Bob, Carol, etc.). Figure 55 shows that Mallory is an active semi-honest adversary who controls the mix node, i.e. Mallory obeys the mix protocol to appear as an honest mix, but she tries to learn information by looking at the packets that are routed through the mix.

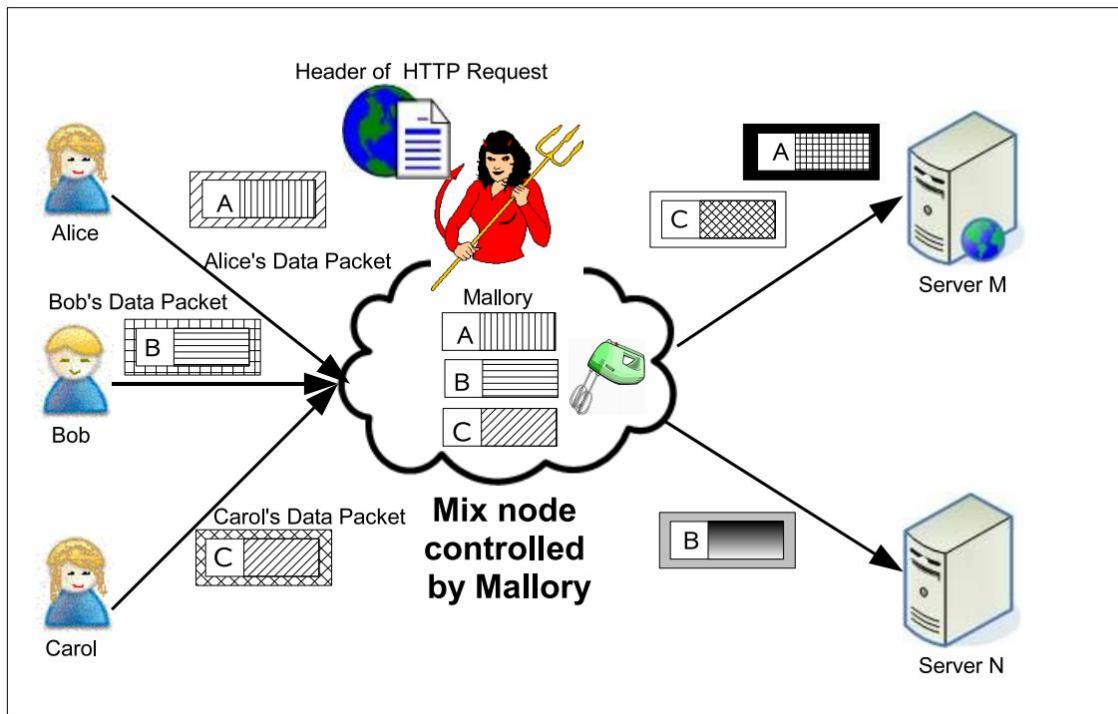


Figure 55: An active attacker controlling a mix node

The body of Alice's packet is encrypted with a symmetric key between Alice and the recipient, so that the intermediaries cannot access the content. According to the mix protocol, when Alice's packet is in transit between Alice and the mix node, it is encrypted with a shared symmetric key between her and the mix. Passive observers monitoring the link between Alice and the mix node cannot access the header of the packet. The mix network decrypts the packet, reads the header, finds the next hop, and routes the packet to the next hop after encrypting it with a shared key between the mix node and the next hop. Again, passive adversaries monitoring the egress packets of the mix node cannot access the packet header.

Moreover, an adversary monitoring the ingress and egress links of the mix network cannot correlate the incoming and outgoing packets because of the mix protocol.

The problem arises because the mix node is controlled by an active adversary Mallory. Mallory shares the encryption key with Alice and the next hop, and accesses Alice's packet header to determine the routing option. This compromises the sender anonymity of Alice. Also, from the header of Alice's packet, Mallory can determine the ultimate recipient, and therefore can compromise sender and recipient unlinkability.

### 28.3. Context

You are designing a mix-based system to protect the privacy of the users. You want to have sender anonymity and sender and receiver unlinkability for the communicating parties. The system can be a mix-based filter on the Internet that is used for email messaging or web browsing.

### 28.4. Problem

In the mix protocol, the mix nodes share symmetric keys between themselves. The mix decrypts and then re-encrypts the packets flowing through the node. This protects against a passive adversary observing the network traffic but is insufficient against an active adversary controlling a mix node.

The mix node accesses the packet headers in order to identify the next hop. The header contains the ultimate destination, and the choice of next hop is determined by that. A malicious attacker controlling the mix node can follow the mix protocol, and yet profile the behavior of a message sender, because of the header in plaintext available to him.

How can the mix network be made secure against an active adversary?

The forces that need to be considered when choosing to use this pattern are as follows.

- Type of Adversary. Privacy can be compromised by different types of adversaries. A passive adversary only observes the network traffic, but does not manipulate the data packets. An active adversary manipulates the data packets or compromises and controls a mix node. After controlling the mix network, an adversary can act semi-honestly, i.e. he continues to act like an honest node by following the mix protocol but at the same time tries to get information from the packets flowing through the mix node. Adversaries can also collude to undermine the anonymity of the message sender.
- Routing Mechanism. The packet header contains information that is essential for the routing decision. A distributed routing mechanism would delegate this decision to the intermediary nodes. Contrarily, in a centralized routing mechanism, the sender determines the route that the packet will take and adds that information in the packet header.
- Cost of Encryption. The cost of decryption and encryption can become an overhead. The mix network should be usable for a low latency messaging requirement like web browsing.
- Key Establishment. The network follows protocols for dynamic negotiation and establishment of keys. A symmetric key share can be established by using a PKI

- scheme, but it assumes the presence of a global PKI framework. Key sharing schemes with low infrastructure requirements can be used, e.g. Diff e-Hellman key exchange [1].
- Application Independence. The mechanism for achieving sender anonymity should be application independent. It should be applicable for low latency messaging domain like web browsing, and latency independent messaging domain like email messaging.

## 28.5. Solution

The sending client is responsible for establishing the path between the sender and the recipient. The neighboring nodes in the circuit share symmetric keys between themselves. The packet is then encrypted in multiple layers (like the onion skin). The innermost layer is encrypted with the symmetric key used in the last hop before the server, the next layer is encrypted with the symmetric key used in the preceding hop and so on.

Thus, the sending client has to construct a chain of nodes, and when the message is in transit through these nodes, each node strips off a layer using its key share, finds the identity of the next hop within the decrypted bundle, and forwards it to that node. For example, for a remailer  $E_i$ , with  $R_i$  as its public key,  $A_i$  as its address, and  $B$  as the destination address, a three-link route between Alice and Bob looks like

Alice –[ $E_1(A_2, E_2(A_3, E_3(B, M)))$ ]->

$R_1$  –[ $E_2(A_3, E_3(B, M))$ ]->

$R_2$  –[ $E_3(B, M)$ ]->

$R_3$  –[ $M$ ]-> Bob

Each remailer is able to decrypt the bundle it receives, but it cannot itself look more than one link ahead, let alone determine the final destination. Moreover, after the first link, the sender's identity has been removed. The first remailer  $R_1$  is connected with the sender but when it receives the message, it has no way to determine whether its previous node is the sender or just another mix node in the remailer chain.

## 28.6. Structure

## 28.7. Dynamics

## 28.8. Implementation

The implementation issues described in the MORPHED REPRESENTATION pattern also applies here. Additional issues are as follows.

**Service Composition.** Layered encryption can be used in conjunction with other services. Layered encryption can be used for the path between a request sender and the anonymizing

proxy (e.g. Anonymizer, LPWA etc.) and the proxy then submits the request on the sender's behalf.

**Layered Encryption Overhead.** The main overhead of layered encryption is the path setup cost. Typically, it is much less than one second, and it appears to be no more noticeable than other delays associated with normal web connection setup on the Internet. Computationally expensive public key operation is only used during the connection setup phase. By using dedicated hardware accelerators on the routers, the burden of public key operations can be relaxed.

## 28.9. Example Resolved

## 28.10. Consequences

The pattern has the following benefits.

- Sender-determined Routing and Privacy. In a distributed routing protocol, the intermediaries determine the path of the packet on its route, but for this the intermediaries need to have access of sender and recipient information. Sender and recipient anonymity is achieved by using this pattern because here the routing decision is taken by the sender only. The sender initiates a path setup protocol to create the route. This can be done in offline (i.e. when the system is idle) to reduce performance overhead.
- Application Independence. Layered Encryption can be used with proxy-aware applications, as well as several non-proxy-aware applications. Layered encryption supports various protocols, e.g. HTTP, FTP, SMTP, rlogin, telnet, finger, whois and raw sockets. Proxies can be used with NNTP, Socks 5, DNS, NFS, IRC, HTTPS, SSH and Virtual Private Networks (VPN).

The pattern has the following liabilities.

- Data Integrity. The layered encryption technology does not perform integrity checking on the data. Any node in the path of layered encryption can change the content of data cells. However, if the adversary controlling a mix node alters the data content of the packet, the subsequent node will figure out the discrepancy. This mix node sends the information about the mal-formed data packet in the backward path, and eventually the sender finds out about the integrity violation. The sender can then initiate a new path and send the message along that path.
- Path Setup Overhead. The sender has to create the complete route from the sender to the recipient and this setup cost is significant. The layered encryption systems balance this by creating the paths offline.

## 28.11. Known Uses

Onion Routing [2, 3] systems are based on mixes, but they leverage the idea of mixes and add layered encryption. Onion Routing systems have better latency values than mix networks and therefore are more applicable in a web browsing scenario. Private web browsing systems for

peer-to-peer communication like Morphmix [4], Tarzan [5] etc. follow layered encryption mechanism.

## 28.12. See Also

Layered Encryption follows the Morphed Representation pattern by performing cryptographic operations at each nodes in the path.

## 28.13. References

- [1] Diffie, W., & Hellman, M. (1976). New directions in cryptography. *IEEE transactions on Information Theory*, 22(6), 644-654.
- [2] Goldschlag, D. M., Reed, M. G., & Syverson, P. F. (1996, May). Hiding routing information. In *International workshop on information hiding* (pp. 137-150). Springer, Berlin, Heidelberg.
- [3] Syverson, P. F., Goldschlag, D. M., & Reed, M. G. (1997, May). Anonymous connections and onion routing. In *Proceedings. 1997 IEEE Symposium on Security and Privacy (Cat. No. 97CB36097)* (pp. 44-54). IEEE.
- [4] Rennhard, M., & Plattner, B. (2002, November). Introducing morphmix: Peer-to-peer based anonymous internet usage with collusion detection. In *Proceedings of the 2002 ACM Workshop on Privacy in the Electronic Society* (pp. 91-102).
- [5] Freedman, M. J., & Morris, R. (2002, November). Tarzan: A peer-to-peer anonymizing network layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security* (pp. 193-206).

## 28.14. Source

Hafiz, M. (2006, October). A collection of privacy design patterns. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-13).

# **29. Informed Consent for Web-based Transactions**

## **29.1. Intent**

This pattern describes how websites can inform users whenever they intend to collect and use an individual's personal information.

Although this pattern includes elements of a user interface design, it speaks more deeply than the interaction between a consumer and website and the sort of infrastructure that is needed to support the informed consent interaction model. A user interface would define the surface of the interface, such as how the interface should look and how content should be phrased. The user interface is just a part (but alone not sufficient) of a properly functioning consent form. In addition, well designed user interfaces are also an important part of fostering a website's credibility and may affect the extent to which a user chooses to disclose private information. However, that is beyond the scope of this pattern.

This privacy pattern focuses on user consent and information relegation. The principles described here are meant to help designers determine their design goals when communicating with users or collecting their information. These goals do not necessarily have to be highly technical or full of legalese. For example, they may simply include text next to e-mail subscriptions such as, "we will not sell or share your email under any circumstances." This pattern will, therefore, also help users understand the consequences of disclosing identifiable information once completing a transaction.

## **29.2. Example**

## **29.3. Context**

Web developers and website interaction designers are creating a website that will collect personal information from users for a survey, registration, or other purpose. The organization may be motivated to protect the privacy of its users either because of legislative requirements such as HIPAA or COPPA or because of consumer market pressures.

## **29.4. Problem**

To facilitate transactions, websites often use cookies to track users and web forms to collect personal information. However, users are often resistant to disclosing personal information because they are uncertain if it will be used without their consent or against their interests. The problem is: How can website designers communicate their intended uses for the information they collect from users?

As website owners and designers, you must balance the following forces:

- You realize users want to visit your website and participate in its services without fear of unnecessarily being tracked and identified
- You realize users want to maintain as much control over their personal information as possible
- You have the right to request and use information and to refuse service to users who don't provide their information (except as restricted by law)
- You are able to provide richer and more customized services when you know who your users are
- You know you must protect your users' privacy, but you want to do so while minimizing your cost

## 29.5. Solution

To the extent possible given the limits imposed by web technology, provide the user with the following six elements of informed consent: disclosure, agreement, comprehension, voluntariness, competence, and minimal distraction.

**Disclosure:** If you are either implicitly or explicitly collecting identifiable data from a user, fully disclose how that data will be used and for how long. Also, clearly inform users of the practical risks and benefits of participating in the online interaction such as having the information sent to 3rd parties for marketing or research purposes. Place disclosure information both on pages that are easily accessible throughout the website and particularly at the point of data collection as this is where it is most relevant. Providing easy access to this privacy policy will allow the user to form a decision before committing to the transaction. Where important fields of data are requested, provide clear indication to the user as to why the data is required and how they will be used.

**Agreement:** Provide the user with the ability to opt-out of the agreement at any time. This would allow them to cancel any marketing or incentive solicitations and prevent further information from being used by 3rd parties. If an opt-in feature is used (for example, for extra marketing incentives), setting the default value of "yes" or "checked" will typically produce greater positive results as people who are rushed or accept all default values will not change the options. However, this may reduce the voluntariness of the agreement.

**Comprehension:** To the extent possible, ensure that the user understands how the information that is being requested of them will be used. That is, confirm that the user realizes the liabilities and benefits. E.g., does the user know what a "cookie" is? Does the user understand when or if the data will be deleted? Who can have access to the date and for what purposes? Users may see the text of a privacy policy, but have they read it, and do they know what it means?

**Voluntariness:** Ensure as best you can, that the information is being offered without coercion or external influence by:

- Not manipulating the options so as to suggest a certain course of action. E.g. suggesting that users can only enjoy special services if they register on the website.
- Not manipulating the options so as to mask useful or necessary information (contributing to information asymmetry). E.g. hiding the privacy policy.

- Offering alternate means of fulfilling the service to users if they feel uncomfortable with the current method. This may be an online chat service, phone number to access a live customer service representative, fax service or standard postal mail.

**Competence:** To the extent possible ensure that the user from whom you are soliciting information is adequately competent to provide that information. For example, ensure that the user is of legal age. A commonly used (but not foolproof) practice is asking the user to submit their birth date during the registration process.

**Minimal Distraction:** Provide each of these functions without significant diversion from the service that you are providing. Not doing so would both cause frustration on the part of the user and likely result in fewer transactions. One method for accomplishing this is to open a separate browser window that displays the relevant information, such as a clearly formatted privacy policy. Using a separate browser window allows the user to continue with the transaction (e.g. filling in a web form) without having to be directed away from the form, then back, forcing them to re-enter data.

## **29.6. Structure**

## **29.7. Dynamics**

## **29.8. Implementation**

## **29.9. Example Resolved**

## **29.10. Consequences**

This pattern offers the following benefits:

- Helps to reduce information asymmetry between the user (data owner) and the website (data controller).
- Empowers users to make informed decision that do not conflict with their tolerance for private information disclosure.
- Provides a basis for trust between the consumer and website owner by establishing an expectation of practice by the website. Consider the risk of lost trust for ecommerce, medical and financial companies such as eBay, Amazon, Bank of America, ehealthinsurance.com, etc..
- This pattern can be applied to many other systems that interact with the user and external systems such as email and location aware devices (e.g. cellphones, PDAs).

This pattern suffers from the following liabilities:

- This pattern cannot provide any assurance that a website will comply with the informed consent model.
- Privacy policies are generally known to be confusing for the user to read and fully understand.
- The website may not wish to disclose their ability to track users without their knowledge.
- The website may not have the infrastructure to offer and support each of the solution elements for every user. For example, the ability for users to opt-out of the agreement.
- If the distraction due to implementing this pattern is sufficiently great, the user may simply cancel the transaction altogether [1].
- Information provided to gain consent is necessarily a) limited and b) manipulated by the site to obtain consent – this implies that the actual consequences of the revelation of personal information may remain unknown to the user.
- Smaller web-enabled devices such as cell phones and PDAs may not be able to support Minimal Distraction as easily as full featured web browsers.

## 29.11. Known Uses

This pattern is used in whole or part by many ecommerce, financial and health websites such as Yahoo!, Intuit, Google, and ehealthinsurance.com. For instance, the Yahoo! Email registration form, as shown in Figure 56, provides mouse-over dialogue boxes that inform the user about why certain fields (e.g. birthdates) must be filled out accurately.

The screenshot shows a registration form with several input fields. A tooltip is displayed over the 'Birthdate' field, which contains placeholder text: 'Four characters or more. Make sure your answer is memorable for you but hard for others to guess!' The tooltip also includes a note: 'Please provide an accurate birthdate for your own protection. We ask your birthdate to verify your account if you ever forget your Yahoo! ID or password. (Yahoo! will never request your password or ID in an unsolicited email or phone call.)'

Figure 56: Yahoo! Registration Form

The Intuit registration form shown in Figure 57 discloses how they use the information and offers the option of opting-out of correspondences.

Provide a valid e-mail you can always access. We'll use it in case you need to reset your password or retrieve your user ID.

<b>E-mail Address</b>	<input type="text"/>
<b>Confirm E-mail</b>	<input type="text"/>

We will not rent, sell or share your personal information with outside companies for their promotional use. The information you provide to Intuit will be used to send you messages regarding your tax return account. Occasionally we may contact you with special offers that may interest you. If you prefer, you can tell us how you would like Intuit to [contact you](#).

Figure 57: Intuit Registration Form

This pattern is consistent with the Fair Information Practices (FIP) recognized by many website policies and privacy laws and is part of the standard practice for patient care established by the American Medical Association, and U.S. Office for Human Research Protections (OHRP).

Platform for Privacy Preferences (P3P) [2] is a computer readable (and searchable) method used by websites to define and publish their policies for collecting and using information. The policies can be automatically read by user-agents to indicate whether or not the website's policies match a user's privacy preferences and helps provide both Disclosure and Minimal Distraction.

This pattern is also used by software developers of desktop applications who request that users provide personal information, or as a means for the application to collect usage data from the user. This information is typically captured during installation or while using the application and transmitted online to the software vendor.

## **29.12. See Also**

## **29.13. References**

[1] Friedman, B., Felten, E., & Millett, L. I. (2000). Informed consent online: A conceptual model and design principles. *University of Washington Computer Science & Engineering Technical Report 00-12-2*, 8.

[2] Cranor, L. (2002). *Web privacy with P3P*. " O'Reilly Media, Inc.".

## **29.14. Source**

Romanosky, S., Acquisti, A., Hong, J., Cranor, L. F., & Friedman, B. (2006, October). Privacy patterns for online interactions. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-9).

# 30. Session-Based Attribute-Based Authorization

## 30.1. Intent

Allow access to resources based on the attributes of the subjects and the properties of the objects but limit the rights that can be applied at a given time based on the context defined by the access session.

## 30.2. Example

Meili is a teenager who likes movies and subscribes to several movie services through the Internet. She logs in a central portal where she can reach a variety of movies. Sometimes she gets movies that she finds offensive or inappropriate (pornographic, racist, plain stupid). She doesn't have much time to read details about the movies in advance and some of them don't even have good descriptions so reading about the movies is not a good approach. She would like some kind of filter according to her characteristics and her preferences. Also the portal may be breaking the law in making available to her some of these movies.

## 30.3. Context

Dynamic systems supporting a large set of objects and subjects in which the structure of the subjects changes rapidly, such as webbased information systems, e-government, and e-business portals. In this environment there is the need to control access to computing resources and the subjects may not be pre-registered. We want to give access to resources based on characteristics of the subjects such as groups to which they belong, company for which they work, biological characteristics such as age or sex, or on characteristics of the objects, such as type of object, filtering rules, or payment requirements.

## 30.4. Problem

As indicated access may depend on the age or other attributes of a user. In this case, privilege assignments to the user cannot be done statically by a security administrator but automatically by the system based on the value of some of the attributes, e.g. "DateOfBirth". As the user gets older or changes functions his authorization state changes automatically. Access rights might even depend on an external attribute, such as "physical location" of a user in a mobile environment. In this case the authorization state changes automatically when the user moves around. At the object's side, metadata such as the scope of a document, or the MPAA rating of a movie are examples of proper ties. All these constraints can be applied through predicates in the rules [1], but it is difficult to have a variety of prepackaged rules for the typical cases.

The solution is constrained by the following forces:

- We need to limit the rights of subjects that are in a variety of groups or roles or have special characteristics. Unrestricted access might allow policy or law violations.
- This control should not imply an extra burden for the security administrator or security vulnerabilities may appear through administration errors.

- This control should not imply a significant performance overhead, or the system may not be practical to use.
- The environment is very dynamic, and changes should be easy to make. Otherwise, the users will get annoyed and leave the system.

## 30.5. Solution

Access rights are based on the comparison of values of selected attributes of subjects and properties of objects (so called subject and object descriptors). In this pattern descriptors are a construct to somehow “group” objects and subjects dynamically, not explicitly by an administrator but implicitly by their attribute or property values. This grouping may result in unpredictable sets of rights that may violate security policies. A session delimits the rights that can be applied at a given moment; that is, the subject attributes define a context for access rights.

## 30.6. Structure

Figure 58 shows the class diagram for the solution. A Subject Descriptor is formed by applying Qualifiers (>, +, ...) to Attribute Values to define constraints such as ‘age > 15’. A Session selects some specific attribute values as execution context that defines the Subject descriptor at this moment. Similarly, objects are defined based on the values of selected attributes.

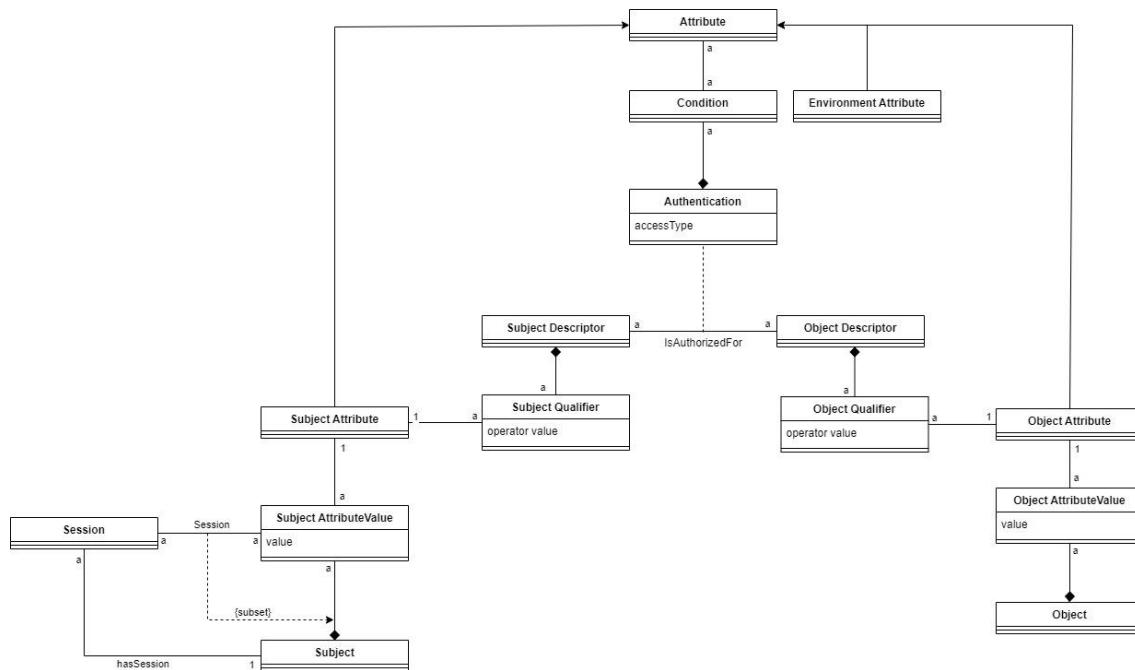


Figure 58: Class model for the Session-Based ABAC pattern

## 30.7. Dynamics

## **30.8. Implementation**

1. Select an appropriate package to convey the subject's credentials including attributes. Examples would be attribute certificates [2][3] or Kerberos tickets.
2. Select an implementation to express the object's attributes. Candidates could be standards on meta-data resource discovery, such as the Dublin Core Metadata Initiative [DCM].
3. Define an enforcement mechanism for the rights defined in contexts. See for example [4].

## **30.9. Example Resolved**

The portal implemented an ABAC model. Now when Meili opens a session she is given access to contexts with sets of preselected movies according to her preferences and restricted according to legal aspects and to the services she has paid for.

## **30.10. Consequences**

The advantages of this pattern include:

- The rights of subjects that belong to a variety of groups, roles, or have special attributes can be limited by restricting them to specific contexts selected by sessions.
- This control does not imply an extra burden for the security administrator because the contexts can be defined by application designers according to application policies.
- This control does not imply a significant performance overhead because changing from one context to another just means changing a set of rights.
- Changes in access restrictions can be easily accommodated by defining new contexts or deleting existing contexts.

Possible disadvantages are:

- Higher complexity. Although the contexts are defined by others, it is hard for administrators to know who has access to what.
- There might still be some performance overhead if we need to switch often between contexts.

## **30.11. Known Uses**

Session-based ABAC is implemented as an alternative to RBAC in the security module CSAP [5] of the Webocrat system. A similar pattern is also used in the authorization system of the .NET component framework [6] and in AAIs (authentication and authorization infrastructures), such as Permis [7] and Shibboleth [8].

The XML standard XACML [9][10] uses attributes of subjects and objects for the specification of access control policies. As shown in the UCONABC [11], ABAC may also have potential for digital rights management.

## **30.12. See Also**

## **30.13. References**

- [1] Fernandez, E. B., Summers, R. C., & Wood, C. (1981). *Database security and integrity*. Addison-Wesley Longman Publishing Co., Inc..
- [2] Morrison, P., & Fernandez, E. B. (2006, October). The credentials pattern. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-4).
- [3] Oppliger, R., Pernul, G., & Strauss, C. (2000). Using attribute certificates to implement role-based authorization and access controls. *Sicherheit in Informationssystemen (SIS 2000)*, 169-184.
- [4] Corrad, A., Montanari, R., & Tibaldi, D. (2004, September). Context-based access control management in ubiquitous environments. In *Third IEEE International Symposium on Network Computing and Applications, 2004.(NCA 2004). Proceedings*. (pp. 253-260). IEEE.
- [5] Dridi, F., Fischer, M., & Pernul, G. (2003, May). CSAP—An Adaptable Security Module for the E-Government System Webocrat. In *IFIP International Information Security Conference* (pp. 301-312). Springer, Boston, MA.
- [6] LaMacchia, B. A., Lange, S., Lyons, M., Martin, R., & Price, K. T. (2002). *.NET framework security* (p. 616). Reading: Addison-Wesley.
- [7] Chadwick, D. W., & Otenko, A. (2003). The PERMIS X. 509 role based privilege management infrastructure. *Future generation computer systems*, 19(2), 277-289.
- [8] InCommon. (n.d.). Shibboleth. <https://www.incommon.org/software/shibboleth/>
- [9] Delessy, N., Fernandez, E. B., & Sorgente, T. (2005, September). Patterns for the extensible access control markup language. In *Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005)* (pp. 7-10).
- [10] Anderson, A., Nadalin, A., Parducci, B., Engovatov, D., Lockhart, H., Kudo, M., ... & Moses, T. (2003). extensible access control markup language (xacml) version 1.1. OASIS.
- [11] Park, J., & Sandhu, R. (2004). The UCONABC usage control model. *ACM transactions on information and system security (TISSEC)*, 7(1), 128-174.

## **30.14. Source**

Fernandez, E. B., & Pernul, G. (2006, October). Patterns for session-based access control. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-10).

# 31. Privacy-Aware Network Client

## 31.1. Intent

This pattern provides a way to make a user of a network site aware of the privacy policies followed by that site. It introduces the concept of a Privacy Proxy to enhance the user's comprehension of any privacy-related concerns. Even though the current uses of this pattern are constrained to the web browsing domain, it can have a more general use.

## 31.2. Example

The users in our company connect to websites for a variety of purposes, including product search, component purchasing, and looking for general information. Every interaction may require the user to provide some information and our users may unwittingly provide too much information. This extra information could be used later to steal their identities or to send spam to them. We would like our users to be aware of what information the sites really need to collect and to learn to avoid sites that require unnecessary information and do not guarantee privacy.

## 31.3. Context

Users interacting with Internet sites that sell goods or provide services, where to have access, one needs to provide some personal information.

## 31.4. Problem

A main concern about privacy is the awareness level of the user. A network server can use a standard such as P3P to conveniently publish privacy policies [1], which describe how each connecting user's private data is gathered and utilized. However, how can we ensure that a user connecting through a network client will be made aware of these policies prior to divulging this data?

The possible solution is constrained by the following forces:

- Privacy policies must be displayed to the user in a form that can be clearly understood.
- The user must be able to select what information can be gathered and used through a simple, easy-to-use interface.
- Privacy policies may change, and the user must be able to see the latest ones; otherwise she might follow obsolete policies that may compromise her privacy.

## 31.5. Solution

Define a privacy proxy that will be able to understand the machine-readable policies made available by the server and translate them to easy-to-use human-readable form for the user.

### 31.6. Structure

Figure 59 shows a class diagram for the relationships between the user, the server, and the proxy. Each server can publish many policies and each user can be made aware of many policies at a time through the proxy.

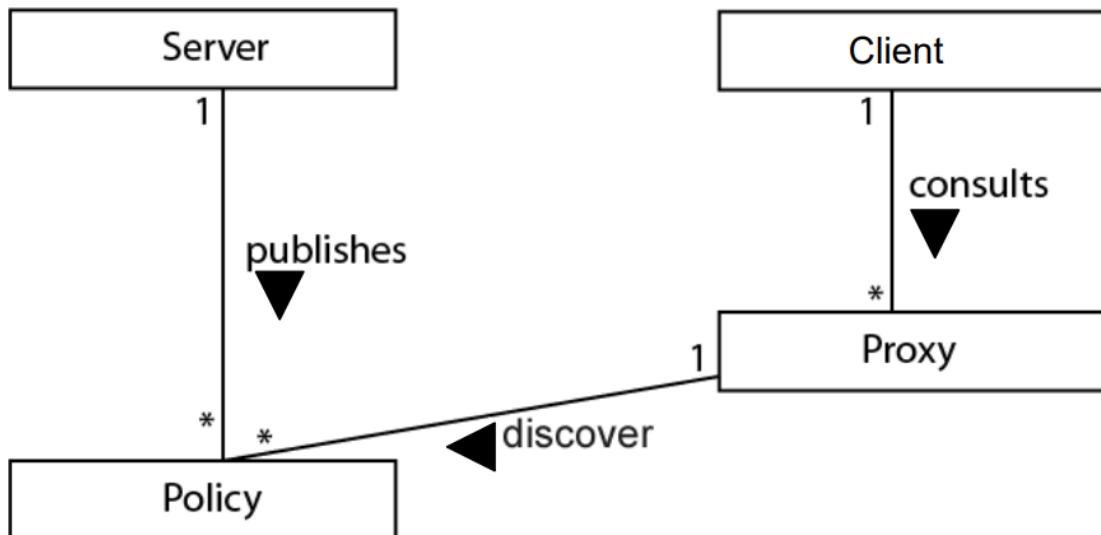


Figure 59: The Privacy-aware Network Client pattern

### 31.7. Dynamics

In Figure 60, a user wishes to access some information or interact with files on the server, which publishes its privacy, Policy. The access occurs in the following sequence:

- The User interacts with the Server through a network Client.
- The Client consults the Proxy for privacy policies.
- The Proxy discovers the correct Policy (or Policies) made available by the Server, for the information or files in question.
- The Proxy displays a user-friendly screen to the User requesting approval of the Policy, prior to allowing access to the information or permitting the interaction.
- The User decides after reviewing the Policy.

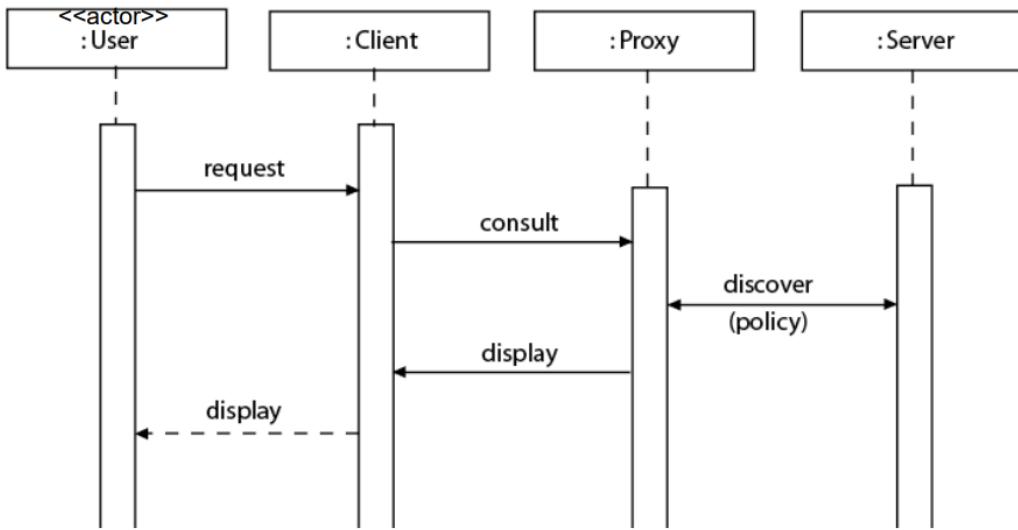


Figure 60: Sequence diagram for performing an interaction through a privacy-aware client.

## 31.8. Implementation

- Design and implement a proxy able to parse and interpret privacy policies written in some standard language. This proxy could be built as a specialized version of the Proxy pattern [2]. The proxy could be able to interpret several privacy languages or just one of them. Successful use of the pattern requires that the proxy can understand the server's privacy language.
- Design and implement a secure communication channel between network clients and their proxies. This is necessary to avoid interception of the user choices by malicious users.

## 31.9. Example Resolved

With the use of this pattern our users have now a clear view of the privacy policies of the sites they visit. Unnecessary information is not provided anymore, and they know what sites to avoid.

## 31.10. Consequences

The Privacy-Aware Network Client Pattern has the following advantages:

- The User can always be conveniently aware of the privacy policies for a specific interaction, allowing a better-informed decision prior to releasing private information.
- Though it has been used only for web-related activities, it is an appropriate pattern for general use, such as database access that could potentially deal with private information.
- Changes in privacy policies of the server will automatically be detected through the Proxy.

The Privacy-Aware Network Client Pattern has the following liabilities:

- Extra overhead in network connectivity since every access to a privacy-sensitive area needs a separate secure connection for the Proxy. This can potentially be reduced through the use of a cache.
- The pattern's concern is with the connection to the Server and the network connectivity issues only. The privacy-related constraints need to be stored locally in the Client's operating environment. Any knowledgeable attack to that machine could potentially compromise privacy.
- If the Server administrators can show (based on the user interactions) that a Privacy-Aware client has been used for a specific access, then any claims of privacy breaches can be directly blamed on the client.
- It requires that all sites use one or a small set of privacy languages.

### **31.11. Known Uses**

JRC P3P Proxy Version 2.0 i.

- From [1]: “The JRC P3P Proxy Version 2.0 is a P3P user agent, which acts as an intermediary agent (the middleman) that controls access to remote web servers dependent upon the privacy preferences a User specifies.”

Mozilla P3P Privacy Policy Viewer

- Version 7 of the Mozilla web browser has an extension called Privacy Policy Viewer, which implements a P3P reader and displays privacy policies for each site in human-readable format. Figure 61 shows its interactions. The new Privacy Policy Viewer lets a user easily locate and view the privacy policies of P3P-compliant sites.

AT&T Privacy Bird

- AT&T’s Privacy Bird implements a complete Proxy for web browsing which displays warnings when a website gathers private information. Note that the user’s response may have been previously determined and saved in a local software profile. The AT&T Privacy Bird lets you see what’s really going on at Web sites. The bird icon alerts you about Web site privacy policies with a visual symbol and optional sounds.

Internet Explorer 6 for Windows XP (cookie privacy)

- Internet Explorer is a partial implementation of this pattern. It protects only cookies, and its policy display capabilities are minimal, only supporting reading of P3P policies.
- It allows the user control over cookie privacy, however. After reviewing the P3P privacy policy, you can specify how you want Internet Explorer to handle cookies from the selected Web site. If you want Internet Explorer to determine whether or not to allow this Web site to save cookies on your computer by comparing the privacy policy with your privacy settings, select Use my privacy settings. If you want Internet Explorer to always allow cookies from this Web site to be saved on your computer, select Always allow this site to use cookies. If you want Internet Explorer to never allow cookies from this Web site to be saved on your computer, select Never allow this site to use cookies.

Figure 61 illustrates a typical use of the pattern, using the Mozilla P3P Privacy Policy Viewer example, which follows the steps:

1. User requests interaction with Server to Network Client.
2. Client consults Proxy, which can be internal to client or an external plug-in.
3. Proxy discovers the Policy published by the server over the Internet.
4. The Server responds to the Client's request.
5. Proxy seeks the User's approval to the interaction. This may or may not include a step in which the Proxy can block the interaction in case the User does not approve the Policy.
6. Privacy conscious, the User continues the interaction.

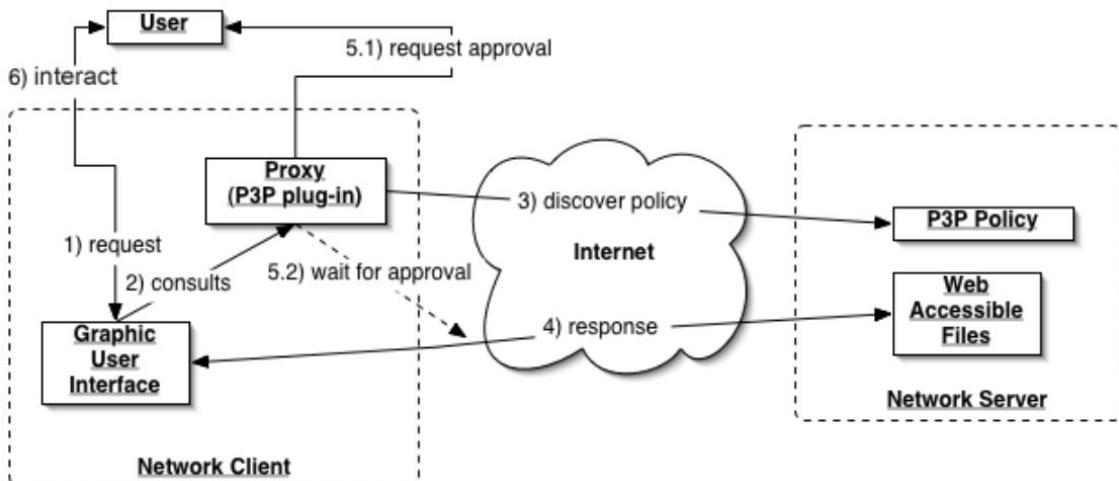


Figure 61: Privacy-aware Network Client (Mozilla) example

### 31.12. See Also

Proxy [2]. The Privacy-Aware Network Client uses a specialized version of the Proxy pattern.

Web Shopping Process [3]. This is one of the patterns most likely to be combined with this pattern.

Adaptive Web Applications [4]. These are patterns for web applications that change their behavior according to the current user. They would display more or less complete privacy disclosures depending on the type of user.

### 31.13. References

- [1] Joint Research Centre. (n.d.). P3P Resources. P3p. <http://p3p.jrc.it/>
- [2] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [3] Fernandez, E. B., Liu, Y., & Pan, R. (2001). Patterns for Internet shops. *Procs. of PLOP*.
- [4] Koch, N., & Rossi, G. (2002, July). Patterns for adaptive web applications. In *Proc. 7th European Conference on Pattern Languages of Programs*.

### 31.14. Source

Sadicoff, M., Larrondo-Petrie, M. M., & Fernandez, E. B. (2005). Privacy-aware network client pattern. In *Proceedings of the Pattern Languages of Programs Conference*.

# 32. Secure Pre-Forking

## 32.1. Intent

In a multitasking server environment, pre-forking is a widely used mechanism for improved performance. If the pre-forked processes run as daemon processes, then they pose security and reliability risks. When an attacker compromises one of the daemon processes, he can use this process to try to infect other processes more effectively because the daemon processes never die. To avoid this vulnerability, all the pre-forked processes should be created with a limited lifetime and new processes are forked in their places after this pre-defined time span.

## 32.2. Example

Apache is a well-known web server that runs on Unix as a pre-forking server. On startup, it creates a pool of child processes ready to handle incoming clients. As requests are processed, Apache tries to make sure that there are at least a few spare servers running for subsequent requests. If the process spawning cost is negligible, processes do not need to be pre-forked. Instead, a bunch of listeners listen to web requests and fork processes lazily. In practice, process creation has significant overhead. Pre-forking enhances the performance of the server by reducing the process creation overhead when a connection request comes through. However, this architecture has some security issues.

The pre-forked processes reside in a resource pool. When a request comes, the listener process hands the task to an idle process taken from the resource pool. Once the task is finished, the process returns to the resource pool. These processes remain resident until the server is shut down. The consequences of a security compromise with these processes are more severe because the processes give the attacker a working ground that remains resident in the system for a long time. This can be utilized to further exploit other processes.

How can the vulnerability associated with daemon processes be minimized?

The solution is to limit the lifetime of daemon processes. The processes are monitored so that they are killed after serving a pre-configured number of requests. The identification criterion for the processes that need to be killed is not a single parameter like the number of requests served. Multiple parameters are considered for this. The identified process is killed, and a new process is forked and included in the resource pool in its place. Information about the new process is passed to the monitor mechanism for control purpose.

Apache uses this approach. Apache has various configuration parameters that are used during server startup and process management. Startup parameters are used by the master server to spawn off a number of child processes and to put them in a resource pool. Memory resident data structures are maintained by master server for process accounting. Based on these data values, processes are replaced after their limited lifetime.

### **32.3. Context**

Network servers handling concurrent requests usually exhibit multitasking architecture with resource pooling. This pattern is used to make the resource pooling mechanism more secure. Normally a system architect uses this pattern.

A simple example of such a multitasking handler is a master server process acting as a gatekeeper to the system. It waits for a connection request and when the request arrives, it creates the connection and forks itself to create a child process. The child server process handles the request and runs in parallel to the master server that returns to listening for incoming requests.

Web servers handle a lot of incoming requests; hence performance is the key issue. If process forking takes a significant amount of time, the lazy forking by the master server means that it cannot accept incoming requests during this period. This lack of availability is the main reason why on-demand forking architecture cannot be used for network servers.

### **32.4. Problem**

The forces that need to be considered when choosing to use this pattern are as follows.

- Performance. Released resources should be reused. This reduces the overhead associated with repetitive acquisition and release.
- Latency. The latency for serving an incoming request should be minimal.
- Security. The impact of a security breach should be minimal so that the compromise of one process is limited and cannot be used for further perpetration.
- Lifetime of Processes. Processes should not be reused infinitely. They should be released after some time. Long running processes have the same vulnerability. Such processes have to be monitored to find if they are compromised or not.
- Availability. The clients expect the server to be available and they expect that their service requests will not be refused.
- Simplicity and Loose Coupling. The implementation should not introduce more bugs into the system. The monitoring logic should be separated from the main functionality of the system.
- Overhead of process monitoring. The execution overhead for running the secure pre-forking mechanism should be minimal. Process monitoring should not take up the CPU cycles saved by resource pooling. The cost for releasing resources during execution time and forking new processes in their place should be minimal.

The resource pooling pattern resolves the performance and latency aspects. Without the limitation of process lifetime, the system remains vulnerable to a security breakdown attempt.

### **32.5. Solution**

### **32.6. Structure**

Here the structure of the pattern illustrates the key components of the pattern.

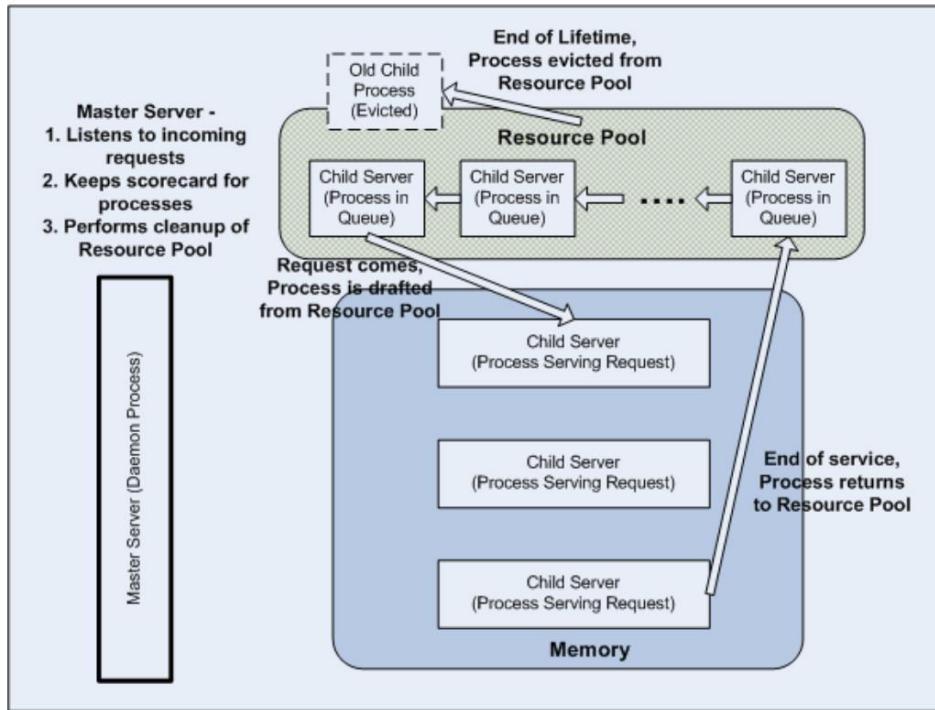


Figure 62: Secure Pre-forking structure

**Master Server Process.** The master server process listens to incoming requests. It keeps usage statistics of pre-forked processes and sends cleanup commands.

**Child Server Process.** The child server processes are pre-forked. They serve the incoming requests. Upon creation, the child server processes reside in the resource pool. When needed, they are transferred to main memory. When the task is finished, the processes are returned to the resource pool where they wait to serve future requests. After a limited lifetime, the child server processes are evicted.

**Resource Pool.** The pre-forked child-server processes wait in the resource pool.

**Memory.** The child server processes run in the main memory when they are handed incoming requests. The processes return to resource pool after the service completion.

Initially, the master server pre-forks a number of child processes and puts them in the resource pool. This pre-forking stage usually happens before the master server starts listening for client requests. The number of pre-forked processes is configured by parameters set by the system administrator. This number can change during runtime based on the number of requests coming from the client. Clients submit their connection requests to the master server. The request is assigned to one child process from the resource pool. After finishing, the child process returns to the resource pool. The master server keeps track of information about child processes, e.g. the length of time they are serving, the number of requests serviced etc. These parameters are configured through some external configuration file.

## 32.7. Dynamics

## 32.8. Implementation

Here are the issues of implementing the pattern.

1. Criteria for process killing. A key issue in pre-forking is to determine what parameters to consider when replacing the pre-forked processes. An obvious parameter is the number of requests served by a process or the duration of service. However, this parameter alone is insufficient as attackers are tempted to launch their attacks during low-traffic or off-peak hours. Hence the idle time of processes should also be considered. Another related parameter is the overall lifetime of processes. This parameter alone cannot be used. Processes are pre-forked as a group and if the only parameter is the total lifetime, then several of them reach the threshold together. The key is to consider multiple parameters.
2. Setting up the pre-forking parameters. A number of factors related to the pre-forking architecture have to be configured. Separate parameters are maintained for controlling these issues externally. For example, the number of pre-forked processes can be controlled with parameters. If the value is too high, then the server startup time is increased. If the value is too low, then soon after startup the server will require more processes to handle incoming requests and it will have to spawn new processes. The minimum and maximum value of parameters are set according to some empirical values seen from trial runs of the software.
3. Smart pre-forking. In many systems, pre-forking is handled by parameters specified through configuration files (see the example of parameters in Apache implementation example). A simple approach to pre-forking uses the parameters directly. For example, if 'x' is the number of resources specified through some external configuration, then 'x' resources are pre-forked. A smart pre-forking approach reduces the overhead of pre-forking by creating processes based on load. The system can use an algorithm that incrementally increases the number of processes spawned dynamically in response to increasing load. A pre-forking server may spawn a fixed number of processes and then double this number when more requests come and so on. An example is given in the Apache implementation section.
4. Process monitoring. The processes have to be monitored to do accounting on the creation time and use of processes. A separate data structure has to be maintained in memory to do that. Careful monitoring has to be done so that it does not become a performance overhead.
5. Pre-forking of resources. The idea of resource pooling can be extended to pre-fork instances of threads, sockets etc. Windows implementation of resource pooling is primarily based on thread pooling.
6. Freeing of resources. Process eviction is an important aspect of the pre-forking architecture. The process eviction task can be entrusted to a daemon process. The process is simple and does nothing else than freeing of resources. This simplicity means that it can be thoroughly tested and made bug free.
7. Deadlock recovery in secure pre-forking. Pre-forking works best with systems where the incoming requests are for short-lived tasks. If the processes handle tasks that take a long time it also makes the system vulnerable. The processes therefore go through an auditing phase and the processes that are blocked for a long time are identified and killed. This has an additional advantage. The pre-emptive killing of a process that is blocked for a long time provides a deadlock recovery mechanism. However, the most

difficult issue is to identify whether a process, that is handling a task that naturally takes a long time, is compromised or not. If a process that was running under normal circumstances is killed, the task entrusted to the process is not finished. This has severe reliability and availability issues.

## 32.9. Example Resolved

## 32.10. Consequences

The pattern has the following benefits.

- Increased security. In a process-based system architecture, the key principle for a secure architecture is the insulation of processes. An example is the architecture of qmail. qmail is a secure mail transfer agent. The primary tasks of a mail transfer agent are receiving mail from local and remote hosts, storing the mail, and delivering the mail to local and remote recipients. In the qmail architecture, the processes are partitioned according to their functionalities. There are separate processes for local and remote mail recipients, the queue manager, and local and remote mail senders. These processes do not trust each other and validate the communication payload between them. Hence even if some process is compromised, other parts of the system remain unharmed because they run in a separate address space. In qmail, most of the processes are not daemon processes, and so even if some process is compromised, after a brief period it dies and is garbage collected. The attacker cannot use this process to try to compromise other processes. Even if some attacker does not try to infect other processes, he can send a garbage payload to other processes and create an internal Denial of Service (DoS) scenario. Secure pre-forking adds to the security of the overall system by limiting the lifetime of the daemon processes so that the effect of a process compromise becomes transient.
- Improved Performance. Secure pre-forking improves performance because processes do not have to be spawned for each incoming request. If the pre-forked processes run as daemons, it should have even better performance because the cost of process creation is entirely eliminated. However, in this scenario security is an equally important requirement along with performance. The limited lifetime of processes adds some overhead of forking more processes, but this limit adds a lot to the entire system being secure.
- Availability. The main problem with a server that forks processes on demand is the process creation overhead incurred every time a request comes in. When the master server is forking some child server process, the requests that come at that moment are not served. Normally this happens only if the delay for forking a process is not negligible. However, if the load is high the delay for forking cannot be avoided. It is inconvenient for the communicating entity to find their requests refused by the server. This is not a serious problem in mail transfer agent architecture as processes communicate with the server and they can start again. However, for web server architecture, the communicating entity is a human user, and it is a major disturbance for the client to see his web page access request denied because the server was unavailable.

- Defense in Depth. When the processes run under an owner with a low privilege level, an attacker after compromising a process cannot do a lot of harm. In that case it can be argued that running daemon processes may not be harmful. The counter argument is that daemon processes can still be vulnerable, and it might still contain a hole that comes from bad programming. Running the processes as daemon even at a low privilege level is dangerous. The secure pre-forking pattern follows the defense in depth principle [1] by limiting the vulnerability associated with processes with long lifetime.

The pattern has the following liabilities.

- Complexity of pre-forking architecture. The major problem with secure pre-forking architecture is the complexity. This means more faults, less portability, and a larger binary. The performance improvement of pre-forking only becomes evident in the case of heavy load. In the case of light load, the pre-forked processes occupy memory and become a bottleneck instead.
- Negative impact on availability. This pattern can lead to limited availability. Let us suppose that a batch of pre-forked processes have reached their threshold and should be killed. However, at that moment the server is experiencing heavy load. If the master server begins pre-forking new processes, some incoming requests may go unserved. Two things can be done to counter this scenario, a) killing the processes but not forking new processes in the resource pool or b) temporarily halting the killing of processes. If the processes are killed but new processes are not spawned off, then the processes remaining in the resource pool may be insufficient to handle the incoming load. In that case, after some time the server may have to lazily fork new processes. Halting the killing of processes is another option. However, this solution is temporary and after a short delay the marked-down processes are killed. If the stoppage is not temporary, it leads to another vulnerability. An attacker, after the compromise of one process, may launch a DoS attack to keep the master server in heavy load. A heavy load means the server cannot kill processes. So, the processes effectively become daemon processes. To avoid this, after some time the processes that have passed their life limit are killed anyway. This is an example of trade-off between availability and security.

## 32.11. Known Uses

We provide here two examples from Unix server domain. One of them is the Apache web server. The other is the Postfix mail transfer agent.

1. Implementation in Apache. Apache runs on Unix platforms as a pre-forking server. On startup, it creates a pool of child processes ready to handle incoming client requests. As requests are processed, Apache tries to make sure that there are at least a few spare servers running for subsequent requests. Apache provides three directives to control the resource pool.

`StartServers < number > (default 5)` This determines the number of child processes Apache will create on startup.

**MinSpareServers < number >** (default 5) This determines the minimum number of Apache processes that must be available at any one time; if processes become busy with client requests, Apache will start up new processes to keep the pool of available servers at the minimum value.

**MaxSpareServers < number >** (default 10) This determines the maximum number of Apache processes that can be idle at one time; if many processes are started to handle a peak in demand and then the demand tails off, this directive will ensure that excessive numbers of processes will not remain running.

These directives used to be more significant than they are now. Since version 1.3 Apache has a very responsive algorithm for handling incoming requests, starting from 1 to a maximum of 32 new processes each second until all client requests are satisfied. The objective of this is to prevent Apache from starting up excessive numbers of processes all at once unless it is actually necessary because of the performance cost. The server starts with one, then doubles the number of new processes started each second, so only if Apache is genuinely experiencing a sharp rise in demand will it start multiple new processes.

Apache's smart and dynamic handling of the server pool makes it capable of handling large swings in demand. Adjusting these directives has little actual effect on Apache's performance except in extremely busy sites.

Apache has another two directives related to the control of processes:

**MaxClients < number >** (default 256) Irrespective of how busy Apache gets, it will never create more processes than the limit set by MaxClients, either to maintain the pool of spare servers or to handle requests. Clients that try to connect when all processes are busy will get 'Server Unavailable' error messages.

**MaxRequestsPerChild < number >** (default 0) This limits the maximum number of requests a given Apache process will handle before voluntarily terminating. The objective of this is to prevent memory leaks causing Apache to consume increasing quantities of memory; while Apache is well behaved in this respect the underlying platform might not be. Setting this to zero means that processes will never terminate themselves, but this has security consequences.

A low value for the MaxRequestsPerChild directive will cause performance problems as Apache will be frequently terminating and restarting processes. A more reasonable value for platforms that have memory leak problems is 1000 or 10000:

MaxRequestsPerChild 10000

The Unix version of Apache also runs in a multi-threaded mode, therefore the ThreadsPerChild directive is also significant. This is used for secure resource pooling for threads.

Detailed documentation of Apache implementation can be found in [2] and [3].

2. Unix Implementation example in Postfix. Postfix [4] uses this pattern to implement the remote mail recipient. Detailed description of Postfix's use of this pattern is described in [5].

qmail [6] was the first Mail Transfer Agent (MTA) with security as one of the primary requirements. Postfix follows qmail architecture in many places [5, 7], but it has some clever performance hacks that contribute to overall performance improvement. In qmail, processes are forked on demand and their lifetime is limited for the duration of serving the request. Postfix improves this by resource pooling. The size of the resource pool is specified by the system administrator. Postfix does not have sophisticated modules for resource pooling and load balancing like Apache. Instead, the pre-forked processes serve a fixed number of requests and then die. This number is also specified by the system administrator. After the process dies, the parent process forks off a replacement. To limit the vulnerabilities associated with pre-forked processes, the pre-forking parameters have to be set carefully.

The performance improvement in Postfix is evident from benchmark tests in comparison with other MTAs [8, 9].

## 32.12. See Also

The Pooling [10] pattern focuses on the pre-forking and resource pool creation issues. The Resource Lifecycle Manager [10] pattern decouples the management of lifecycle of resources from their use by introducing a separate resource lifecycle manager, whose sole responsibility is to manage and maintain the resources of an application. This can be applied for process monitoring. Another important aspect of the secure pre-forking pattern is the release of resources. The Evictor [10] pattern describes how and when to release resources to optimize resource management.

The interesting fact about security patterns is that there are analogous mechanisms in human immune systems. The key thing about the secure pre-forking pattern is the killing of processes that are long lived. The Programmatic Cell Death (PCD) mechanism known as Apoptosis [11] is an analogous mechanism where healthy cells are programmatically killed for some benefit. The cells that are killed are not degenerated. Similarly, in secure pre-forking processes that are killed do not have any problem. This is done for security purpose only.

## 32.13. References

- [1] Viega, J., & McGraw, G. R. (2001). *Building secure software: How to avoid security problems the right way, portable documents*. Pearson Education.
- [2] Gröne, B., Knöpfel, A., Kugel, R., & Schmidt, O. (2004). *The Apache modeling project*.
- [3] The Apache HTTP Server Project. (n.d.). Apache. <https://httpd.apache.org/docs-project/>
- [4] The Postfix Home Page. (n.d.). Postfix. <http://www.postfix.org/>
- [5] Hafiz, M. (2005). *Security architecture of mail transfer agents* (Doctoral dissertation, University of Illinois at Urbana-Champaign).
- [6] qmail: the Internet's MTA of choice. (n.d.). Cr.Yp.To. <http://cr.yp.to/qmail.html>

- [7] Hafiz, M. (2005, October). Security patterns and evolution of MTA architecture. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 142-143).
- [8] Andree, M. (n.d.). MTA benchmark. <http://www.dt.e-technik.uni-dortmund.de/~ma/postfix/bench2.html>.
- [9] Andree, M. (n.d.). Postfix vs. qmail - performance. <http://www.dt.e-technik.uni-dortmund.de/~ma/postfix/vsqmail.html>.
- [10] Kircher, M., & Jain, P. (2004). *Pattern-oriented software architecture, patterns for resource management*. John Wiley & Sons.
- [11] Kerr, J. F., Wyllie, A. H., & Currie, A. R. (1972). Apoptosis: a basic biological phenomenon with wideranging implications in tissue kinetics. *British journal of cancer*, 26(4), 239-257.

## 32.14. Source

- Hafiz, M. (2005, December). Secure pre-forking-a pattern for performance and security. In *Proceedings of the Conference on Pattern Languages of Programs* (pp. 1-9).

# 33. Input Guard

## 33.1. Intent

Often, large scale systems are an assembly of independently developed components, like in the case of systems built out of Commercial off the Shelf (COTS) components. In such cases the developer of an individual component wants to prevent errors that may occur elsewhere in the system from infecting the component he developed. One way to achieve this is to verify that every input fed to his component by the system conforms to the input for his component as described in the system specification.

## 33.2. Example

## 33.3. Context

The Input Guard pattern applies to a system which has the following characteristics:

- The system is composed from distinguishable components, which can play the role of fault compartments, and which interact with each other by feeding one's output into another's input.
- It is possible, either directly or implicitly, to validate the input of a component against the legitimate input described in the component's specification.
- The errors that can be propagated into a system component have the form of erroneous input, i.e. input whose content or timing does not conform to the system specification.

The second characteristic implies that internal errors (e.g. changes to the internal state due to electromagnetic disturbances in the environment where the system operates) are not considered by this pattern since they are not expressed as erroneous input according to the system specification. Moreover, this pattern does not deal with cases where the input to the system conforms with the system specification, but it still contains errors according to the specification of the system's environment (e.g. see case (b) in Figure 63).

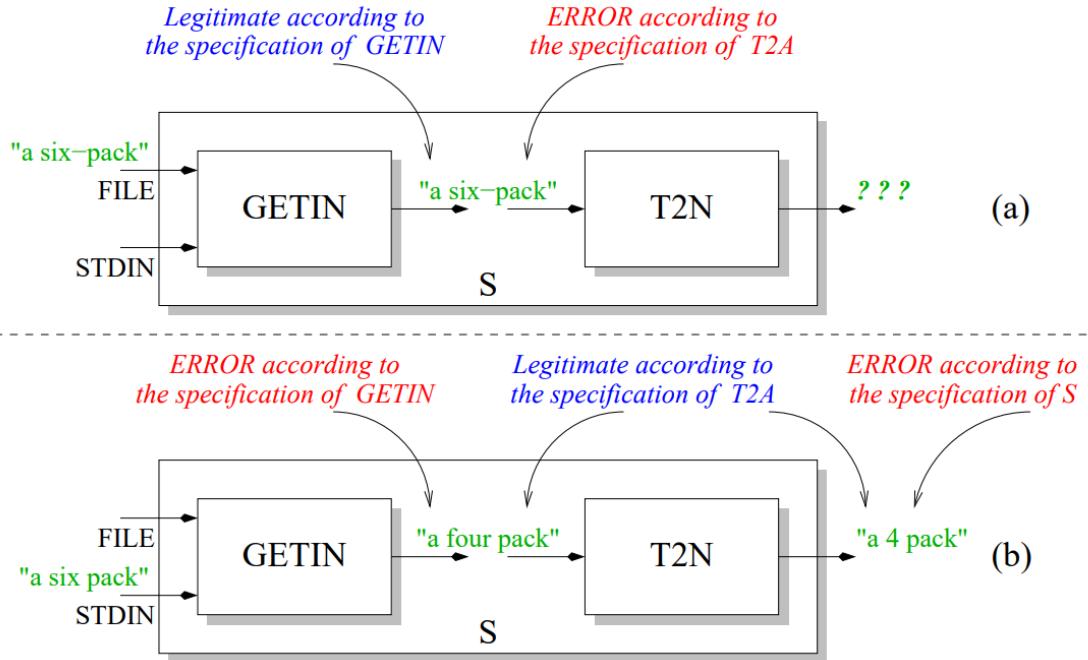


Figure 63: Example of errors that may occur to system  $S$  and its constituent components.

### 33.4. Problem

In the above context, the Input Guard pattern solves the problem of stopping the propagation of an error from the outside to the inside of the guarded component by balancing the following forces:

- A system may be composed of components that are developed independently from each other, without keeping strict consistency with each other's specifications.
- A system may be infected with errors from its environment even when the environment is not experiencing any errors according to its specification.
- Different systems have different requirements regarding size impact of the fault containment mechanism.
- Different systems have different requirements regarding the time penalty of the fault containment mechanism.
- Fault containment is usually integrated with other solutions provided for other fault tolerance constituents (e.g. error masking, error detection and fault diagnosis) in order to provide wider fault tolerance guarantees.

### 33.5. Solution

To stop erroneous input from propagating the error inside a component a guard is placed at every access point of the component to check the validity of the input. Every input to the guarded component is checked by the guard against the component specification. If and only if the input conforms with that specification, then it is forwarded to the guarded component.

Notice that the above solution does not define the behavior of the guard in the presence of erroneous input, besides the fact that it does not forward it to the guarded component. This is intentionally left undefined in order to allow implementations of the Input Guard to be

combined with error detection mechanisms (e.g. when a check fails, an error notification is sent to the part of the system responsible for fault diagnosis) or with the implementations of error masking mechanisms (e.g. the comparator entity of the Active Replication pattern [1]). Hence, the behavior of the guard when the checks performed on the input fail depends on the other fault tolerance constituents with which the input guard is combined.

### 33.6. Structure

The Input Guard pattern introduces two entities:

- The guarded component which is the part of the system that is protected against the fault contamination from external errors propagated to it through its input.
- The guard which is responsible to check for errors in the input to the guarded component against its specification.

There may be many instances of the guard entity for the same guarded component, depending on the system design and on the number of different access points the guarded component may have. For example, a software component with a number of interfaces and a number of operations declared in each interface may have one guard per interface or one guard per operation declared in its interfaces or any possible combination of those. Figure 64a illustrates graphically the structure of the Input Guard pattern for a guarded component with a single access point. Figure 64b contains the activity diagram that describes the functionality of the guard.

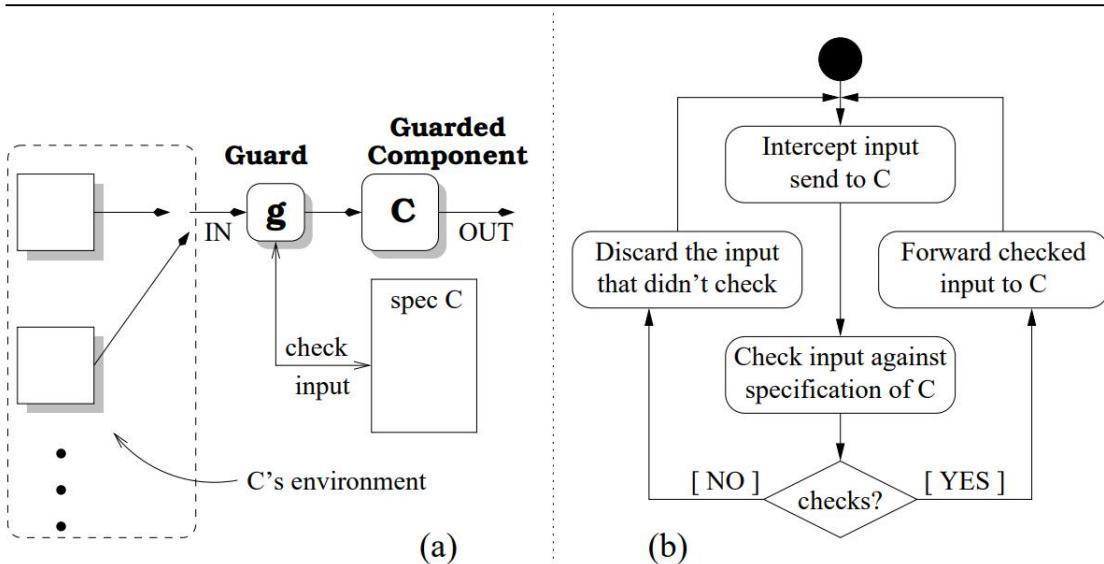


Figure 64: The structure (a) and the activity diagram (b) of the Input Guard pattern.

One possibility is to implement the guards as separate components in the system. This approach allows to have a number of guards proportional only to the number of the access points of the guarded component. The time overhead introduced by this approach is quite high since it includes the invocation of an additional component (i.e. the guard). Also, the space overhead of this approach is rather elevated since it increases the number of the components in a system by the number of guards that are implemented. Furthermore, in the case where components are mapped to individual units of failure (i.e. each component can fail as a whole and independently of other components) this approach introduces a well-known

dilemma in fault tolerance: “*QUIS CUSTODIET IPOS CUSTODES?*” (“who shall guard the guards?”). This dilemma has also well-known consequences, which are difficult and very costly to resolve (e.g. redundant instances of the guards, distributed among the constituent components of a system, which have their own synchronization protocol).

Despite the above inconveniences, this implementation approach is valuable in the case of COTS-based systems composed from black-box components where the system composer does not have access to the internals of the components. Also, this approach can be applied when fault containment comes as a late- or after-thought in the system development and a quick fix is needed in form of a patch. This implementation approach does not require any modification on existing components of a system; rather, guards are introduced as separate add-on components to the existing system.

Another implementation approach is to make the guard part of the implementation of the guarded component. This practice is often employed in programming where a method checks its arguments before using them to perform its designated task. This allows the coupling of the guard(s) and the guarded component. By integrating the guard with the guarded component, the space overhead of the Input Guard implementation is kept low since it does not introduce another component in the system. Coupling the guard and guarded component implementation is usually applied in the development of COTS software where the developer has no knowledge about the rest of the system in which the component will be integrated. Hence, in order to assure robust functioning of a component, the developer checks the input of the component on every call. The drawback of this implementation approach is the fact that the time overhead is high and fixed. This is because the guard is engaged on every call to the guarded component, even when the supplied input has already been checked by other fault tolerance means.

A third implementation possibility is to place the guard inside each of the components which may provide input to the guarded component. This approach allows the integration of the guard with other fault tolerance mechanisms (e.g. the guard of the Output Guard pattern for each component that provides input to the guarded component). Furthermore, this approach allows the elimination of redundant checks for errors which can increase the time and space overhead of fault tolerance solutions in a system. On the other hand, this approach is not applicable to COTS software. Third party developers may not have information about the specification of the other components to which they will feed their output, hence they do not know what conditions to check in the guard. A drawback of this implementation approach is the elevated space overhead; the number of guards is not only proportional to the access points of the guarded component but also to the number of components that provide input to the guarded component. Another drawback is that this guard cannot protect the guarded component from communication errors that occurred during the forward of the checked input from the guard to the guarded component. On the positive side however, this approach allows the guard to be selectively integrated only with those components that are considered not robust enough and subject to produce erroneous input for the guarded component. This can be used to reduce the elevated space overhead of the approach.

### **33.7. Dynamics**

### **33.8. Implementation**

### **33.9. Example Resolved**

### **33.10. Consequences**

The Input Guard pattern has the following benefits:

- It stops the contamination of the guarded component from erroneous input that does not conform to the specification of the guarded component.
- The undefined behavior of the guard in the presence of errors allows its combination with error detection and error masking patterns, and fault diagnosis mechanisms. Whenever this is applicable, the system benefits in terms of reduced run-time overhead introduced by the implementation of the fault tolerant mechanism (e.g. the combination of fault containment and error detection in the context of system recovery from errors).
- The similarities between the guard entities of the Input Guard pattern and Output Guard pattern allow the combination of the two in a single entity. This entity will operate on the same data and will perform two checks: one against the specification of the component that produced the data as output and the other against the specification of the component that will consume the data as input. When applicable, this combination can provide significant benefits in terms of time and space overhead since two separate checks will be performed by the same piece of code.
- There are various ways that the Input Guard pattern can be implemented, each providing different benefits with respect to the time or space overhead introduced by the guard. It is also possible to integrate the guard with an existing system without having to modify the internals of the system components (first implementation alternative). That significantly reduces the amount of system re-engineering required for applying the Input Guard pattern to COTS-based systems made of black-box components.

The Input Guard pattern also imposes some liabilities:

- It is not possible to minimize both the time and the space overhead of this pattern. To keep low the time overhead introduced by the Input Guard pattern, the functionality of the guard must not be very time consuming. This results in a tendency to introduce a separate guard for each different access point (e.g. one guard per interface or even per operation declared in an interface) of the guarded component. Each such guard checks only a small part of the specification of the guarded component, minimizing thus the execution time of an individual guard. However, this results in a large number of guards, hence in an elevated space overhead. On the other hand, to keep low the space overhead introduced by the Input Guard pattern, the number of guards needs to remain as small as possible. This implies that each guard will have to check a larger number of input for the guarded component, becoming a potential bottleneck and thus penalizing the performance of the system with elevated time overhead.

- For certain systems that require guards to be implemented as components (e.g. systems composed from black-box COTS software), the Input Guard pattern results unavoidably to an elevated time and space overhead. The space overhead is due to the introduction of the new components implementing the guards. The time overhead is due to the fact that passing input to the guarded component requires one additional indirection through the component implementing the guard that check the given input.
- The Input Guard pattern cannot prevent the propagation of errors that do conform with the specification of the guarded component (e.g. see case (b) in Figure 63). Such errors may contaminate the state of the guarded component if it has one. Although these errors cannot cause a failure on the guarded component since it operates according to its specification, they can cause a failure on the rest of the system. Such a failure of the entire system will be traced back to an error detected in the contaminated guarded component. Unless the error detection and fault diagnosis capabilities of the system allow to continue tracing the error until the initial fault that caused it, it is possible that inappropriate recovery actions will be taken targeted only at the guarded component, which, nonetheless, has been operating correctly according to its specification.
- The Input Guard pattern can effectively protect a component from being contaminated by erroneous input according to its specification. However, unless it is combined with some error detection and system recovery mechanisms, this pattern will result in a receive-omission failure (i.e. failure to receive input) of the guarded component. For certain systems, such a failure of one of their components may cause a failure on the entire system. Hence, the Input Guard pattern has limited applicability to such systems if it is not combined with other fault tolerance patterns.

### **33.11. Known Uses**

The guard entity can be seen as one possible realization of the pre-conditions validation prior to the execution of a piece of code, as this is described in the design by contract principles [2]. The concept of conditions guarding the execution of tasks has been introduced as monitors already in the '70s [3]. Monitors, however, would not prevent the flow of erroneous input to the guarded component; rather, they would prevent the guarded task from being executed if the starting/input conditions were not met. Nowadays, the majority of the books that introduce a programming or scripting language also instruct the validation of input arguments upon the call of a function, procedure, method, routine or whatever the name of a functionality block in the corresponding language.

### **33.12. See Also**

The Input Guard pattern has similarities with the Output Guard pattern. Also, the guard entity of this pattern complements the acknowledger entity of the Acknowledgment pattern [1] in combining fault containment and error detection. The acknowledger is responsible to inform the sender of some input about the reception of it. The combination of the acknowledger and the guard will provide a confirmation of the reception of correct input and a notification in the case of reception of erroneous input.

### **33.13. References**

- [1] Saridakis, T. (2002, July). A System of Patterns for Fault Tolerance. In *EuroPLoP* (pp. 535-582).
- [2] Meyer, B. (1997). *Object-oriented software construction* (Vol. 2, pp. 331-410). Englewood Cliffs: Prentice hall.
- [3] Hoare, C. A. R., & Monitors, C. A. R. An operating system structuring concept. *CACM. october 1974.*

### **33.14. Source**

Saridakis, T. (2003, June). Design Patterns for Fault Containment. In *EuroPLoP* (pp. 493-520).

# 34. Output Guard

## 34.1. Intent

Whereas the Input Guard pattern prevents an error in the input of the guarded component from contaminating that component, it is often highly desirable stop the error when it occurs rather than when it is about to be propagated to another component. The Output Guard pattern describes how to confine an error in the component that contains the fault which led to that error. The technique described by the Output Guard pattern is very similar to the one described by the Input Guard pattern: the output of a component is checked against the specification of the component to ensure conformance. Despite the similarity of the technique, these two patterns serve different purposes. The Input Guard pattern prevents the contamination of a component from an error occurred elsewhere while the Output Guard pattern confines an error inside the component where that error occurred.

## 34.2. Example

## 34.3. Context

The Output Guard pattern applies to a system that has the following characteristics:

- The system is composed from distinguishable components, which can play the role of fault compartments, and which interact with each other by feeding one's output into another's input.
- It is possible, either directly or implicitly, to validate the output of a component against the legitimate output described in the component's specification.
- The errors that occur in a system are expressed as erroneous system output according to its specification, i.e. output whose content or timing does not conform to the system specification.

Similar observations as for the Input Guard pattern apply here. The second characteristic implies that internal errors (e.g. changes to the internal state due to electromagnetic disturbances in the environment where the system operates) are not considered by this pattern unless they result in erroneous output according to the system specification. Moreover, this pattern does not deal with cases where the output of the system conforms with the system specification, but it still contains errors according to the specification of the system's environment (e.g. see case (a) in Figure 63).

## 34.4. Problem

In the above context, the Output Guard pattern solves the problem of confining an error inside the component where it occurs by balancing the following forces:

- A system may be composed of components that are developed independently from each other, without keeping strict consistency with each other's specifications.

- A system may infect with errors its environment when the environment is not able to distinguish the erroneous output from a correct one.
- Different systems have different requirements regarding size impact of the fault containment mechanism.
- Different systems have different requirements regarding the time penalty of the fault containment mechanism.
- Fault containment is usually integrated with other solutions provided for other fault tolerance constituents (e.g. error masking, error detection and fault diagnosis) in order to provide wider fault tolerance guarantees.

### 34.5. Solution

To stop an error from being propagated outside the component where it occurred, a guard is placed at every exit point of the component (be it message emission or invocation return point). Each such guard checks the produced output against the specification of the component. If and only if the output conforms with the component specification, then it is allowed to reach the component's environment.

Notice the above solution does not define the behavior of the guard in the presence of erroneous output, besides the fact that it does not allow it to reach the component's environment. Similarly, to the Input Guard pattern, this behavior is intentionally left undefined in order to allow implementations of the Output Guard pattern to be combined with error detection mechanisms (e.g. when a check fails, a notification is sent to the error detection mechanism [1]). Hence, the behavior of the guard when the checks performed on the output fail depends on the other fault tolerance constituents with which the input guard is combined.

### 34.6. Structure

Since the technique captured by the Output Guard pattern is very similar to the one captured by the Input Guard pattern, it does not come as a surprise that the entities the former introduces are similar to those introduced by the latter:

- The guarded component which is the part of the system, which is guarded against the occurrence of errors, and in which occurred errors will be confined.
- The guard which is responsible to check for errors the output to the guarded component against its specification.

Similarly, to the Input Guard pattern, there may be many guards for the same guarded component, depending on the system design and on the number of different exit points the guarded component may have. For example, a software component with a number of interfaces and a number of operations declared in its of them can have one guard per interface, or one guard per message-send and return operation, or any possible combination of those. Figure 65a illustrates graphically the structure of the Output Guard pattern for a guarded component with a single exit point. Figure 65b contains the activity diagram that describes the functionality of the guard.

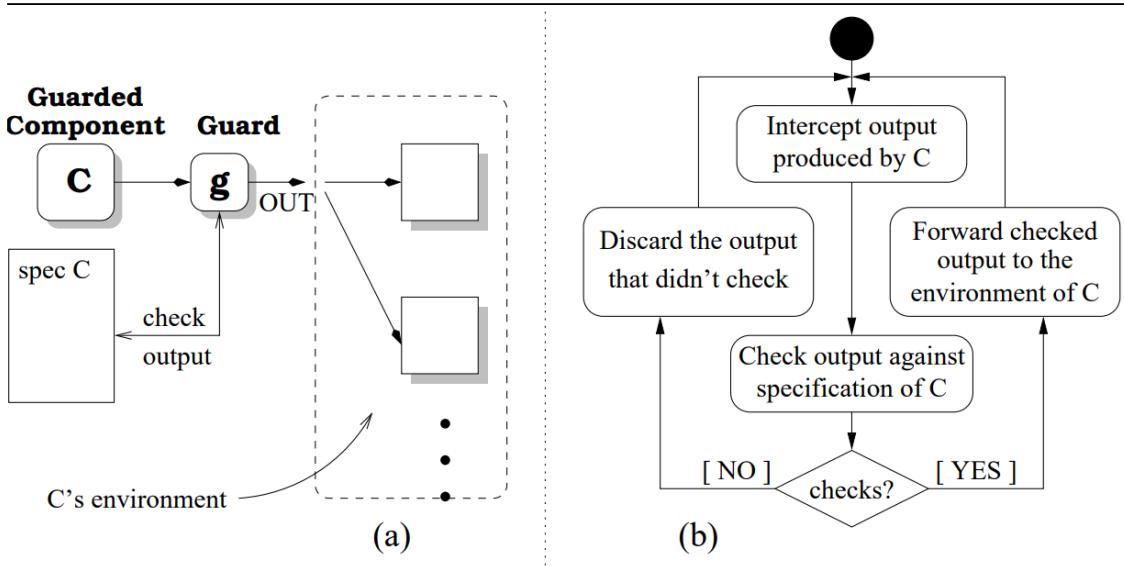


Figure 65: The structure (a) and the activity diagram (b) of the Output Guard pattern.

One way to implement the guards is as separate components in the system. This approach has the same advantages and inconveniences as its counterpart in the Input Guard pattern. It allows to have a number of guards proportional only to the number of the access points of the guarded component. However, the time overhead is quite high since it includes the invocation of an additional component (i.e. the guard). Also, the space overhead of this approach is rather elevated since it increases the number of the components in a system by the number of guards that are implemented. Furthermore, in the case where components are mapped to individual units of failure (i.e. each component can fail as a whole and independently of other components) the problem of “guarding the guards” raises again as for the Input Guard pattern. This dilemma has also well-known consequences, which are difficult and very costly to resolve (e.g. redundant instances of the guards, distributed among the constituent components of a system, which have their own synchronization protocol).

Despite the above inconveniences, this implementation approach is valuable in the case of COTS-based systems composed out of black-box components where the system composer does not have access to the internals of the components. Also, this approach can be applied when fault containment comes as a late- or after-thought in the system development and a quick fix is needed in form of a patch. This implementation approach does not require any modification on existing components of a system; rather, guards are introduced as separate add-on components to the existing system.

Another implementation approach is to make the guard part of the implementation of the guarded component. This practice is often employed in programming where a method checks the validity of the output it produced before returning it to its environment (e.g. to its caller). This allows the coupling of the guard(s) and the guarded component. By integrating the guard with the guarded component the space overhead of the Output Guard implementation is kept low since it does not introduce another component in the system. Coupling the guard and guarded component implementation is usually applied when developing software components that are meant to upgrade an existing system. In these cases, the developer of the new software component wants to make sure that the produced output will not introduce any errors to the existing system. To archive this, the developer of the component introduces self-

checking code for the output the component produces (e.g. see [2] and [3] for more information on self-checking code). The drawback of this implementation approach is the fact that the time overhead is high and fixed. This is because the guard is engaged every time the guarded component produces output, even when that output will be anyway checked as whether it is valid input for the component which will receive it (e.g. see the Input Guard pattern).

A third possibility is to place the guard inside each of the components which may receive the output produced by the guarded component. This approach allows the integration of the guard with other fault tolerance mechanisms (e.g. the guard of the Input Guard pattern for each component that receives the output produced by the guarded component). Furthermore, this approach allows the elimination of redundant checks for errors which can increase the time and space overhead of fault tolerance solutions in a system. This is the case when the guard entities of the Input Guard and Output Guard patterns are integrated. Then, each guard will perform on the same data its own checks with respect to different component specification: the former guard checks the data against the specification of the component that produces the output and the latter against the specification of the component that receives the input.

On the other hand, this approach is not applicable to COTS software. Third party developers may not have information about the specification of the other components from which they will receive input, hence they do not know what conditions to check in the guard. A drawback of this implementation approach is the elevated space overhead; the number of guards is not only proportional to the exit points of the guarded component but also to the number of components that receive as input the output produced by the guarded component. On the positive side however, when this implementation approach is applicable it also protects the environment of the guarded component from communication errors that occurred during the forward of the produced output to the guard. Attention is required when this approach is combined with an error detection and fault diagnosis mechanism, because it is difficult to deduce the source of the error, i.e. whether it is a communication error or it is due to a fault in the guarded component. Another advantage of this approach is the fact that the guard can be selectively integrated only with those components which are not robust enough to resist the propagation of errors occurred in the guarded component. This can be used to reduce the elevated space overhead of the approach.

### **34.7. Dynamics**

### **34.8. Implementation**

### **34.9. Example Resolved**

## 34.10. Consequences

The Output Guard pattern has the following benefits:

- It confines an error to the component where it occurred by forwarding to the component's environment only output that conforms to the specification of the component.
- The undefined behavior of the guard in the presence of errors allows its combination with error detection and error masking patterns, and fault diagnosis mechanisms (e.g. see [1]). Whenever this is applicable, the system benefits in terms of reduced run-time overhead introduced by the implementation of the fault tolerant mechanism (e.g. the combination of fault containment and error detection in the context of system recovery from errors).
- The similarities between the guard entities of the Input Guard and Output Guard patterns allow the combination of the two in a single entity. This entity will operate on the same data and will perform two checks: one against the specification of the component that produced the data as output and the other against the specification of the component that will consume the data as input. When applicable, this combination can provide significant benefits in terms of time and space overhead since two separate checks will be performed by the same piece of code.
- There are various ways that the Output Guard pattern can be implemented, each providing different benefits with respect to the time or space overhead introduced by the guard. It is also possible to integrate the guard with an existing system without having to modify the internals of the system components (first implementation alternative). That significantly reduced the amount of system re-engineering required for applying the Output Guard pattern to COTS-based systems made of black-box components.

The Output Guard pattern also imposes some liabilities, similar to those of the Input Guard pattern:

- It is not possible to minimize both the time and the space overhead of this pattern. To keep low the time overhead introduced by the Output Guard pattern, the functionality of the guard must not be very time consuming. This results in a tendency to introduce a separate guard for each different exit point (e.g. one guard per invocation-return or per message-send) of the guarded component. Each such guard checks only a small part of the specification of the guarded component, minimizing thus the execution time of an individual guard. However, this results in a large number of guards, hence in an elevated space overhead. On the other hand, to keep low the space overhead introduced by the Output Guard pattern, the number of guards needs to remain as small as possible. This implies that each guard will have to check a larger number of output for the guarded component, becoming a potential bottleneck and thus penalizing the performance of the system with elevated time overhead.
- For certain systems that require guards to be implemented as components (e.g. systems composed from black-box COTS software), the Output Guard pattern results unavoidably to an elevated time and space overhead. The space overhead is due to the introduction of the new components implementing the guards. The time overhead is due to the fact that passing output from the guarded component to its environment

requires one additional indirection through the component implementing the guard that check the given output.

- The Output Guard pattern cannot prevent the propagation of errors that do conform with the specification of the guarded component (e.g. see case (a) in Figure 63). Such errors are not due to a malfunction of the guarded component and do not affect its internal state. Although these errors do not have their source in the guarded component which is checked to produce output according to its specification, they can cause a failure on the rest of the system. Such a failure of the entire system will be traced back to an error detected in the guarded component. Unless the error detection and fault diagnosis capabilities of the system allow the detection of faults in the system design, it is highly probable that inappropriate recovery actions will be taken targeted at the guarded component, which, nonetheless, has been operating correctly according to its specification.
- The Output Guard pattern can effectively protect the component's environment from being contaminated by erroneous output produced by the component according to its specification. However, unless it is combined with some error detection and system recovery mechanisms, this pattern will result in a send-omission failure (i.e. failure to deliver output) of the guarded component. For certain systems, such a failure of one of their components may cause a failure on the entire system. Hence, the Output Guard pattern has limited applicability to such systems if it is not combined with other fault tolerance patterns.

### 34.11. Known Uses

Contrary to the relation of the Input Guard with the design by contract principles [4] where the guard maps to the pre-conditions, the Output Guard pattern has little relation with them. In the design by contract the post-conditions are properties guaranteed at the end of a successful process execution, rather than being conditions checked to verify whether the execution was successful or not. Applications of the Output Guard pattern are found in Double Modular Redundant and Triple Modular Redundant systems where the guard checks whether the output of two or three replicas of the guarded component is identical and, if not, it does not deliver it to the environment of the guarded system. A special case of application for the Output Guard pattern is the use of exceptions in various programming languages (e.g. C++, Java) and distributed computing frameworks (e.g. various implementations of the CORBA specifications). In those cases, the guard explicitly knows about output of the guarded component that is exceptional and must be treated by the exception handlers rather than been let through to the environment of the guarded component.

### 34.12. See Also

Besides the obvious similarities with the Input Guard pattern, the Output Guard pattern relates to the Fail Stop Processor (FSP) pattern [5]. In the FSP pattern a comparator entity receives the output of two or more identical processor entities and compares it. If all output received is identical, then the output is forward to the environment; otherwise, the comparator stops delivering any further output to the environment. A combination of the guard and comparator entities would enable the self-checking code to identify which of the

processors has failed and notify the error detection and system recovery mechanisms in order for them to take the appropriate actions.

### **34.13. References**

- [1] Leveson, N. G., Cha, S. S., Knight, J. C., & Shimeall, T. J. (1990). The use of self checks and voting in software error detection: An empirical study. *IEEE Transactions on Software Engineering*, 16(4), 432-443.
- [2] Yau, S. S., & Cheung, R. C. (1975). Design of self-checking software. *ACM SIGPLAN Notices*, 10(6), 450-455.
- [3] Rosenblum, D. S. (1992). Towards a method of programming with assertions. In *Proceedings of the 14th international conference on Software engineering* (pp. 92-104).
- [4] Meyer, B. (1997). *Object-oriented software construction* (Vol. 2, pp. 331-410). Englewood Cliffs: Prentice hall.
- [5] Saridakis, T. (2002). A System of Patterns for Fault Tolerance. In *EuroPLoP* (pp. 535-582).

### **34.14. Source**

Saridakis, T. (2003, June). Design Patterns for Fault Containment. In *EuroPLoP* (pp. 493-520).

# 35. Fault Container

## 35.1. Intent

The Fault Container pattern aggregates the fault containment techniques described by the Input Guard and the Output Guard patterns. This pattern proposes the use of a wrapper that transforms a software component into its fault containing counterpart.

## 35.2. Example

### 35.3. Context

The Fault Container pattern applies to a system which has the following characteristics:

- The system is composed from distinguishable components, which can play the role of fault compartments, and which interact with each other by feeding one's output into another's input.
- It is possible, either directly or implicitly, to validate the input and output of a component against the legitimate input and output described in the component's specification.
- The errors that occur in the system are expressed as erroneous input (i.e. input whose content or timing does not conform to the system specification) or as erroneous output (i.e. output whose content or timing does not conform to the system specification).

The second characteristic implies that the Fault Container pattern cannot deal with the types of errors that are not addressed by the Input Guard and the Output Guard patterns. In other words, this pattern does not deal with internal errors of a component (e.g. changes to the internal state due to electromagnetic disturbances in the environment where the system operates). Moreover, this pattern does not address cases where the input or the output of a component conforms to the component specification but still contains errors according to the specification of the system in which the component operates (e.g. errors due to design faults). For example, the Fault Container pattern cannot provide fault containment when applied either to GETIN or to T2N components in Figure 66.

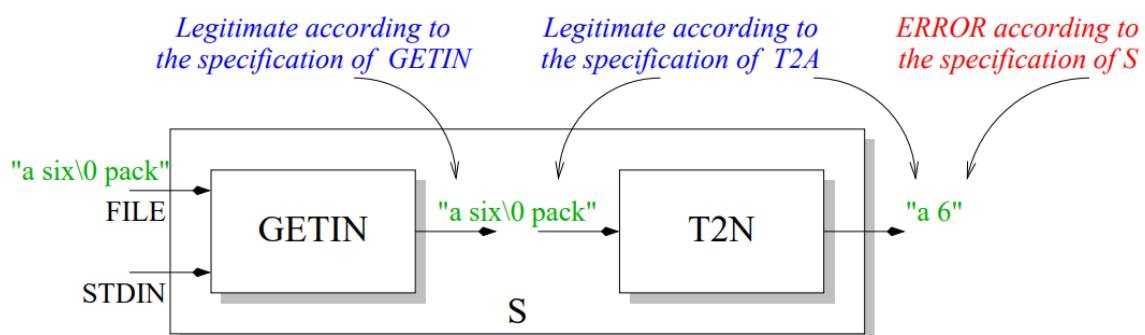


Figure 66: Example of error that may occur to system  $S$  and its constituent components.

## 35.4. Problem

In the above context, the Fault Container pattern solves the problem of prohibiting the propagation of an error both from inside a component to the rest of the system and from the component's environment into the component itself, by balancing the following forces:

- A system may be composed of components that are developed independently from each other, without keeping strict consistency with each other's specifications.
- A system may infect with errors its environment or be infected with errors from its environment despite the fact that the exchanged data that propagate the error is not perceived as erroneous by the system's environment.
- Different systems have different requirements regarding size impact of the fault containment mechanism.
- Different systems have different requirements regarding the time penalty of the fault containment mechanism.
- Fault containment is usually integrated with other solutions provided for other fault tolerance constituents (e.g. error masking, error detection, fault diagnosis) in order to provide wider fault tolerance guarantees.
- The system to which the Fault Container pattern is applied must be indistinguishable from the mechanism that implements the Fault Container pattern, i.e. it must not be possible to address or access system without addressing or accessing that mechanism at the same time.

## 35.5. Solution

As stated by its name, the Fault Container pattern provides a container that embraces the system on which the pattern is applied. This container forms the fence that stops the propagation of errors in both ways, from inside the container to the outside and vice versa. Whenever the system receives input from its environment or produces output to be delivered to its environment, the container checks it against the specification of the system. If and only if the input (or the output) confirms with the system specification, then it is allowed to reach the system (or reach the system's environment respectively).

A similarity of this pattern with the other two patterns presented previously in this paper is the fact that the behavior of the container is not defined in the presence of an error, besides the fact that it does not allow it to enter or leave the system. This behavior is intentionally left undefined in order to allow implementations of the Fault Container pattern to be combined with error detection, fault diagnosis and system repair mechanisms. It is worth mentioning that in practice there has not been any pure implementation of the Fault Container pattern as described here in the fault tolerance literature. Rather, very often the functionality of this pattern is combined with other functionalities. For example, the container functionality is combined with error detection and fault repair through reflection functionalities to result in a mechanism that provides fault containment and repair for the Chorus OS [1]. In another case, the container functionality has been combined with the functionality described by the Adapter pattern [2] in order to result in a fault containing Hive cell made out of four (or more) modified Unix kernels for shared memory multiprocessor architectures [3].

Strictly speaking, from the fault containment perspective the Fault Container pattern provides the same benefits as the combination of the Input Guard and the Output Guard patterns, i.e. it

prevents an error from being propagated inside and outside a given component. From a conceptual perspective however, this pattern contains more than the mere aggregation of the guarding of the access and exit points of a component. It also contains the potential of coordinating these two functionalities, i.e. the possibility to correlate an error in the output of the component to the input that, when processed by the component, led to the erroneous output. This additional functionality becomes visible when combining the Fault Container pattern with some fault diagnosis, error detection, fault repair, or system recovery mechanism. In practical terms, an implementation of the Fault Container pattern will provide to the environment of the contained component an interface similar to the aggregation of the interfaces provided by implementations of the Input Guard and Output Guard patterns. However, behind this interface and in addition to the functionality of the guards corresponding to the aforementioned patterns, the functionality of their coordination may be provided.

### 35.6. Structure

The Fault Container pattern introduces two entities, similar to those of the Input Guard and Output Guard patterns:

- The contained component, which is the part of the system that is shielded from getting contaminated by errors propagated from its environment through its input and from contaminating its environment with errors propagated through its output.
- The container, which is responsible to check against the contained component specification for errors in the input the component receives and in the output the component produces. In addition, the container may correlate the output checks with the checks of the input that led to that output in order to derive causal relations between input and erroneous output.

In practice, the container is a new component in the system which embeds the contained component and also implements the error fencing functionality. That makes the container and the contained component a single addressable constituent in the system composition. Figure 67a illustrates graphically the structure of the Fault Container pattern for a contained component with a single access and a single exit point. Figure 67b contains the activity diagram that describes the functionality of the container.

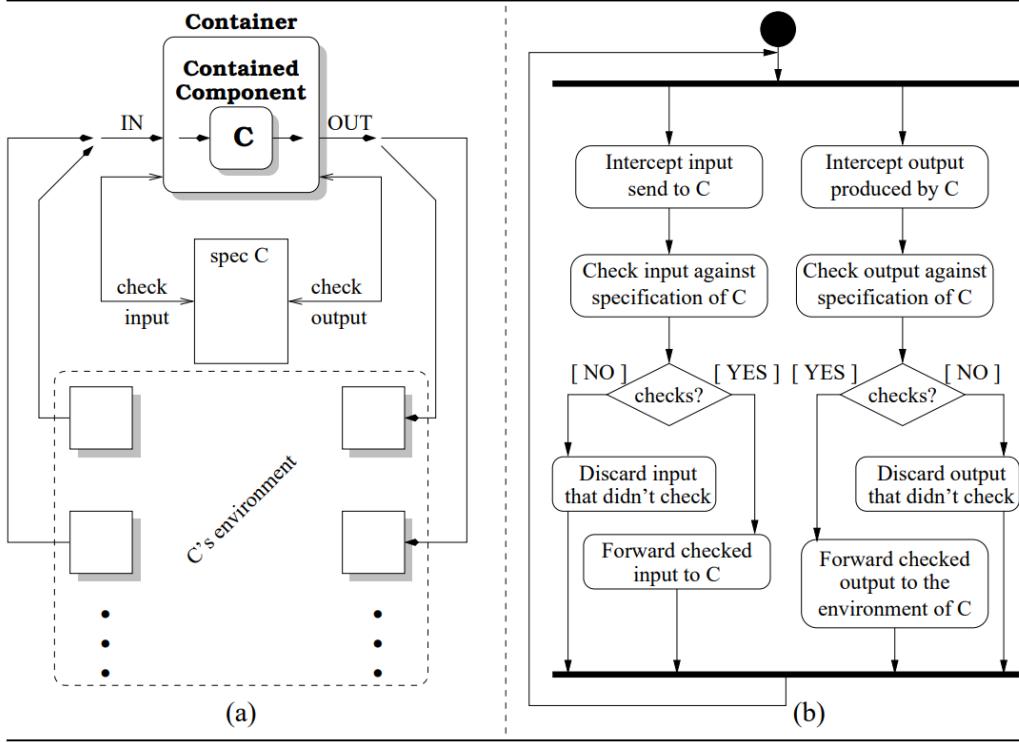


Figure 67: The structure (a) and the activity diagram (b) of the Fault Container pattern.

The implementation approach of the Fault Container pattern that is most widely used is the tight coupling of the container with the contained component. This approach is a straightforward application of the solution described above and it has appeared in two variants. In the first variant, usually applied to custom-made software, the container functionality is integrated with the contained component. Every time the latter receives some input or is about to deliver some output, the container functionality is engaged to check that input or output against the contained component specification. In the second variant, usually applied to systems composed from black-box COTS software, the container is a new component that embeds the contained component. Every input sends to the contained component and every output produced by it are intercepted by the container, which checks them against the specification of the contained component. Both variants of this implementation approach are applicable in many cases of software system development, ranging the composition of the system from custom-made software component to the use of black-box COTS software. As long as the specification of the contained component is available, this implementation approach can be applied to transform a software component into its fault containing counterpart. The main characteristic of this approach is that the container introduces a fixed time and space overhead. In cases where the input and output of a component is already checked by other means, the error fencing functionality provided by the container is redundant. Consequently, the system performance is unnecessarily penalized in terms on time and space overhead.

Another implementation alternative is to integrate the container with the environment of the contained component. This results in redundant instances of the container inside the components interacting with the contained component, which introduce a higher space overhead to the system compared to the first implementation alternative. Moreover, with this approach the contained component is entirely shielded from its environment but only shielded from those parts of its environment that contain the container. At the same time however, this

approach allows to integrate the container only with selected component from the environment of the contained component. Consequently, the system developer has better possibilities for tuning the time overhead introduced by the Fault Container pattern since the container functionality will not be engaged in every single interaction of the contained component with its environment.

This approach has an important drawback: it allows the contained component to be addressed separately from the container. This contradicts the second force of this pattern which requires the contained component to be indistinguishable from its container and results in a partially wrapped component. Nevertheless, this approach can be favored in practice when performance issues are of primary importance. The choice and the responsibility of partially wrapping a component in order to gain performance benefits lies entirely with the system designer. Another drawback of this approach is the additional communication overhead that is introduced by the coordination messages that need to be exchanged among the distributed instances of the container in order to deduce the causal relations between input and erroneous output of the contained component. Some of this overhead can be amortized by integrating these messages with other coordination messages for fault tolerance specific purposes (e.g. with acknowledge messages sent for error detection purposes).

In theory, a third implementation alternative is to place the container in a separate component which acts as proxy for the contained component. However, this approach is very similar to the second variant of the first implementation alternative, and there are no application cases where the first approach does not apply and this one does. On the other hand, the implementation of the container as a separate component bears two drawbacks: it has an elevated space overhead due to the additional component implementing the container, and it also has an elevated time overhead due to the indirection through the component implementing the container of every interaction of the contained component with its environment. For these reasons, this implementation approach has very low practical interest.

### **35.7. Dynamics**

### **35.8. Implementation**

### **35.9. Example Resolved**

### **35.10. Consequences**

The Fault Container pattern has the following benefits:

- It stops of errors expressed as input and output content or timing that does not conform to a component specification from entering or exiting that component.

- The undefined behavior of the container in the presence of errors allows its combination with error detection and error masking patterns (e.g. see [1] and [3]). Whenever this is applicable, the system benefits in terms of reduced run-time overhead introduced by the implementation of the fault tolerant mechanism (e.g. the combination of fault containment and error detection in the context of system recovery from errors).
- The eminent similarities of this pattern with the Adapter pattern [2] allow the straightforward combination of the two patterns in the same implementation. Employing the Adapter pattern to adjust the interface of a component that is otherwise incompatible with the interface its environment expects from it may introduce faults in the system. The Fault Container pattern complements the Adapter by ensuring that the adaptation of the component interface to the interface expected by its environment does not cause erroneous input and output to propagate from the component to its environment and vice versa.
- The different implementation alternatives allow the system developer to choose the best way to apply the Fault Container pattern in a given system. By embedding the contained component inside the container, the former is completely shielded from its environment for a fixed space and time overhead. When appropriate however, the system developer may choose to integrate the container with selected parts of the contained component's environment. This decreases the time overhead introduced by the Fault Container pattern for the price of a higher space overhead (redundant instances of the container) and for lower shielding guarantees since some selected interactions of the contained component will not be checked by the container.

The Fault Container pattern also imposes some liabilities:

- It is not possible to minimize both the time and the space overhead of this pattern. To keep low the time overhead introduced by the Fault Container pattern, the functionality of the container can be integrated with selected components in the environment of the contained component. In addition to reducing the fault containment guarantees only to those selected parts of the system, this approach also increases the space overhead because of the redundant instances of the container. On the other hand, to keep low the space overhead introduced by the Fault Container pattern one container can be used, which will embed the contained component. Then, every interaction between the contained component and its environment will have to go through the container, causing an elevated and fixed time penalty for the system execution.
- For certain systems that require the container to embed the contained component (e.g. systems composed from black-box COTS software), the Fault Container pattern results in an elevated implementation overhead. This is because the system developer has to implement the code necessary for embedding one component into another, in addition to the implementation of the container functionality.
- The Fault Container pattern cannot prevent the propagation of errors that do conform with the specification of the contained component as is illustrated in Figure 64 and 67. Such errors are not due to a malfunction of the contained component. Despite the fact that these errors do not have their source in the contained component which is checked to receive input and to produce output according to its specification, they can cause a failure on the rest of the system. Such a failure of the entire system will be traced back to an error detected in the contained component. As a result, recovery

actions targeted at the guarded component will probably be taken. However, such recovery actions do not deal with the source of the problem, which is the fault that caused the initial error (e.g. a design fault or an error in the input of the guarded component). To allow effective system recovery, sophisticated (and thus space and time consuming) error detection and fault diagnosis techniques must be employed which will allow the error tracing to be continued through the guarded component until the real source of the propagated error is revealed.

- The Fault Container pattern can effectively protect a component from being contaminated by erroneous input according to its specification. It also prevents the component from delivering to its environment erroneous output according to the component specification. However, unless it is combined with some error detection and system recovery mechanisms, this pattern will result in send- or receive-omission failures (i.e. failure to send output or receive input) of the contained component. For certain systems, such a failure of one of their components may cause a failure on the entire system. Hence, the Fault Container pattern has limited applicability to such systems if it is not combined with other fault tolerance patterns.

### 35.11. Known Uses

The Fault Container pattern has been applied in practice in the implementation of fault containing Hive cell made out of four (or more) modified Unix kernels for shared memory multiprocessor architectures [3]. Other cases where the Fault Container pattern has been used to shield the propagation of errors to and from an operating system kernel are the cases of fault-tolerant Mach [4] and the Chorus kernel [1].

### 35.12. See Also

The Fault Container pattern has obvious similarities with both the Input Guard and Output Guard patterns. Also, the Fault Container pattern complements the Adapter pattern [2] with fault containing properties. Finally, this pattern can be seen as the customization of the Proxy pattern [2] to meet the fault containment requirements of a system.

### 35.13. References

- [1] Salles, F., Rodriguez, M., Fabre, J. C., & Arlat, J. (1999, June). Metakernels and fault containment wrappers. In *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No. 99CB36352)* (pp. 22-29). IEEE.
- [2] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [3] Rosenblum, M., Chapin, J., Teodosiu, D., Devine, S., Lahiri, T., & Gupta, A. (1996). Implementing efficient fault containment for multiprocessors: confining faults in a shared-memory multiprocessor environment. *Communications of the ACM*, 39(9), 52-61.
- [4] Russinovich, M., Segall, Z., & Siewiorek, D. (1994). Application transparent fault management in Fault Tolerant Mach. In *Foundations of Dependable Computing* (pp. 215-241). Springer, Boston, MA.

### **35.14. Source**

Saridakis, T. (2003, June). Design Patterns for Fault Containment. In *EuroPLoP* (pp. 493-520).

# 36. Actor-Based Access Rights

## 36.1. Intent

This pattern allows us to depict and classify all business actors that act on an information system. By classifying we get a better hold on the completeness of the set of actors and therefore we get a better hold on the access rights granted to these actors. As a result, we get a better hold on authorization.

## 36.2. Example

In the case of an Internet banking environment (see Known Uses) the information system X consisted of four subsystems. The grouping of actors proved very useful in determining the set of actors and the accompanying access rights.

Within the group of consumers, the following actors were determined:

- Visitor, an unknown person who just visits the site, retrieving public information.
- Prospect, a known person who took an application form by filling in some general information, but whose application form is not yet accepted.
- Customer, a known person who already became a customer by filling out an application form that was accepted by the banking organization.

Within the group of employees three groups were determined:

- Regular business actors, a group that was further decomposed later in the project and consisted of 15 to 20 different business actors.
- Governance organization actors, a group that was further decomposed later in the project and consisted of approximately 10 different actors, including actors from outsourcing companies.
- Security organization actors, another group that was further decomposed; this group included security managers from outsourcing companies.

Within the group of the business partners the most important actors were the ones from the line of control and the ones to deliver reports to.

Within the group of business partners, the most important actors were credit check organizations and credit card organizations.



Figure 68: Example of Actor Groups.

### 36.3. Context

This pattern can be applicable in heterogeneous environments where multiple types of users need access to systems or information. Typically, this could be a business application with multiple channels to their customers, consumers, business partners or others.

### 36.4. Problem

In everyday life we do not have a complete overview of the set of actors that participate in an information system. As a result, it is difficult or impossible to get hold of the access rights granted. A classification of all actors can help us to improve the completeness of the set and to improve the control on their access rights.

Heavily stimulated by the almost endless possibilities of the Internet, the exchange of data between organizations and between organizations and their customers increased enormously in recent years. All kinds of new systems were introduced in the so-called “e-business”. At the other side of the coin a lot of new people got access to these new systems and, as a result of this, got access to information.

In the rush for new business and new customers organizations tend to forget about restricting the access rights. As a matter of fact, they often do not even know who have access to their systems! Besides that, the set of new business and new customers is very volatile, since new systems come and go within months.

In the meantime, the demand for more powerful systems with a high granularity in functions of the systems increases. So, authorization for different customers has to be granular as well. A lot of organizations are currently shifting to a “need to protect” security strategy.

So, there's the problem. How do we control granular access rights when we do not have an overview of all people who, all with their own characteristics, have access to our systems?

The following forces apply:

- Organizations left the fortress mentality and open up for actors like customers or business partners. These actors demand and get access to information related to them. Organizations delegate responsibilities to the actors but remain responsible for granting the access rights.
- Granting the wrong access rights to the wrong actors might lead to unauthorized access with consequences in fraud, privacy violation or image damage.
- The population of actors is highly dynamic, due to new business partners, acquisitions and mergers, new customers and in or outsourcing partners.
- The changes in access rights are highly dynamic as well, due to continuously changing functionality of information systems due to changing business requirements.
- Standard available role-based access in information systems increases. However, these roles do not always match the roles within the organization. In order to control access rights, organizations have to be able to match the standard available roles.

## 36.5. Solution

In order to control a highly dynamic environment we must look at the elements that remain stable. Stable elements that allow organizations to provide products or services.

1. Business processes that have to be performed to provide products or services
2. Information accompanying these processes

Anyone or anything that acts on the information within a business process is a business actor. Access rights for actors only depend on the actions they perform on information; they do not depend on things like the location of these actors. To be clear: the security controls that have to be in place to enforce these access rights of course do depend on location, or specific channel, etc.

To some extend the types of actors in an organization are independent of the business processes or information involved. We distinguish the following actors.

Group 1. Employees.

In every business process employees of the organization will be actor. But not all employees will be actor. Therefore, we need higher granularity within this group. A first cut.

- Regular Business Employees. These actors can perform their actions in front or back office. In most situations we will divide this group even further.
- Governance actors. Every system has to be governed and these actors do influence the way a business process operates. In most situations we will divide this group even further.
- Security actors. Within each organization these actors influence the way a business process operates by granting access rights, etc. In most situations we will divide this group even further.

Group 2. Consumers.

This group is optional, since not all organizations will allow their consumers to act direct on the business processes and information. For instance, if consumers do act via (snail) mail or call center, they act indirect not direct on information.

Usually, we need higher granularity within this group. A first cut.

- Visitor. Typically, this would be a visitor to a web site who browses through the site and leaves without ordering anything or without requesting information.
- Prospect. Typically, this would be someone who request information and therefore has to leave some personal information.
- Consumer. Typically, this would be someone who orders products or services once.
- Customers. Typically, this would be someone with some order history.

Group 3. Business partners.

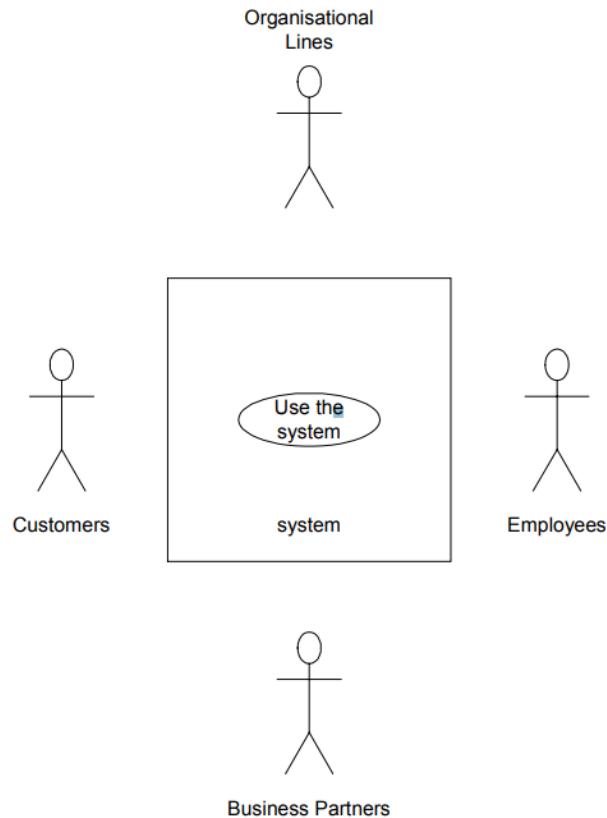
This group is optional, although that's not likely. Most organizations do have automated interaction with business partners. Other examples might be regulators and governmental agencies. The trend is that organizations will interact with their "volatile eco-systems" more intense than in the past. A typical first cut is based on actors in front office, mid office, or back office.

- Front office, e.g. content providers, information and brokerage portals
- Mid office. Typically, these business partners provide services like credit check, criminal record check, etc.
- Back office. Typically, these business partners provide financial services and transactions.

Group 4. Organizational lines of control.

This group is optional. Depending on the scope of the set of business processes, these organizational lines of control could be within the organization or to a mother or daughter organization. A typical first cut.

- Controlling actors.
- Reporting actors.



*Figure 69: Groups of Actors*

## 36.6. Structure

## 36.7. Dynamics

## 36.8. Implementation

## 36.9. Example Resolved

## 36.10. Consequences

Consequences in implementing the pattern might be the following.

Some benefits:

- A proven decomposition of actors can save time in determining the right access rights for the right actors.
- The grouping of actors will provide overview in an environment with a lot of very different actors, such as in web-enabled systems.
- The structured decomposition of the complete set of actors guarantees that easy to forget actors, like security administrators, will not be overlooked.
- A quick actor overview helps to scope projects and identify window policies, contracts, and service level agreements (SLA's)
- The need for security administration procedures / tools can be quickly identified.
- The pattern can be used as reference for security measures or to scope risk analysis studies
- The pattern is easy to visualize and can be easily used to support further analysis and design

Some pitfalls:

- The decomposition of the actors in groups can be an enormous job in a large environment. Remember that having a complete decomposition is not the goal in itself, but it is merely a means to control access rights.
- It is tempting to put all actors together and to call them just “users”. This might save time and a lot of discussion with businesspeople or business units, but it will not lead to a sound role-based access scheme, where the access rights can be explained in business terminology.
- Not all classes are applicable in all situations.
- Especially the class of business partners cannot be enumerated; there is an endless number of these business partners. A sub classification can be applied.

## 36.11. Known Uses

The authors used this pattern to establish access rights in an Internet bank. Another use of this pattern is for an extranet security architecture for a high-tech company.

## 36.12. See Also

In general, all patterns regarding role-based access are related, especially:

- The Actor and role life-cycle pattern.
- This pattern provides input for the Role pattern [1].
- Determining the variety of users helps to assign rights to users according to their roles in an institution [2].
- This pattern helps to identify user responsibilities within the organization in order to establish Role Based Access Control.

## 36.13. References

[1] Yoder, J., & Barcalow, J. (1997, September). Architectural patterns for enabling application security. In *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97)* (Vol. 2, p. 30).

- [2] Fernandez, E. B., & Pan, R. (2001, September). A pattern language for security models. In *In Proc. of PLoP* (Vol. 1).

### **36.14. Source**

Hofman, A., & Elsinga, B. (2002). Control the Actor-Based Access Rights. In *EuroPLoP* (pp. 233-244).

# 37. Actor and Role Lifecycle

## 37.1. Intent

The Actor and Role Lifecycle Pattern depict the most elementary steps, processes, and actions to be established in order to govern the authorization rights. The pattern is based on the life cycle of an actor.

## 37.2. Example

Suzan is currently working for a high-tech company for two years. So, her employee life cycle for this company did start two years ago. From the start her role is secretary of the CEO. On the first of April, Suzan will move to the administrative department where she will be in charge of the invoicing team.

What does this mean for her actor and role life cycles?

Looking at the current role of Suzan, she is an actor in the electronic agenda information system with the role “Secretary of CEO” which gives her the authority to update the agenda of the CEO. In the new role of Suzan in the invoicing team, Suzan needs to be able to work as an actor in the ERP information system with the role “team leader invoicing”.

If we apply the actor and role life cycle pattern to this example, then the pattern will tell us that two security procedures need to be triggered just before the first of April:

- Deregister role “Secretary of CEO”. E.g. Suzan will still be an actor in the electronic agenda information system because she needs to be able to access her own agenda.
- Register actor in the ERP system. E.g. before Suzan her role can be entered into the ERP system she must first be an actor within this system.
- Register role “team leader invoicing” for Suzan in the ERP - system

## 37.3. Context

This pattern is applicable in almost every environment where access rights have to be maintained. It can be used in designing the processes, but also as a control or audit tool.

## 37.4. Problem

The processes necessary to govern the access rights to information systems are hardly ever explicitly established, although they are quite standard. Based on the actors of a system and their life cycle we can easily depict the necessary processes. By using this pattern, we do not only guarantee the establishment of the granting of the access rights, but also on the withdrawal of these rights.

Apart from the problems in establishing the right access rights to the right people, the right actors (see the other patterns in this document), one of the well-known problems is also the management of the access rights. Therefore, we need to establish some processes.

The following forces apply:

- Employees, or people in general, strive to a maximum set of access rights. Having access to is associated with having power. Power to do something, power to see information. Aiming at a maximum set of access rights is not in line with security where we normally grant access rights based on the “need to know” principle. This hampers controlling access rights.
- When people change roles within an organization, this is usually associated with promotion. In general, promotions are associated with more power, better income, and more rights. It’s not appreciated when promotions result in limiting access rights, especially when people were used to having these access rights in their previous role.
- In order to do their job, people need access to information and computer systems. They will complain if they can’t do their job because of lacking access rights. They will hardly ever complain if they have too many access rights for doing their job. Sometimes they won’t even notice having too many rights, sometimes they think it might be convenient and sometimes they deliberately won’t mention it.
- Access rights are usually granted when a person starts a new job or role. Due to small changes in job roles, the job to be performed or small changes in the information systems, in time the granted access rights may delineate from the necessary rights, but nobody notices.

These forces all plead for having structured processes in place to grant and withdraw access rights.

## 37.5. Solution

Starting from the actor life cycle we can determine the essential business processes related to the management of the access rights. These are not only based on granting access, but also on the withdrawal of the access rights. That’s the advantage of the life-cycle principle.

### 37.5.1. Role, actor, and employee life cycle

When a human (e.g. employee or student) joins an organization, he (or she) will leave the same organization in the end. This could be called the “employee life-cycle”. Since there is no obligation that every human also has to be an actor on any information system, the granting of access rights is no part of the employee life cycle. Therefore, we will not consider this life cycle in the remainder. When a human becomes an actor on any information system, the “Actor Life-cycle” starts. During this life cycle the actor can perform several roles, for which access rights have to be controlled. Within the actor life cycle there can be several roles to be performed, where each role can have its own access rights to be controlled. Sometimes an actor can change roles almost instantaneously; sometimes a role can take weeks or months.

According to his role, each actor can perform certain business processes. This is the elementary level where the actor is identified and authorized to perform the business processes. Figure 70 and 71 illustrates the above.

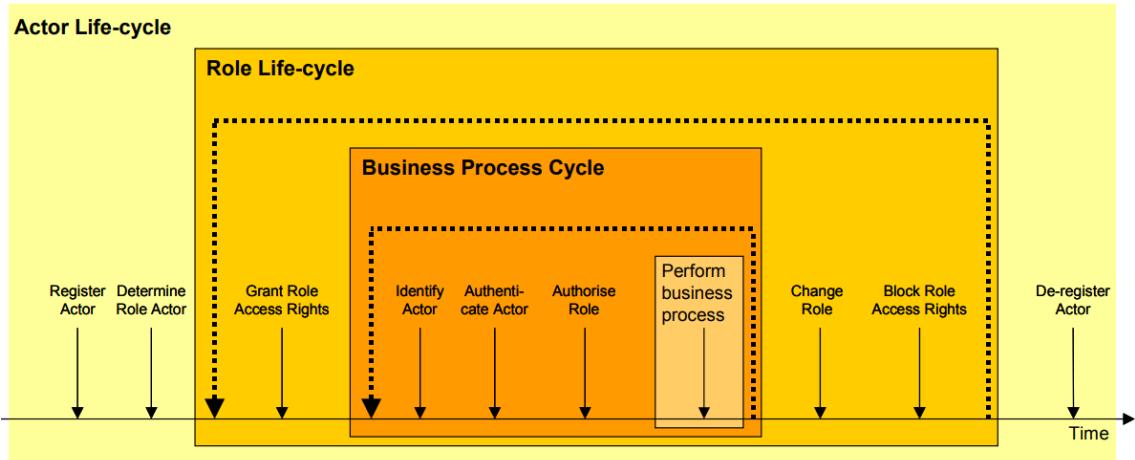


Figure 70: Processes necessary in controlling access rights.

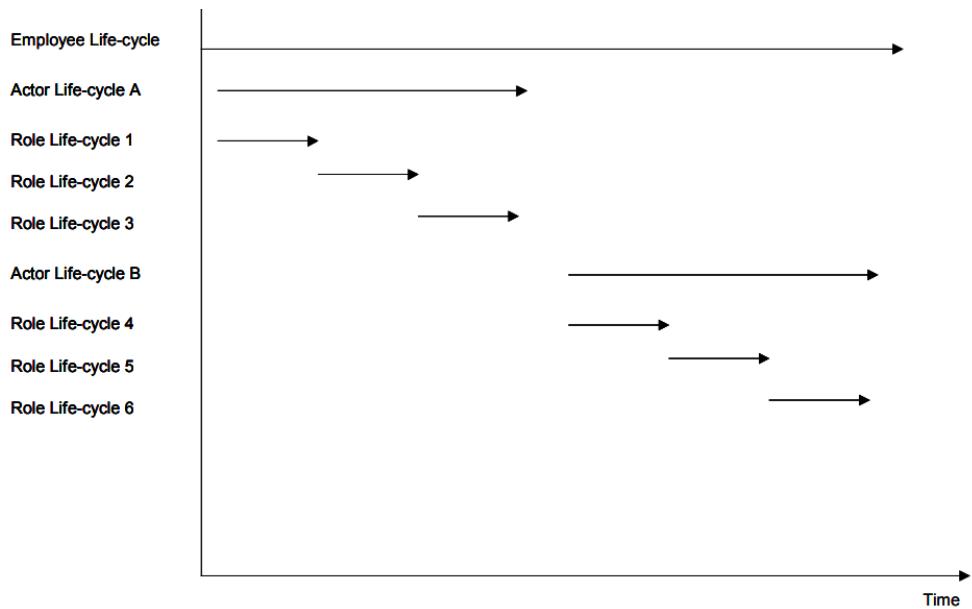


Figure 71: Role, Actor and Employee Lifecycle.

It's not easy to use an activity diagram since normally the activities follow each other without any delay. Since an employee life cycle or a role life cycle can take years, we are still searching for improvements or other diagramming techniques.

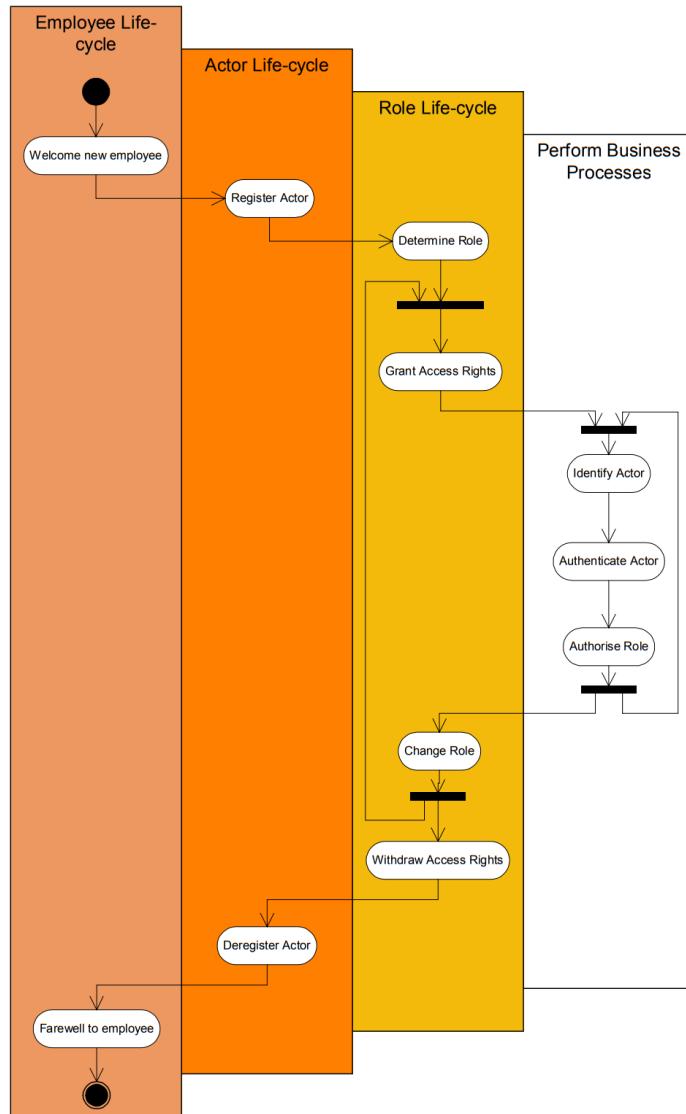


Figure 72: Life cycles in an Activity Diagramming technique

### 37.5.2. Authorization Processes during the life cycle

Starting with the actor life cycle and thinking about processes that have to be in place in order to control the access rights of the actor, we find the following.

1. Register Actor is the process to register the actors that will have access to a specific system. We do not specify how this registration is done, although automated registration is preferred. At least this process should register the combination of the identity of the employee, the actor he will be and the information system upon which he will act.
2. De-register Actor is the counterpart to registration. This process builds on the registration from the previous process and should register at least the date at which the employee was no longer associated with the specific actor. Preferably the process should also register the date of deregistration for auditing purposes.

During the role life cycle the following authorization processes have to be in place.

3. Determine role of actor
4. Grant access rights
5. Withdraw access rights
6. Change role

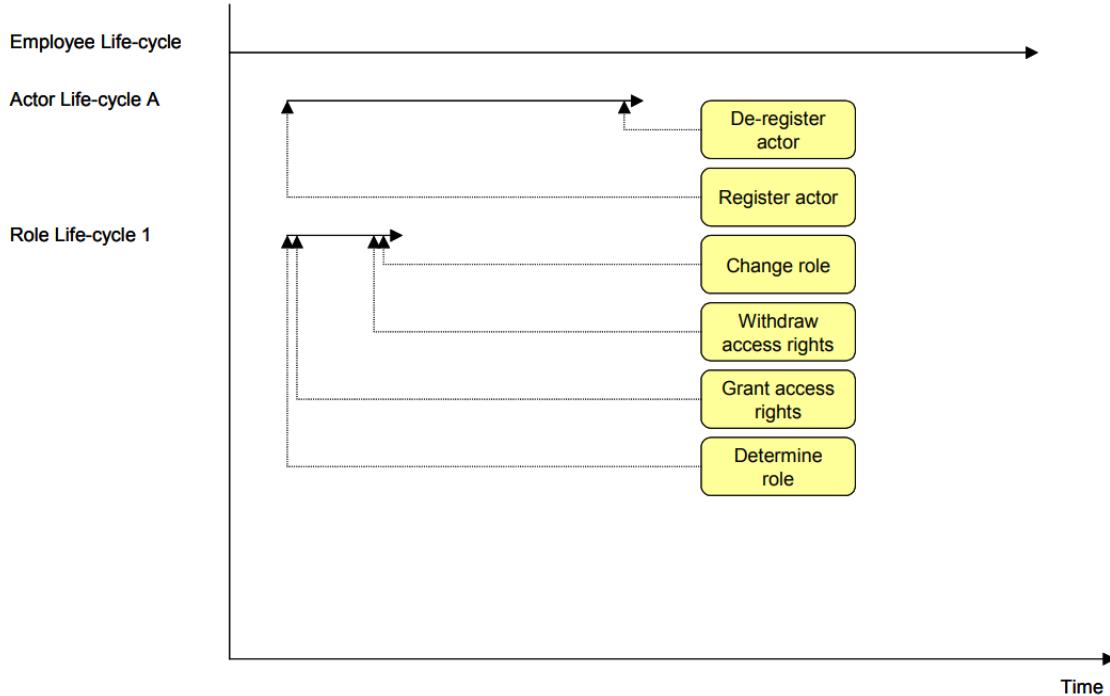


Figure 73: Authorization processes during life cycle.

Acting in his role, the actor will perform business processes. In order to control the access to these processes, identification, authentication, and authorization has to be in place. Since these are regular security controls, they are out of scope for this document.

## 37.6. Structure

## 37.7. Dynamics

## 37.8. Implementation

Implementation issues:

- To administrate access rights sometimes package-based solutions are used. Sometimes it is quite difficult to pinpoint the described processes in these package-based solutions.
- When auditing or reporting tools are used to detect incidents on the use of access rights, be aware that the distinction between employee, actor and role is not always clear or adjustable in these tools.
- Be aware that the implementation of this pattern might be influenced by the security policy and classification schemes.

- Make sure that the business owners and information owners check the authorizations on a regular basis.
- In highly volatile environments this pattern needs to be complemented with other security controls to keep a grip on the dynamics of the environment.

## **37.9. Example Resolved**

### **37.10. Consequences**

Consequences in implementing this pattern might be the following.

Some benefits:

- By paying explicit attention to easy to forget processes like the withdrawal of access rights or deregistration of actors, these processes will be implemented and therefore access rights are better controlled.
- The notion of life cycle implicitly makes clear that access rights are only granted for a certain period.
- The distinction between actor and role will give insight in the specific access rights needed for a certain role.

Some pitfalls:

- Although this pattern makes clear that these processes have to be in place, it is not possible to determine where and when these processes have to be in place. It can still be a tough job finding the right moment for withdrawal of access rights.
- In environments where actors seldom change role or where employees seldom change to another actor, implementing this pattern can lead to rigorous administration that will be outdated very soon.
- Cultural aspects like lack of discipline are known pitfalls to implement the procedures related to this pattern. Solutions: Automate as much as possible and produce special security reports to check the authorizations on a regular basis.

## **37.11. Known Uses**

This pattern was used by the authors in a previous project where security administration was developed and implemented in an Internet banking environment. Another use of this pattern is for an extranet security architecture for a high-tech company.

## **37.12. See Also**

In general, all patterns regarding role-based access are related, especially:

- The Actor based access rights pattern.
- The processes described in this pattern are applicable to the User-Role-Privilege relationship in Role [1].

- The activities related to managing the rights in Role Based Access Control are part of the processes described in this pattern.
- The activities related to managing the rights in Administrator Objects pattern are part of the processes described in this pattern.

### **37.13. References**

[1] Yoder, J., & Barcalow, J. (1997, September). Architectural patterns for enabling application security. In *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97)* (Vol. 2, p. 30).

### **37.14. Source**

Hofman, A., & Elsinga, B. (2002). Control the Actor-Based Access Rights. In *EuroPLoP* (pp. 233-244).

# 38. Policy Enforcement

## 38.1. Intent

Enforce different kinds of policy in a uniform way. The policies and their enforcement processes can be changed easily to accommodate future changes.

## 38.2. Example

Consider a license management system of software component, as shown in Figure 74, in which we want to make sure that rights for identified clients are correctly attributed and authorized, and the usage of protected software component is properly recorded. To realize access control, and usage tracking and recording, it is necessary that there be a variety of policies that specify both the clients and their corresponding license types; that is, what they are able to do within the licensing scheme, and what they cannot do (for example make unauthorized distributions of a software component). Whether a license request is allowed or denied is based on policies that may be pre-defined by the developer of the licensing system, or a third-party supplier, or may even be set up or changed by licensors at a later date, so long as this falls within the capabilities (i.e. legal restrictions) of that licensor. There may be occasions where this process is inherited by secondary licensors from primary licensors. In relation to the licensing of software, an area of high complexity and economic significance [1], such a process is a commonly recurring pattern within the legal system and in business. The type of license and related licensing activities vary from client to client and over time; thus, policies may need to change frequently. To maintain such licensing policies is difficult and resource intensive.

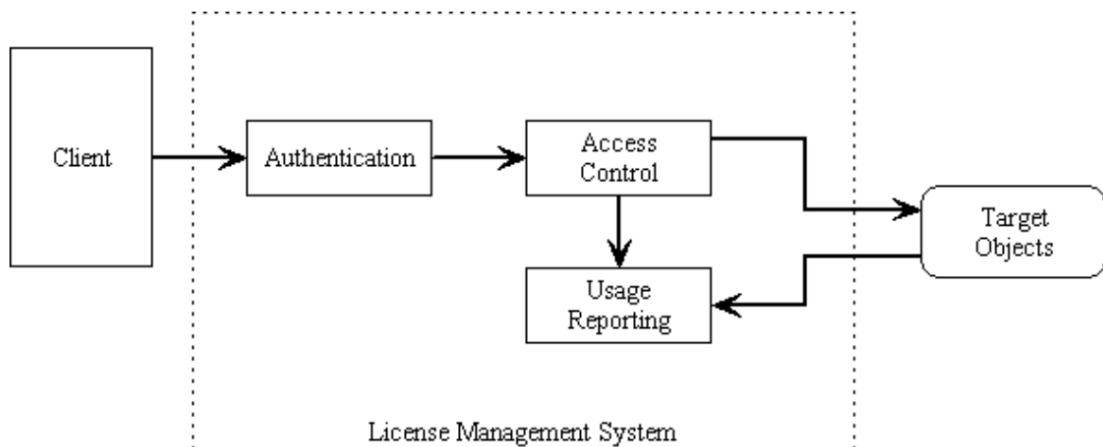


Figure 74: License Management System

A generalized and flexible architecture is required in which corresponding policies can be applied, so long as the policy elements are readily available in the rapidly changing policy environment.

### 38.3. Context

In many business environments, whether working with network management, quality of service (QoS) or within the mortgage and insurance sectors, we often have to face a policy pattern of the kind “When Who Can/Cannot Do What”. As shown in Figure 75, Privileges specify “When”, Subject specifies “Who”, Conditions specify “Can/Cannot”, and “Do What” is specified by Object and Actions.

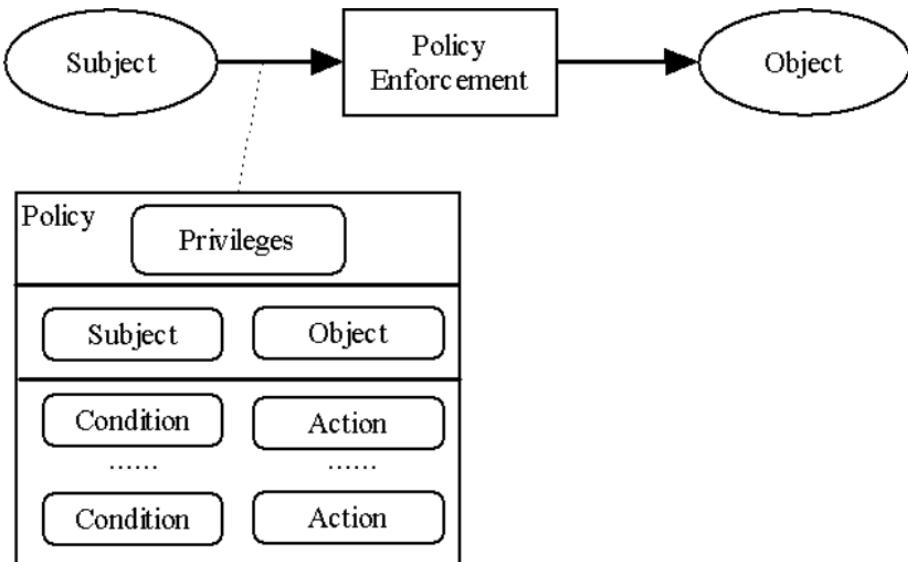


Figure 75: Policy Model

Business and management environments need flexibility and adaptability when implementing their policies. Although the target objects and their structures may be stable, the tasks, including the handling rules and procedures, are varied.

### 38.4. Problem

How to enforce policies no matter what kind of policies are required to be enforced? How to provide flexibility and adaptability to this policy enforcement, to accommodate future changes?

System environments are in a constant state of flux, whether looked at from a high level of abstraction in business or from a lower level, such as in component management. In the face of such changeable circumstances, there have been some approaches proposed that fulfil the purpose of representing the dynamic business rules and management policies adaptively and effectively, including Command Dispatcher Pattern [2], Rule Object [3], Ponder [4], Internet Engineering Task Force (IETF) Policy Core Information Model [5].

However, the Command Dispatcher pattern emphasizes a plug-in mechanism and run-time interpretation of a sequence of command executions written in plain text but does not consider organizing those commands into policies like “if condition then action else action”. The Rule Object Pattern, on the contrary, considers the language for policy/rule construction, but does not provide an easy way for building up policies/rules for people who are not familiar with any programming language. Ponder and PCIM take an implementation perspective and are mainly concerned with management of networks and QoS.

When the requirements and goals of network policies, management policies, and business policies are taken into consideration, the following forces need to be addressed in the context of policy enforcement:

- Integrating pre-built policy libraries: pre-built libraries are modules for a large number of policy elements that are used in a policy, such as privilege, condition, and action. The libraries are extensible, which means that the policy elements inside can be independently developed and delivered even when the system is in runtime.
- Flexible configuration: In a policy, there is the need for several collaborating elements to reach a specific purpose. Furthermore, a policy element may invoke other elements in order to complete its execution. In both situations, a number of options may be available for a certain operation to be completed by selecting from a range of policy elements.
- User-defined policy rules: The client should be able to implement and customize business rules easily.
- Central Policy Handling: Policies should be stored in a repository and managed by a central instance.
- Efficiency: Runtime implementation of customizable and dynamic systems often decreases the efficiency because runtime indirections, lookup mechanisms, and plain text parsing/interpretation place an overhead on the system.

## 38.5. Solution

The Policy Enforcement pattern (Figure 76) provides a uniform way to enforce a variety of business and management policies with flexibility and adaptability. With policy elements, policy processes, configuration policies and scripted policy rules, clients can customize their own policies and later register them in the repository that may be either centralized or distributed depending on the particular demands of the operation environment.

### Architecture Overview

As shown in Figure 76, clients request an operation on a target object. The Policy Factory creates the corresponding policy and related policy elements in the pool. Policy Decision decides whether or not the client's request is allowed. Policy Enforcement enforces the policy and executes the requested operation on the target object.

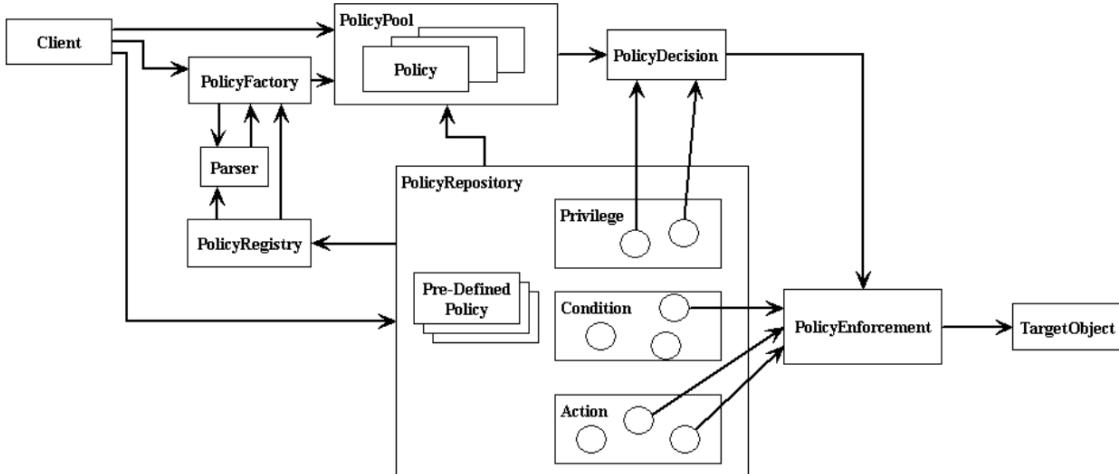


Figure 76: Policy Enforcement Pattern

## Participants

The following participants form the structure of the policy enforcement pattern:

- Common clients may request an execution on a targeted object. There may also be administrative clients to grant the privileges for clients, and to define and register the administrative policies. Some clients, who are developers, can define and register business rules as Policies, or they can develop and register policy modules into the Policy Repository.
- The Policy Factory creates runtime Policy instances and puts them into the Policy Pool.
- The Policy Pool provides efficient runtime storage for the policy instances. If a policy that a client request is already in the pool, it can be used directly without instantiating it again.
- A Parser parses the plain text format of policies, and then queries the policy element specifications from Policy Registry and in doing so helps create runtime Policy instances.
- The Policy Registry is a lightweight storage for specifications of policy element interface.
- Pre-Defined Policies are rules that can be applied to the clients' request. There are two kinds of policies, Configuration Policy, and Policy Rule.
- Policy Elements are implemented modules used in policies, including Privilege, Condition and Action.
- A Policy Decision decides whether or not a certain request is allowed.
- The Policy Enforcement component evaluates the required policy conditions and invokes the corresponding policy actions. Depending on the type of the policy, the enforcement process may vary.
- The Policy Repository provides storage for the pre-defined policies, and policy elements. It supplies the references of Policy Element to Policy Decision and Policy Enforcement and helps them both find the appropriate modules at runtime. The new functions and optimized elements can be integrated into it at a later date.
- The Target Object is the object that the client requests.

## 38.6. Structure

Figure 77 illustrates a class diagram of this Policy Enforcement pattern. There are two important interactions in this diagram: PolicyObject interacts with ConfigurationPolicy, and PolicyRuleInterpreter interacts with PolicyRule. The three kinds of PolicyElement, which are Privilege, Condition and Action, along with PolicyObject, ConfigurationPolicy and PolicyRule, can be defined, developed, and delivered independently.

The PolicyDecision assesses the privileges based on the client, the target object, and corresponding policies. It queries the pre-built policy decision modules, and provides the security functions, such as authentication, access control, non-repudiation, and software license control.

PolicyEnforcement enforces the policy in two ways, through invoking either PolicyObject or PolicyRuleInterpreter, which are related to ConfigurationPolicy or PolicyRule respectively.

- PolicyObject is pre-defined policy process that interacts with ConfigurationPolicy. Some popular and well-known rules in certain target domain may be pre-defined as policy processes. These policy processes can be integrated into the system and be invoked when necessary.
- The PolicyRuleInterpreter interprets PolicyRule at runtime. Both kinds of PolicyEnforcement evaluate the required policy conditions and invoke the corresponding policy actions.

PolicyElement includes Privilege, Condition and Action:

- Privilege is module for PolicyDecision to help determine, according to its policy, whether or not the client's request on the required target object is legal.
- Condition is used to decide how a given policy will be carried out on a certain step.
- Actions are the operations that will be executed depending on different situations.

PolicyFactory creates runtime Policy instances and put them into the Policy Pool. There are two kinds of policies, ConfigurationPolicy and PolicyRule:

- ConfigurationPolicy contains the configuration information in a plain text format, which is used by both PolicyDecision and PolicyObject to choose proper policy elements.
- PolicyRule is an executable script file also containing the configuration information. In the execution of PolicyRule, the script is interpreted and the configuration information inside it provides the proper option of module choice.

The PolicyRepository provides storage for these pre-defined policies (PolicyRule and ConfigurationPolicy), policy processes (PolicyObject), and policy elements (Privilege, Condition, and Action). It supplies the references of required policy element to PolicyDecision and PolicyEnforcement and helps both of them find the appropriate modules at runtime.

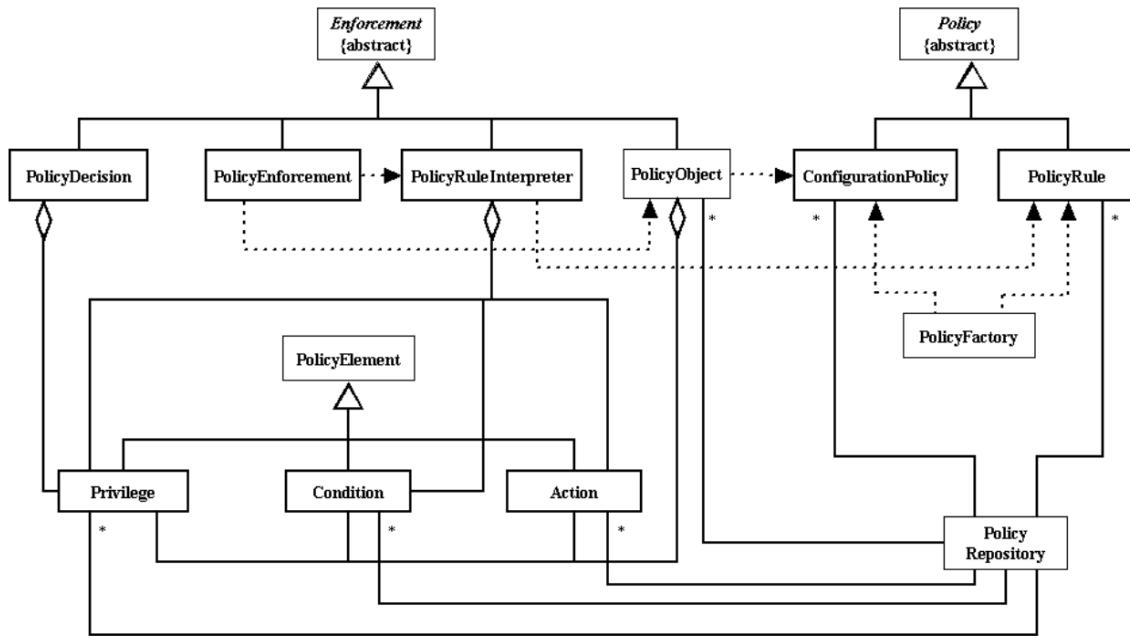


Figure 77: Class Diagram of Policy Enforcement Pattern

### 38.7. Dynamics

Figure 78 shows the collaboration when dealing with *ConfigurationPolicy* and *PolicyObject*.

- The *PolicyFactory* checks if the policy for current client is running. If found, it will obtain the references of policy instance and use them directly.
- If not found, *PolicyFactory* will create the correct *ConfigurationPolicy* instances. In this way, the run-time costs, such as speed and storage space, are reduced.
- The *PolicyDecision* uses the *ConfigurationPolicy* instances as the configuration commands to obtain the runtime references of the pre-built *Privilege* modules. It can use these modules to check the authorisation and access control information, and then make the final decision.
- If the decision is positive, the *PolicyEnforcement* will invoke *PolicyObject*. Depending on the configuration policy, the *PolicyObject* gets the runtime references of the pre-built *Condition* modules and *Action* modules; then it evaluates the Conditions and invokes the Actions directly.

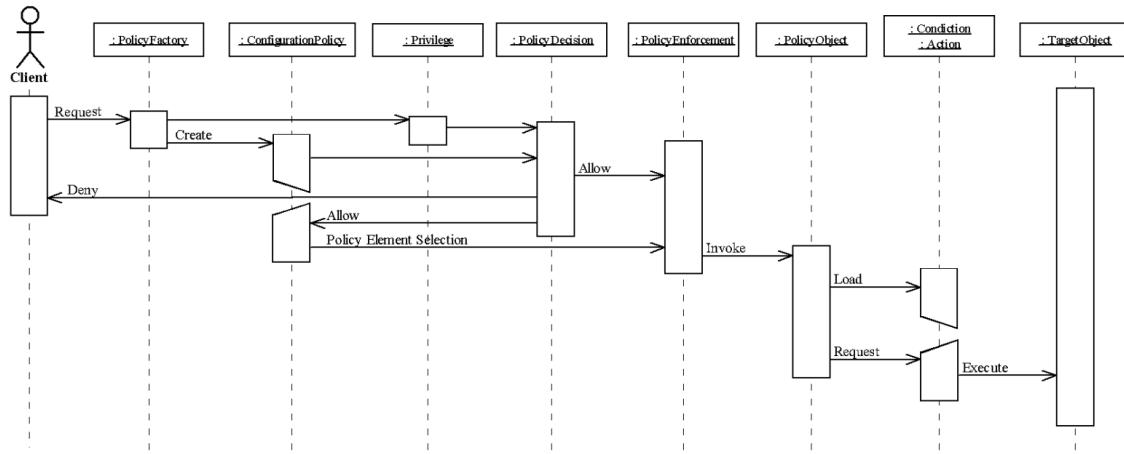


Figure 78: Collaboration for PolicyObject

When handling the scripted PolicyRule, as shown in Figure 79, the collaboration is slightly different from that for ConfigurationPolicy:

- The PolicyFactory will create the correct PolicyRule instances.
- The PolicyDecision uses the PolicyRule instances as the configuration commands to obtain the runtime references of the pre-built Privilege modules.
- If the result of Policy Decision is positive, Policy Enforcement will invoke PolicyRuleInterpreter that interprets and executes the policy rule. While interpreting, PolicyRuleInterpreter also evaluates the conditions and invokes the actions modules.

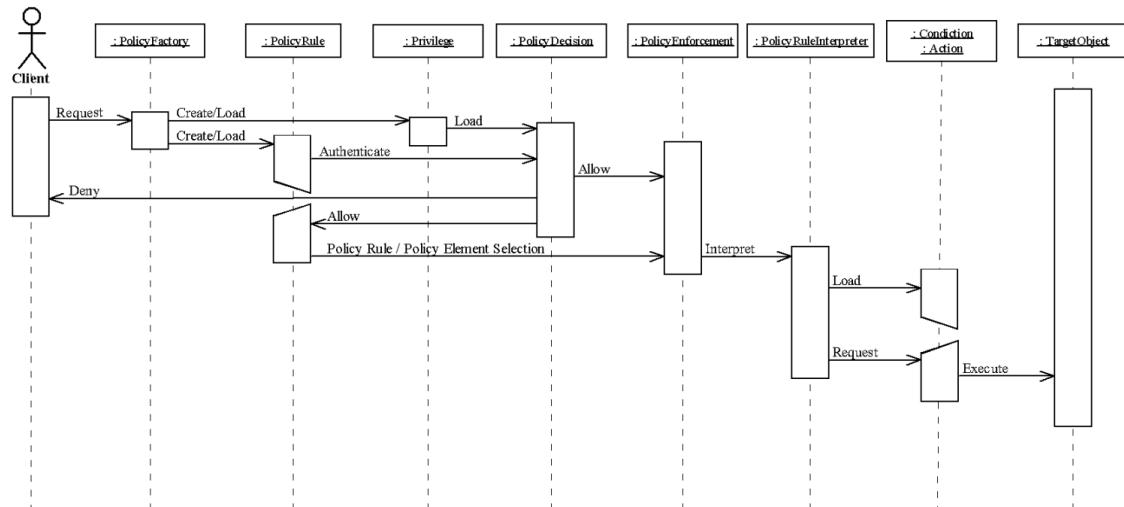
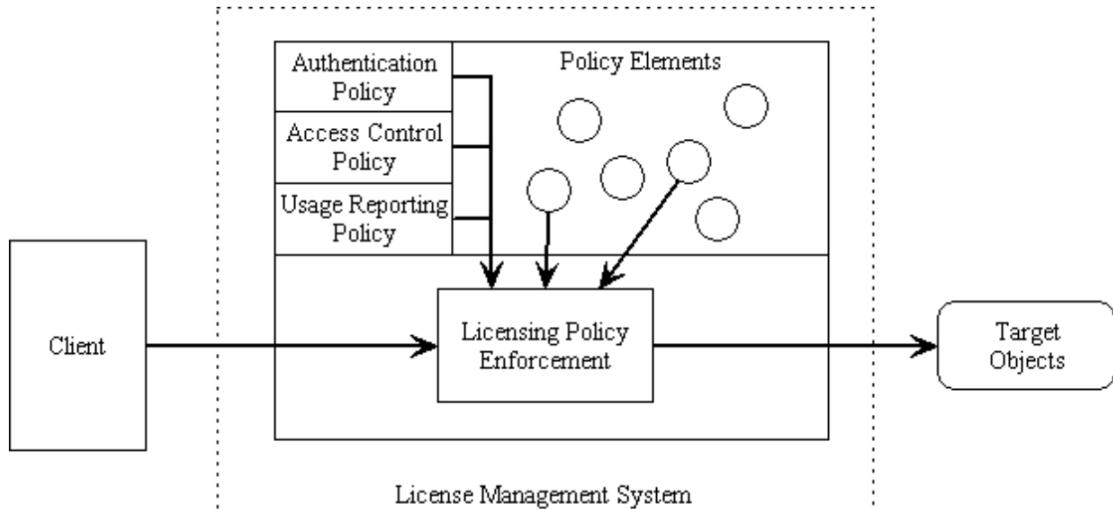


Figure 79: Collaboration for PolicyRule

## 38.8. Implementation

## 38.9. Example Resolved

Applying the Policy Enforcement pattern to a license management system, as shown in Figure 80, solves the problems we discussed in the example section.



*Figure 80: Example Resolved Using Policy Enforcement Pattern*

If the license type of a client changes, we need only to assign another policy related to the new license type to this client and delete the old one. The Policy Decision and Policy Enforcement of the new policy will execute when the client sends a request.

If some more optimized policy elements are developed according to the published common interfaces, they can be integrated into the licensing system due to the plug-in mechanism and be used right away without affecting existing policies and other policy elements.

If a rule inside a policy does not fit into a given situation and needs to be changed, the scripted policy rule provides a convenient way to generate new policies utilizing existing policy elements.

As noted above, the Policy Enforcement Pattern helps to easily maintain these varying licensing modules and policies and allows for future upgrades.

## 38.10. Consequences

### Benefits

- Flexibility: The application of this pattern allows end users to define their own policies to meet specific situations.
- Customizability: This pattern facilitates customization of policy enforcement process by providing configuration information in policies.
- Adaptability: The policy elements can be independently developed and delivered. This ensures that new functions and more optimized elements can always be integrated into the system.
- Universality: This pattern can be applied in a wide variety of environments where there are clients in a policy-controlled situation.

### Drawbacks

- Performance: This framework allows the plain text as a policy description, which needs to be parsed and interpreted at runtime. This will slow down the policy execution. The

Policy pool and pre-built modules provide partial solutions to this performance problem.

The Policy pool saves some time in policy instantiation. Pre-built modules provide time-optimized modules that can accelerate the execution time and improve performance dramatically.

- Higher Complexity: This pattern might increase the complexity of some target systems.
- Understandability: To implement the framework discussed in this study would not be trivial. It will require a relatively sound system design, which in turn will depend on detailed and thorough studies of the domain.

## 38.11. Variants

- Access Control Policy Enforcement: Policy Enforcement Pattern can be used to control the access to protected resources, such as in a membership-based digital library.
- Reporting Policy Enforcement: Policy Enforcement Pattern can be used to report the event that matches the pre-defined situation and to take corresponding actions, such as in a billing system.
- Filter Policy Enforcement: If we need to inspect a data flow, Policy Enforcement Pattern is able to check this flow and filter out the contents that are not allowed by policies.
- Quality of Service (QoS) Policy Enforcement: In QoS management, Policy Enforcement Pattern can be used to specify different types of service and to take corresponding actions.
- Network Management Policy Enforcement: When managing network, the rules of access control to network resource are specified as policy and are enforced using Policy Enforcement Pattern.
- Business Policy Enforcement: In a large enterprise, there can be many business policies involving all aspects of running that enterprise. To achieve an efficient management, Policy Enforcement Pattern is needed in the management software.

## 38.12. Known Uses

Some examples of the use of the Policy Pattern are:

- Clients apply policies to network services, such as Quality of Service (QoS). Policies for QoS help in regulating distinct data packages, and to provide the specific services described in the service level agreements (SLAs) or service level specifications (SLSs).
- The Internet Engineering Task Force (IETF) Policy Core Information Model [5] provides a good example of model for representing policy information and classes that could use the pattern described here. It can be a QoS variant of this pattern.
- Clients apply policy to network management. Policies for network management help to control the access to network resources and to avoid illegal use.
- The Ponder [4] and Policy Framework for Management of Distributed Systems [6] address the implementation of managing network systems based on policies. They constitute Network Management Policy Enforcement.

- Clients apply policy to software license management. License policies help to simplify the management of different types of licenses and to detect the violation of these licenses. The Access Control Policy Enforcement, Reporting Policy Enforcement, and Filter Policy Enforcement are used in our software component licensing system.
- The motivation for the OASIS (Open Architecture for Securely Interworking Services) [7] derives from a study undertaken with a view to providing ubiquitous access to Electronic Health Records (EHRs) held within the National Health Service in England. It adopts the idea of the policy to realize the role-based access control on the database and employs the access control variant of this pattern to express different local policies using the same process.
- The core of Component-based Micro-Workflow [8] is another known use of this pattern. The process component is similar to the pre-defined Policy. The execution component works as the Policy Enforcement. The synchronization component is similar to the Policy Decision.
- Adaptive Object Model [9] extracts the business rules from the implementation code and stores these rules separately. To achieve this, properties, and behaviors (called ‘Rules’) are separated from the entity. AOM suggests utilizing the Strategy Pattern [10] or Rule Object Pattern [3] to implement Rules. This could be realized using the Policy Enforcement Pattern.
- Clients apply policies to business environments. Policies are defined according to business rules. They help to make the change from “manual” routines to “automated” systems. This helps save time and money, as well as introducing consistency. The Objectiva telephone billing system [11] uses the Type Object [12] that is the basis of AOM [9]. It could be implemented by using the variants of Reporting Policy Enforcement and Business Policy Enforcement.

### **38.13. See Also**

The Command Dispatcher Pattern [2] is implicitly part of Policy Enforcement Pattern described here. The configuration policy and the scripted policy rule are enforced as the Command Dispatcher Pattern.

The Policy Enforcement Pattern contains the PolicyRuleInterpreter and the PolicyObject, which are similar with the Rule Object [3].

The Abstract Factory Pattern [10] is used in this Policy Enforcement Pattern to create the two kinds of policies.

The Command Pattern [10] separates the request for a service from its execution. It is similar to the process of Policy Enforcement in the Policy Enforcement Pattern, which also manages clients’ requests and schedules their execution.

The Strategy Pattern [10] defines a family of algorithms, encapsulates each, and makes them interchangeable. Strategy lets the algorithm vary independently from the client that uses it. It is similar to the use of flexible configuration in the Policy Enforcement Pattern, although only for simple situations and it does not offer scriptability.

Component Configurator Pattern [13] allows an application to link and unlink its component implementations at run-time without having to modify, recompile, or statically relink the

application. In the pattern described in this paper, the Policy Element, Enforcement and Policy are typical uses of Component Configurator Pattern.

Role Object Pattern [14] adapts an object to different clients' needs through transparently attached role objects, with each representing a role the object has to play in a particular client's context. The clients in this Policy Enforcement Pattern are also organized into different roles, such as common client, administrative client, developer and so on. Each role has its different policies, privileges, conditions, and actions.

### 38.14. References

- [1] Perry, M. (2001). Information Technology. Chapter 30 Lexis/Nexis Butterworths Electronic Business Law 2001
- [2] Dupire, B., & Fernandez, E. B. (2001). The command dispatcher pattern. In *8th Conference on Pattern Languages of Programs*.
- [3] Arsanjani, A. (2000). Rule Object Pattern Language, Proceedings of PLoP2000, Technical Report #wucs-00-29, Dept. of Computer Science, Washington University.
- [4] Damianou, N., Dulay, N., Lupu, E., & Sloman, M. (2001, January). The ponder policy specification language. In *International Workshop on Policies for Distributed Systems and Networks* (pp. 18-38). Springer, Berlin, Heidelberg.
- [5] Moore, B., Ellesson, E., Strassner, J., & Westerinen, A. (2001). Policy core information model—version 1 specification. *IETF RFC 3060*.
- [6] Damianou, N. C. (2002). *A policy framework for management of distributed systems* (Doctoral dissertation, University of London).
- [7] Bacon, J., Lloyd, M., & Moody, K. (2001, January). Translating role-based access control policy within context. In *International Workshop on Policies for Distributed Systems and Networks* (pp. 107-119). Springer, Berlin, Heidelberg.
- [8] Manolescu, D. A. (2003). An extensible workflow architecture with objects and patterns. In *Technology of Object-Oriented Languages, Systems and Architectures* (pp. 44-59). Springer, Boston, MA.
- [9] Yoder, J. W., Balaguer, F., & Johnson, R. (2001). Architecture and design of adaptive object-models. *ACM Sigplan Notices*, 36(12), 50-60.
- [10] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [11] Anderson, F., & Johnson, R. (1998, May). The Objectiva telephone billing system. In *MetaData Pattern Mining Workshop, Urbana, IL, May*.
- [12] Johnson, R., Wolf, B. (1998). The Type Object Pattern, *Pattern Languages of Program Design 3*, Addison Wesley.
- [13] Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000). *Pattern-oriented software architecture, patterns for concurrent and networked objects*. John Wiley & Sons.
- [14] Bäumer, D., Riehle, D., Siberski, W., & Wulf, M. (1998). The role object pattern. In *Washington University Dept. of Computer Science*.

### **38.15. Source**

Zhou, Y., Zhao, Q., & Perry, M. (2002, September). Policy enforcement pattern. In *Proceedings of the Conference on Pattern Languages of Programs* (pp. 1-14).

# **39. Administrator Object**

## **39.1. Intent**

## **39.2. Example**

## **39.3. Context**

In role-based access control environment, user-role assignment and role-privileges assignment are the major administrative tasks. Each user has a unique subject that describes the user's permitted roles and user must activate roles associated with his subject to access information. This pattern creates subjects for users and delegates administrative responsibilities.

## **39.4. Problem**

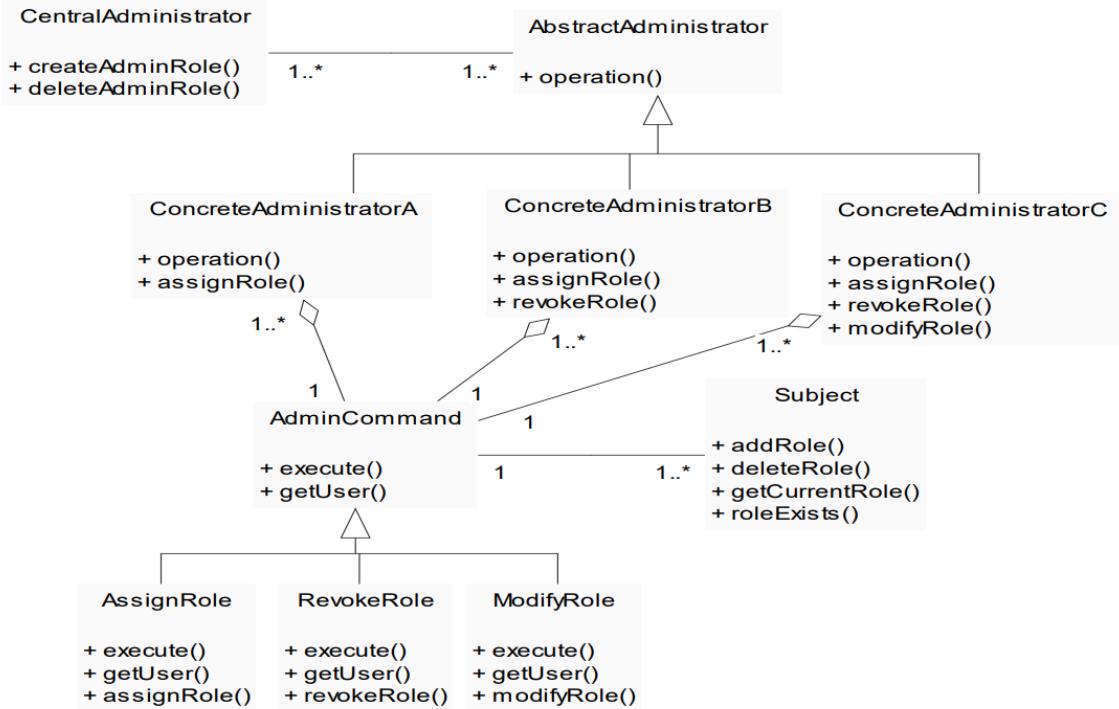
The question is: how can you manage user-role assignments and role-privileges assignments?

The following forces apply:

- A single user may allow playing multiple roles, but he may be restricted to play only one role at any given time.
- Roles may have overlapping access rights due to the overlapping responsibilities of job titles. In such a situation, one role (senior role) may have privileges to access all pieces of information authorized for one or more other roles (junior roles) but inverse is not true. A user playing a senior role must be allowed to play all associated junior roles. However, when the user is playing a particular junior role, he may or may not be allowed playing senior role at the same time.
- In general, senior roles have privileges to access all piece of information granted for its junior roles. However, sometimes some of the junior role's privileges can be private to that role. For example, in software development domain, senior roles may not be allowed to see programmer's incomplete codes.
- A group of users can play the same role, but some roles can only be taken by a limited number of users at any given time. For instant, there should be only one manager in a branch office of a bank.
- One or more administrators can centrally handle users and roles, but this is infeasible or inefficient for more complex applications such as distributed applications.

## **39.5. Solution**

Create a distinct set of roles (administrator roles) with different privileges to manage users and roles. Create a separate role (Central administrator) to manage those administrator roles. Each administrator is given options to handle user-role assignment based on their administrative privileges.



CentralAdministrator defines an interface to create, delete and modify ConcreteAdministrators.

AbstractAdministrator defines an interface for user-role assignment.

ConcreteAdministrators extend and implement the AbstractAdministrator interface based on authorized administrative privileges.

AdminCommand declares an interface for performing administrative actions.

AssignRole, RevokeRole and ModifyRole define bindings between Subject and administrative actions. Implements execute() to invoke the corresponding operation(s) on Subject.

Subject is a list of roles (role names and their status) that a particular user is permitted. Administrators create subjects for users by invoking Subject's interface.

ConcreteAdministrator issues a request to command objects to add, delete or modify roles. In turn, the ConcreteCommand (AssignRole, RevokeRole and ModifyRole) objects invoke the Subject to alter its content.

The CentralAdministrator handles the person-role assignment and role-privilege assignment for administrator roles

## 39.6. Structure

## 39.7. Dynamics

## **39.8. Implementation**

- Since a user may have multiple roles, the Subject class may be implemented to hold a list of roles and their status. Administrator can be implemented to add, revoke, and modify these roles.
- ConcreteAdministrator can be implemented with permitted operations. The methods assignRole(), revokeRole() and modifyRole() can be implemented to send requests to Admincommand object to assign, revoke or modify roles.
- Operation() method can be implemented to provide options to perform only permitted operations and to revoke method according to the selection.
- The Abstract Factory pattern might be used to create different types of Administrator objects.

## **39.9. Example Resolved**

- Administrator roles are designed with only operations permitted to them. Consequently, administrators are restricted to perform only permitted operations.
- The CentralAdministrator can assign administrator roles and privileges by creating ConcreteAdministrator roles.
- Role assignment and removal are the major operations performed by ConcreteAdministrators. In complex applications, administrators may be restricted to assign roles but not to remove or assign and remove roles to and from a particular set of users. Since each ConcreteAdministrator is given options to perform only permitted operations, this pattern prevents misuse of administrative privileges.
- Subject of a particular user can be chosen from a list of Subjects, and then would see a list of assigned roles, along with the information about those roles. Roles can easily be selected for deletion and modification.

## **39.10. Consequences**

## **39.11. Known Uses**

In OASIS, the administrators are responsible for certain security domains, they are allowed to access and manage the security information of these particular domains. The meta-administrators are responsible for specifying domains, defining security concepts. The meta-administrators and administrators are variants of the CentralAdministrator and ConcreteAdministrators of this pattern.

## **39.12. See Also**

- Administrator Manager components of OASIS [1] is a refinement of this pattern.
- This pattern uses the Command pattern to design administrative actions as objects.

### **39.13. References**

- [1] Fernandez, E. B., Larrondo-Petrie, M. M., & Gudes, E. (1994). A method-based authorization model for object-oriented databases. In *Security for Object-Oriented Systems* (pp. 135-150). Springer, London.

### **39.14. Source**

Kodituwakku, S. R., Bertok, P., & Zhao, L. (2001). APLRAC: A Pattern Language for Designing and Implementing Role-Based Access Control. In *EuroPLoP* (Vol. 1, p. 2001).

# 40. Role Validator

## 40.1. Intent

## 40.2. Example

## 40.3. Context

The user may access the system with a valid login name and password. Once the user login to the system, he or she is allowed to access information through roles. This pattern validates the user's permitted roles before letting him or her to perform any operation.

## 40.4. Problem

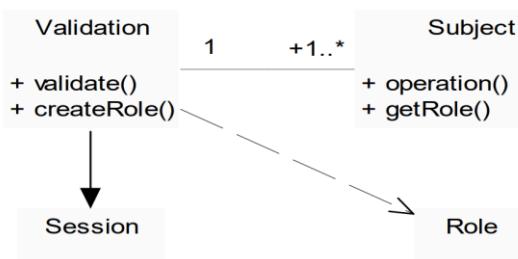
The different organizations have different security policies and there needs to be control of the access to the protected information objects. The question is: how can you ensure the validity of user roles and associated privileges to access the information objects?

The following forces apply:

- Administrator assigns a subject (a role or a set of roles) to each user. The user has rights to act any of the roles associated with his subject. However, user request needs to be validated against his current active role(s) due to the organization's security policies.
- If the user is currently acting a particular role, his new request may conflict with the current role. In such a situation, user should not be allowed to act that role, or his current active role should be inactivated before the new role is activated.

## 40.5. Solution

Create an object that encapsulates the algorithm for organization's access control policy. This object accepts user request and checks the validity of the role against user's subject.



Validation defines interface for validation and creation of roles. This object interacts with the Subject object to validate user-specified role.

Subject holds information of roles assigned to a particular user and status (active or inactive) of those roles.

Role defines role objects based on the privileges of the corresponding role.

Session is an instance of the Session Object pattern describes next.

Validation object accepts user request and validates the user-specified role.

If user has not been assigned the specified role, validation object invalidates the request with an error message.

If user has been assigned the specified role, and he does not have any active role (session), validation object creates an appropriate role object and forwards the request to that object.

If user has been assigned the specified role, and he has a current session with the specific role, he is allowed to use the current session.

If the role conflicts with the currently active role, the request is forwarded to the Session object.

## 40.6. Structure

## 40.7. Dynamics

## 40.8. Implementation

Validation class can be implemented to read user input (user\_id and role\_name) and then interact with the Subject to validate the role. CreateRole() method might be implemented to create different role objects. One might use Abstract-Factory pattern for creation of role objects.

## 40.9. Example Resolved

The users may request to create a session with any role, but the validation object allows users to create sessions with only assigned roles. Validation object can be extended to perform other checks.

Subject holds only the information of user's permitted roles and role objects are created only if a user request to activate a role. This reduces considerable memory overhead.

## 40.10. Consequences

## **40.11. Known Uses**

- The login process for Telnet and ftp servers uses Validation.
- The Caterpillar/NCSA Financial Model Framework [1] uses a variant of this pattern to validate passwords, roles, and machines.
- The PLoP1998 registration program has a Validation, which only allows authorized users to log onto the system and change their user information.

## **40.12. See Also**

- Role Validator is a generalization of the Check Point pattern.

## **40.13. References**

[1] Yoder, J., Manolescu, D. (1998). The PLoP Registration Framework.  
<http://www.uiuc.edu/ph/www/j-yoder/Research/PLoP>

## **40.14. Source**

Kodituwakku, S. R., Bertok, P., & Zhao, L. (2001). APLRAC: A Pattern Language for Designing and Implementing Role-Based Access Control. In *EuroPLoP* (Vol. 1, p. 2001).

# 41. Privilege-Limited Role

## 41.1. Intent

## 41.2. Example

## 41.3. Context

After the administrator has assigned a subject to users, user create a Session with one of the roles in his subject to execute operations. This pattern achieves role-privileges assignment by restricting method execution.

## 41.4. Problem

The system must assess user requests on the basis of privileges belonging to the users' role. The question is: how can you implement the role object to reflect the role-privileges assignment?

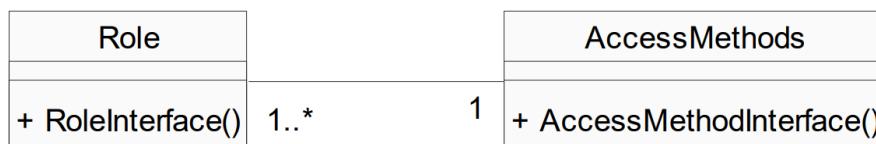
The following forces apply:

- Once a user is assigned a particular role, he can access information by creation a session. However, roles must be assigned privileges to ensure that roles are granted least privileges for information access.
- Role validation can be easier when roles are implemented with only permitted operations. However, users may be confused when some operations are either not present or disable.

Since administrator pattern creates only subjects for users, roles assigned to the subject should be granted access rights to ensure that roles are not allowed to execute operations they do not have privileges for.

## 41.5. Solution

Only let the user to access what they are permitted. Define a role class for each role introduced by the Role Based Access Control pattern. The class interface should reflect the privileges of the role.



Role defines the interface with only permitted operation for the role and maintains a reference to AccessMethod.

AccessMethods declares the interface for all operations perform within the organization and implements them.

Client request Role object to execute operations. In turn, the Role object forwards it to the corresponding method in AccessMethods object. In turn, AccessMethods execute the requested operation.

## 41.6. Structure

## 41.7. Dynamics

## 41.8. Implementation

## 41.9. Example Resolved

- The role classes are defined with only operations permitted to them. Consequently, users are strictly limited to access only information to perform their job functions. As a result, roles need not implement the operations they do not permit.
- Actual operations are implemented within the AccessMethods class, and it is exclusively included in Role objects. Therefore, user access to the information is completely controlled by roles.
- In general, organizations perform a large number of operations, and it is infeasible to define all of them as a single class (AccessMethods). In such a situation, AccessMethods can be decomposed into a number of small objects.

## 41.10. Consequences

## 41.11. Known Uses

OMG-CORBA uses the Proxy variant of the Role Privilege Assignment pattern for two purposes. The client-stub guard clients against the concrete implementation of their servers and the ORB. ORB uses the IDL-skeleton to forward requests to concrete remote server components.

## 41.12. See Also

- Limited-View [1] designs application so user can request to perform only legal operations.
- Proxy pattern provides an indirection to access information. In particular, Protection Proxy pattern protects information from unauthorized access.

## **41.13. References**

- [1] Harrison, N., Foote, B., & Rohnert, H. (Eds.). (2000). *Pattern languages of program design 4* (pp. 15-32). Reading, MA: Addison-Wesley.

## **41.14. Source**

Kodituwakku, S. R., Bertok, P., & Zhao, L. (2001). APLRAC: A Pattern Language for Designing and Implementing Role-Based Access Control. In *EuroPLoP* (Vol. 1, p. 2001).

# 42. Role Hierarchies

## 42.1. Intent

## 42.2. Example

## 42.3. Context

In most of the organizations, users have overlapping responsibilities. Therefore, users acting different roles need to perform some common operations.

## 42.4. Problem

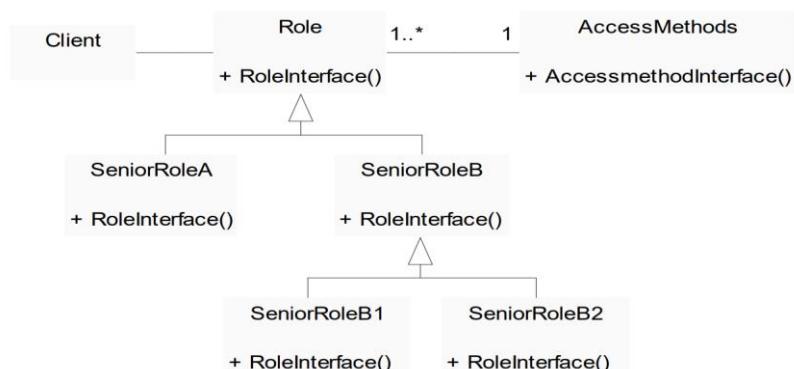
Natural role hierarchies can be identified from most of the organizations. The problem is: how can you design and implement such role hierarchies?

The following forces apply:

- Roles must be designed in such a way that roles in lower down in the hierarchy (senior roles) has all privileges granted to its junior roles, but junior roles should not have additional privileges granted for its senior roles.
- Although senior roles are allowed to access information granted for its junior roles, some of the privileges granted to junior roles may not be granted to a particular senior role(s).

## 42.5. Solution

Define all role objects in the hierarchy with a common interface that conform to the most senior role's privileges. Use Hook Method pattern to define the interface. For each role, implement these methods such a way that only authorized methods get execute.



Role. Defines the interface including all methods (operations) permitted to roles in the hierarchy. The authorized methods are implemented to invoke the corresponding method in AccessMethods. Maintains a reference to AccessMethods.

SeniorRoles (SeniorRoleA, SeniorRoleB, SeniorRoleB1, SeniorRoleB2) Overrides the methods, which are authorized to them but they are not implemented by any of its junior roles.

If any method authorized to its junior role is not authorized to it, override the method with an empty implementation.

AccessMethods declares the interface for all operations perform within the organization and implements them.

Client always requests Role to execute methods (operations).

Role objects forward the valid requests to AccessMethods and invalid requests are denied.

## 42.6. Structure

## 42.7. Dynamics

## 42.8. Implementation

Since each role must be implemented to reflect its access rights, senior roles need to implement only authorized methods, which are not implemented by any of its junior roles. When senior role is not granted some of the methods granted to its junior roles, they can be overridden with empty implementation.

## 42.9. Example Resolved

- When a senior role is revoked from a user, he does not have rights to execute operations permitted for junior roles. If user need to act a junior role, it should be reassigned.
- Users can request to execute unauthorized methods, but the implementation of role objects ensures that only authorized operations get execute.
- Role hierarchies reduce the number of user role assignments. However, it is difficult to handle some restrictions. For example, once a user creates a session with a senior role, all is junior roles are also activated. Therefore, it is hard to restrict users to act only one role at a time.

## 42.10. Consequences

## **42.11. Known Uses**

Many object-oriented user interface toolkits use this pattern to add graphical embellishments to widgets.

## **42.12. See Also**

- The Hook Method pattern is used for implementation.
- The Strategy pattern can be used to design different access control algorithms.

## **42.13. References**

## **42.14. Source**

Kodituwakku, S. R., Bertok, P., & Zhao, L. (2001). APLRAC: A Pattern Language for Designing and Implementing Role-Based Access Control. In *EuroPLoP* (Vol. 1, p. 2001).

# 43. Risk-Based Authenticator

## 43.1. Intent

The RISK-BASED AUTHENTICATOR (RBA) pattern uses information on a subject, so called Authentication Indicators, to verify their risk of not being authentic and, in case of a high risk, it takes further steps to raise the trust in the subject's authenticity. A profile is generated for each subject by continuously and transparently gathering information on them during a session. A risk, that the subject is not authentic, is calculated for each request to the RBA by comparing the authentication indicators with the profile of the subject. If the risk exceeds a defined limit a step-up or reauthentication may be scheduled to reduce the risk or the access is denied.

## 43.2. Example

We want to build an online shop for shoes that addresses consumers with a web application through the internet. Several other shops exist and, thus, usability is one of the key quality characteristics we want to address while still being secure. Users shall offer their personal data as simple as possible and authenticate with easy-to-use methods. We use a weak authentication based on social networks (social login) because it is easy to use. But, if a risky action is performed, e.g., if the user wants to initiate payment, we want to be sure they are authentic, and their account is not hijacked.

## 43.3. Context

Web applications that offer services for consumers and that are accessible through the internet. The consumer needs to authenticate in order to use the services. Transactions of the web application need to be secured properly.

## 43.4. Problem

In general, we are aware of vulnerabilities in our authentication, e.g., passwords can be stolen, fingerprints can be imitated, or the authentication protocol implementation may have a security flaw. Thus, hijacking user accounts is a problem in our web application. How can we prevent attackers from impersonating as a legitimate service consumer and, as a result, causing damage to the user and our enterprise? The following forces have to be tackled by the solution:

- Usability: Ease of use is a competitive advantage and users will switch to a competitor in case of bad usability.
- Effort: The user should be authenticated with little effort. If the effort is too high, the user will not use the application.
- Frequency: The user should interact as often as needed but not more. The frequency of user interaction to authenticate directly affects the usability and should be kept small.

- Performance: The response time of requests in the web application as well as the web user interface's response time should keep almost the same when using a transparent authentication.
- Cost: The authentication should be cost-effective, e.g., the need for hardware raises costs and does not scale well.
- Security level: The authentication should ensure a high security level, thus, the probability that the user is not authentic is small.
- Security: The information used to authenticate the user must be securely stored to avoid unwanted access.

### 43.5. Solution

You should continuously and transparently collect information on users and learn about their typical context. Based on the gathered information, calculate a risk for a concrete request context. If the risk is high, process a case handling for the request, e.g., schedule a step-up authentication, inform the account owner, or deny access. Perform a risk-based authentication for a request based on your requirements, e.g., if a valuable asset is accessed or on each request.

### 43.6. Structure

In Figure 81 the structure of the pattern is depicted. The RbaAuthenticator realizes an Authenticator. Thus, it authenticates a User, and, in case of success, it creates a ProofOfIdentity. The RbaMediator that the user is not authentic, and for giving a recommendation on how to process their request. For this purpose, the RbaAuthenticator forwards each request to the RbaMediator.

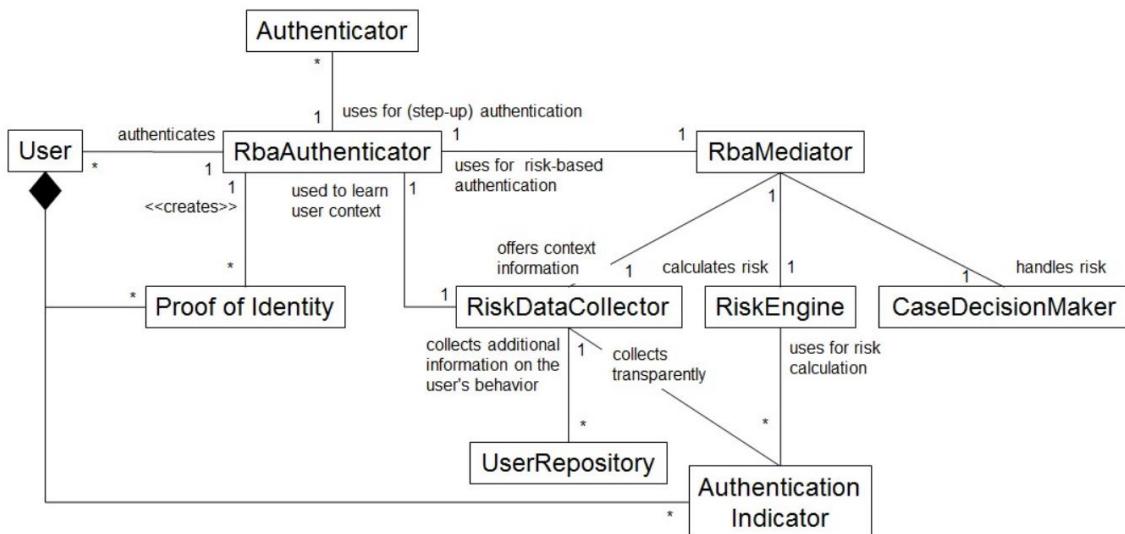


Figure 81: Structure of the Risk-Based Authenticator based on the Authenticator pattern [Schumacher et al. 2006]

The RiskDataCollector collects AuthenticationIndicators and the subject's history information from the http request and UserRepository. The RiskEngine takes the AuthenticationIndicators and calculates a risk for the current request. The CaseDecisionMaker decides how to handle the request based on the risk score. The RbaAuthenticator processes the request taking the recommendation in mind, e.g., it invokes an authenticator or destroys a previously created

`ProofOfIdentity`. It may act against the decision of the `CaseDecisionMaker` under certain circumstances, e.g., if it shall reauthenticate the user, although this does not reduce the risk score.

### 43.7. Dynamics

The flow of the RBA is separated into an abstract flow, that handles the authentication without knowing about risk, and the flow mediated by the `RbaMediator`. We separated the flows of the RBA into three use cases. The use case “Intercept a request” describes the abstract flow while the use cases “Register user” and “Risk-based authenticate user” describes the mediated flow.

The first use case “Intercept a request” focuses on handling further steps based on the risk score, Figure 82. The `RbaAuthenticator` processes the http request to the `RbaMediator` which returns a recommendation for handling the risk. Afterward, it handles the recommendation, authenticates the user, takes further steps with contacting other Authenticators, or rejects the request.

The other use cases describe how the `RbaMediator` handles the first request (use case “Register user”) and how a typical request (“Risk-based authenticate user”) of a user is processed using the `RiskDataCollector`, `RiskEngine` and `CaseDecisionMaker`, Figure 83.

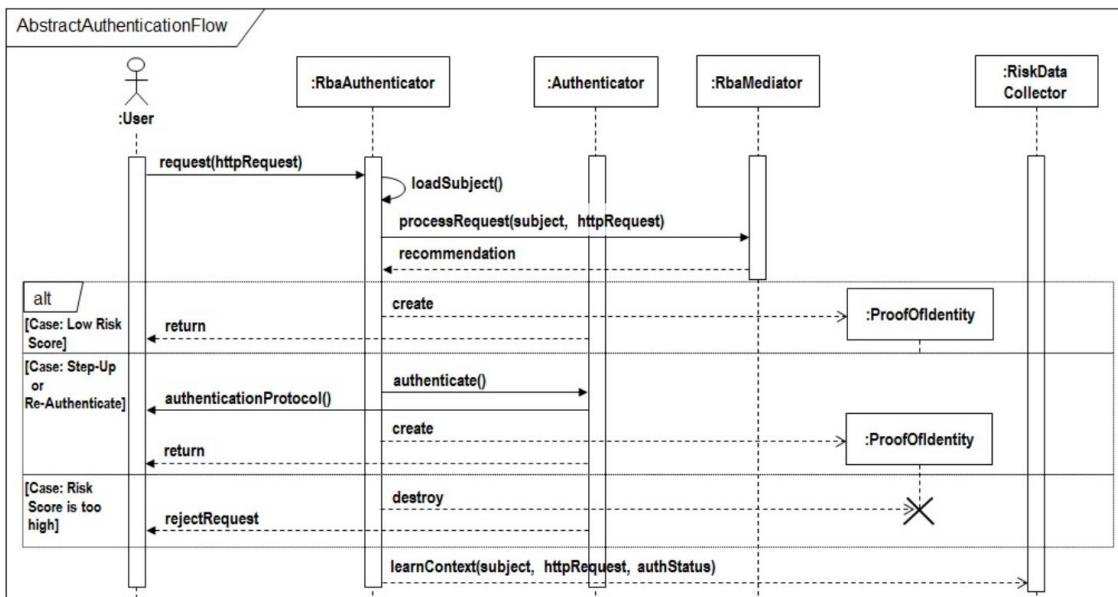


Figure 82: Sequence diagram for use case 1 “Intercept request and process according to risk score”

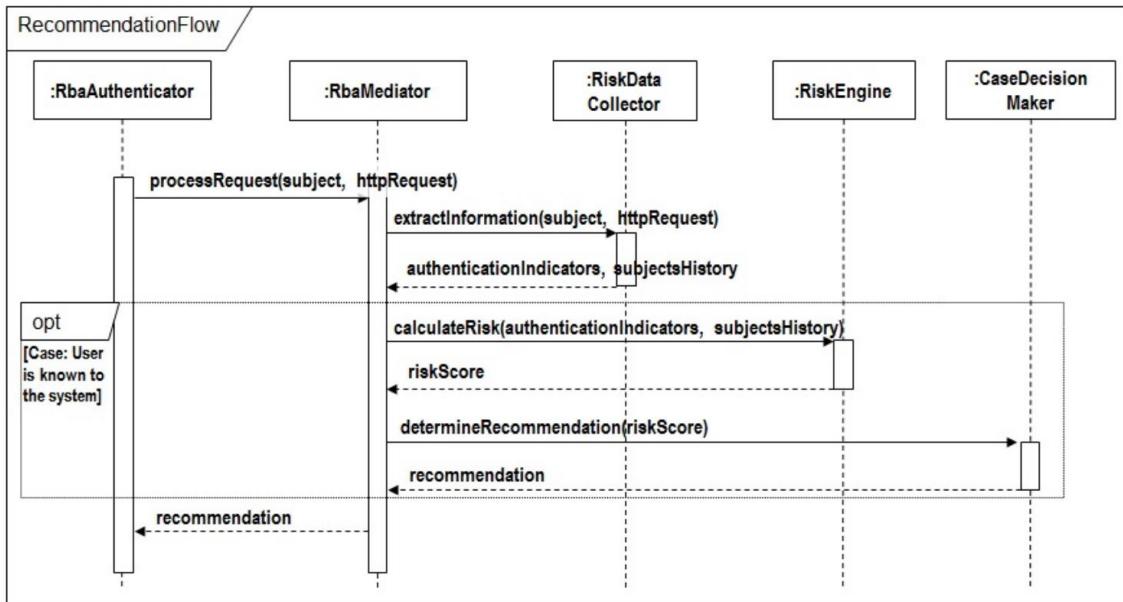


Figure 83: Sequence diagram for use case 2 “Register subject and learn basic context” and use case 3 “Process risk-based authentication request”

### 43.8. Implementation

There are three areas to focus on when implementing the RBA. First, the AuthenticationIndicators, that shall be used to calculate the risk score, have to be determined. Second, all information needed to authenticate the subject based on the AuthenticationIndicators as well as methods to gather this information have to be defined. Third, for each possible risk score the recommendation of the CaseDecisionMaker has to be specified.

The three areas are discussed using a simplified implementation of a campus web application. The application offers services for students. They can see an overview of their grades by authenticating with their matriculation number and a password. The grades our web application offers have a high demand for confidentiality.

#### 1. Authentication Indicators need to be chosen.

We have to choose authentication indicators (hereafter only “indicator”) according to the requirements of our web application. The demand for confidentiality is high, thus, we need to choose indicators that assure a high trust in the user’s authenticity. We decide to take a browser fingerprint, because studies show that these fingerprints identify users with an accuracy above 90% Eckersley [1]. Additionally, we take the geo location of the request into consideration and count the failed authentication requests. In our simplified scenario, a change of the fingerprint shall result in a high-risk score. Requests from foreign countries shall raise the risk score and reduce the count of allowed authentication attempts.

We took the design decision, that the risk score is a ratio scale ranging from 0 to 100 [percent]. According to the previous considerations, we define the following risk calculation, that our RiskEngine implements.

The RiskEngine calculates a sub risk score for each authentication indicator and sums these sub risk scores up, while the maximum is 100. - The sub risk score of the browser fingerprint

indicator is 100, if the browser fingerprint changes, and otherwise 0. - The sub risk score of the authentication attempts indicator is  $20 * |\text{failedAttempts}|$ . - The sub risk score of the geo location indicator is 60, if the user tries to access from a foreign country, and otherwise 0.

2. The RiskDataCollector must collect the relevant information to calculate the risk.

In our scenario, we have three indicators to collect - browser fingerprint, authentication attempts and geo location.

We use an open-source framework (<https://github.com/Valve/fingerprintjs2>) to calculate a browser fingerprint and add it to each request in the HTTP header. The fingerprint is recorded on the user's first login, where they have to use an initial password, they received by encrypted mail. Implementing the authentication attempts indicator is quite simple. The value is initiated to 0 and reset on successful authentication. For each unsuccessful attempt, the value is increased by 1. The geo location is resolved using a free web service (<http://ip-api.com/>). The IP address of the user's HTTP request is sent to this service. The answer of the service includes the country (geo location) of the IP address.

3. The CaseDecisionManager has to be configured to handle the risk score.

Now, we have to define, which risk scores lead to which re- or step-up authentication method and which risk score is too high and results in the denial of the authentication. The risk score has to be harmonized; CaseDecisionMaker and RiskEngine must have the same understanding.

In our example, this is achieved by defining a risk score range. In addition, the risk calculation should be taken under consideration, when defining the risk score handling. As our application has a high demand for confidentiality, we use a strict strategy. Someone accessing our application from a foreign country shall have just one authentication attempt. Requests using an unknown browser shall be declined directly. But the user can request an initial password at will (send to them encrypted) and add further browsers.

Thus, the CaseDecisionManager is configured to reject requests with an overall risk score higher than 70 and no step-up or re-authentication is scheduled. One failed authentication from a foreign country ( $60 + 20 * 1 > 70$ ) and four from the country of our university ( $20 * 4 > 70$ ) are allowed. Changing the fingerprint always results in a denied request ( $100 > 70$ ).

### 43.9. Example Resolved

### 43.10. Consequences

The RBA pattern offers the following benefits:

- **Effort:** Strong authentication mechanisms often go with a high user effort, e.g., when an extra device generating one-time passwords is needed and must be at hand. The RBA lowers this effort by enforcing a quite strong authentication based on authentication indicators. Additionally, if needed, strong authentication is performed only in case of high risk.
- **Frequency:** The user's authenticity is verified continuously, but transparent to the user. The user does not have to interact with the system.

- Performance: The RBA can authenticate the user asynchronously. Thus, an authentication request can be sent to the RbaAuthenticator while the user can go on interacting with the system. This is helpful when the authentication takes a long time.
- Cost: The RBA offers itself a quite strong authentication based on authentication indicators. Thus, it can be used to strengthen weak authentication, such as a password-based, without the need of new hardware.
- Security level: By applying the RBA, The Password Anti-Pattern [2] (identity delegation by revealing username and password to a foreign service provider) as well as stolen passwords are detected and can be treated.
- Separation of Concerns: The RBA separates concerns, thus, data collection, risk calculation and handling of risk is divided into different components.
- Extensibility: Existing Authenticators do not need adaptation. They are treated as resource for the RBA, and they are used for step-up or re-authentication.

The pattern has the following potential liabilities:

- Performance: If the authentication is performed synchronously, the RBA can lead to performance issues. This is typically the case for the first request of a session. Additionally, some authentication indicators may need to collect a lot of data to statistically identify the subject. This may slow down the web application or raise requirements to the subject's client.
- Privacy: Sensitive information on the user is collected. This may be communicated to the user, e.g., depending on the law enforced to the web application.
- Complexity: The RBA is an authenticator on top of existing authenticators; thus, additional components are introduced by the RBA making the application more complex.
- Security: Securing the information used to authenticate the user, including history data, is not part of the RBA. The data store must be secured properly.
- Security: The information used to authenticate the user must be securely stored to avoid unwanted access. Additionally, manipulating this data may lead to account hijacking.
- Storage: The history data of subjects raises the need for a data store.

### **43.11. Known Uses**

The RBA is implemented in several IAM products, such as the following:

- ForgeRock OpenAM [3], an open-source solution offering several authentication (single-sign on, adaptive authentication) and access control (web services security, entitlements) features. OpenAM offers a web user interface to configure the RBA. The administrator can define how the risk is calculated using given authentication indicators and they can define for different URLs, which risk score is acceptable and how high risk shall be handled.
- CA Risk Analytics [4], a commercial product offering authentication based on risk calculation for web and mobile platforms. The system calculates a fingerprint for devices in order to identify subjects. The risk calculation is based on neural networks, learning throughout the subject's session. The software focuses on e-commerce web applications.

- SafeNet Context-Based Authentication [5], offers an RBA authentication based on configurable context information, e.g., Ip-address, daytime, and device identifiers. Their “Context Engine” calculates a context assurance level, which requires a specified authentication level. Each authentication mechanisms are associated with an authentication level. Depending on this information, a step-up authentication may be initiated.
- PortalGuard [6], uses multiple factors to identify the subject. Here, the parameters for allowing access are strictly specified, e.g., access shall only be allowed during office hours. If a subject falls outside this parameters, access is denied. These parameters can be defined by the administrator.

Google, as mentioned in [7], Facebook, the video game developer Blizzard Entertainment and other companies implement the RBA, too. They reject suspicious requests, force to register new devices or schedule further authentication steps in order to strengthen trust.

### 43.12. See Also

- The RBA is a specialization of the Authenticator pattern using Authentication Indicators.
- The pattern needs at least one concrete Authenticator, such as Password, Biometric or X.509, that proofs the identity of a subject. Additionally, these Authenticators can be used by the RBA for step-up authentication.
- When using a Security Session pattern, the RBA can be used to verify the authenticity of the user. For example, a change of the user’s location in a limited extend is acceptable, whereas a complete change of the location indicates a high risk and may invalidate the session.
- An Intercepting Web Agent [8] may be used to transparently authenticate requests using the RBA without affecting the web application.
- Access Control taking Risk into consideration, e.g. [9; 10], is comparable to the RBA. But, in contrast to the RBA, these access control models (being quite similar to patterns) also consider the accessed resource for risk calculation.
- The Information Obscurity pattern may be used to secure the stored information.

### 43.13. References

- [1] Eckersley, P. (2010, July). How unique is your web browser?. In *International Symposium on Privacy Enhancing Technologies Symposium* (pp. 1-18). Springer, Berlin, Heidelberg.
- [2] Crumlish, C., & Malone, E. (2009). *Designing social interfaces: Principles, patterns, and practices for improving the user experience.* " O'Reilly Media, Inc.".
- [3] ForgeRock Community. (2016). OpenAM Project. Retrieved November 14, 2016, from <http://openam.forgerock.org>
- [4] CA Technologies. (2016). CA Risk Authentication. Retrieved November 14, 2016, from <http://www.ca.com/us/products/ca-risk-authentication.html>
- [5] SafeNet Inc. (2016). Context-Based & Step-Up Authentication Solutions. Retrieved November 14, 2016, from <http://www.safenet-inc.com/multi-factor-authentication/context-based-authentication>

- [6] PistolStar Inc. (2016). Contextual Authentication. Retrieved November 14, 2016, from <http://www.portalguard.com/contextual-authentication.html>
- [7] Grosse, E., & Upadhyay, M. (2012). Authentication at scale. *IEEE Security & Privacy*, 11(1), 15-22.
- [8] Steel, C., & Nagappan, R. (2006). *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Pearson Education India.
- [9] Chen, L., & Crampton, J. (2012). Security and Trust Management: 7th International Workshop, STM 2011, Copenhagen, Denmark, June 27-28, 2011, Revised Selected Papers. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Risk-Aware Role-Based Access Control, 140–156.
- [10] Ni, Q., Bertino, E., & Lobo, J. (2010, April). Risk-based access control systems built on fuzzy inferences. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (pp. 250-260).

#### **43.14. Source**

Steinegger, R. H., Deckers, D., Giessler, P., & Abeck, S. (2016, July). Risk-based authenticator for web applications. In *Proceedings of the 21st European Conference on Pattern Languages of Programs* (pp. 1-11).

# 44. Role Based Fine Grain Access Control

## 44.1. Intent

Presents an approach for access control to a business data object and its individual attributes and operations, that depends on subject's roles and the access rights assigned for specific roles to the object, attributes, and operations.

## 44.2. Example

## 44.3. Context

Any environment where subjects (users, clients) are assigned to roles and access to protected business objects, their attributes, and operations must be authorized differently based on the subject's roles.

## 44.4. Problem

Subjects must access protected objects, their attributes, and execute their operations with discrete level of access control and in a way that is scalable and convenient with a large number of objects and subjects. Access permission is granted by a central authority.

## 44.5. Solution

This pattern, shown in Figure 84, is derived from the more abstract pattern for Fine Grain Access Controlled Business Objects, described above. The access right information for business objects (DataAccessRight class) and attributes (OperationExecuteRight class) are associated with roles identified by Role objects. The accessType attribute indicates the rights type (e.g. read, update) that are possible for subjects with a particular role. Similarly, the OperationExecuteRight object specifies that subjects with the associated Role have the right to execute the corresponding operation.

Roles form a hierarchy defined by the subRoleOf association: a subject member of a subrole A is considered to be a member of role B if there exists an association A subRoleOf B between objects A and B.

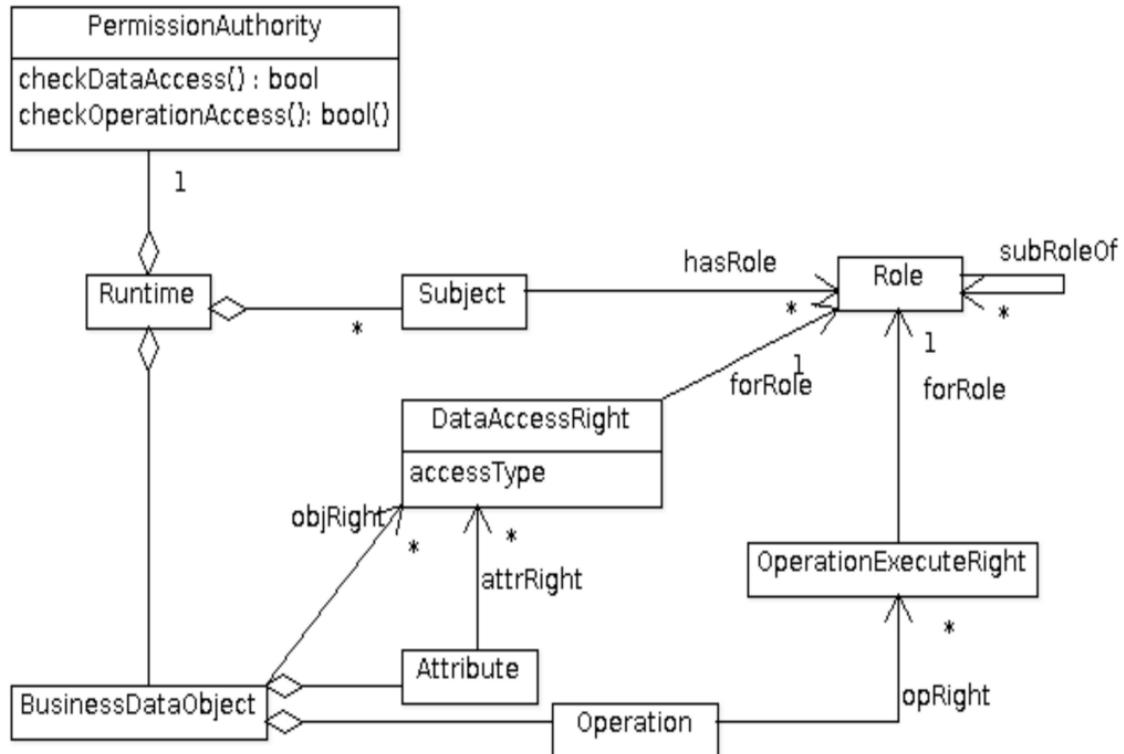


Figure 84: Class diagram for the Role Based Fine Grain Access Control Pattern

With a basic approach, the **PermissionAuthority** object validates access rights to attributes on behalf of the **Runtime** with the following pseudocode algorithm. Note that subject role membership must consider the recursive subrole association.

```

bool checkDataAccess(BusinessDataObject obj, Attribute attr, Subject sub, AccessType type) {
    for (or in obj.objRight) {
        if (or.accessType matches type) and (sub is a member of or.forRole) {
            // subject can access the object. Can it also access the attribute?
            for (ar in attr.attrRight) {
                if (ar.accessType matches type) and (sub is a member of ar.forRole)
                    return TRUE // access granted
            }
        }
    }
    return FALSE
}

```

A similar algorithm validates subject access to operation execution.

## 44.6. Structure

## **44.7. Dynamics**

## **44.8. Implementation**

## **44.9. Example Resolved**

## **44.10. Consequences**

Using roles allows the application developers and administrators to reduce the effort for specifying access permissions as large groups of subjects (e.g. users, web clients) can be assigned to a relatively small number of roles. In addition, access rights for large sets of objects (and their fields) can be handled efficiently through automation and default policies. A per-field rights mechanism provides a much finer level of access control that is suitable for current large web-based systems.

## **44.11. Known Uses**

Unix file access rights is based on a combination of groups, users, owners, and permissions. Users are organized into groups, and each group is assigned read, write, and execute permissions.

## **44.12. See Also**

## **44.13. References**

## **44.14. Sources**

RUBIS, R., & CARDEI, D. I. (2014). Patterns for fine-grain access-controlled business objects. PLoP.

# 45. Secure Storage

## 45.1. Intent

Secure storage provides confidentiality and integrity for stored data, and additionally enforces access restrictions on entities that want to access data. In particular, the secure storage can verify the integrity of software components, and it grants access to the data only to authorized, unmodified components that meet these restrictions.

## 45.2. Example

Consider the problem of storing passwords (e.g., for web services) securely on a computer. Users want to protect their passwords, such that attackers who might install malicious software on the computer (e.g., phishers using malware) cannot access them. In this case, simply encrypting the passwords with a passphrase does not help: The software installed by the attacker might look exactly like the legitimate software (i.e., the password manager they normally use) to users, so they enter their passwords. Thus, the attacker obtains the passphrase, and can decrypt and steal all passwords. We need to store passwords in such a way that only the legitimate password manager can access them.

## 45.3. Context

You need to provide storage that protects the confidentiality and integrity of stored data, and which verifies the integrity of the software before granting access to the data. You trust the hardware, but you need to be able to verify that only authorized, unmodified software can access data stored in the secure storage.

## 45.4. Problem

Cryptographic techniques exist to protect the confidentiality and integrity of data, e.g., encryption and digital signatures. However, the secret keys need to be protected against unauthorized usage. Operating system access controls are not sufficient because the access control component could be manipulated or replaced by an attacker.

The following forces have to be resolved:

- You need to protect the confidentiality and integrity of data, even when the system is not running. Otherwise, an attacker could read or modify protected data.
- You need to protect secret cryptographic keys from unauthorized access and usage. Otherwise, an attacker could use the keys, e.g., to decrypt confidential data.
- You want to allow modifications of the operating system or application binaries. Otherwise, software installation and updates would be problematic.

## 45.5. Solution

A Root Key is used to encrypt and decrypt data and other keys (which in turn can protect data or other keys). The usage of the Root Key is controlled by a Root Key Control component. Root

Key and Root Key Control are both protected by trusted hardware, i.e., the secret part of the Root Key never leaves the hardware, and Root Key Control cannot be manipulated or replaced by users or other software programs. Root Key Control verifies the integrity of Applications and their Execution Environment before it performs cryptographic operations on behalf of an application.

## 45.6. Structure

Figure 85 shows the elements of the Secure Storage pattern. Root Key Control has solely access to the Root Key, and both are protected by hardware. The Root Key can protect an arbitrary amount of Data. A special case of Data is Application Keys, which Applications can use to protect their application-specific data. The Root Key Control maintains a mapping of what Data can be accessed by which Application. It uses integrity verification data (hash reference values, or digital signatures) to verify the integrity of Applications and their Execution Environment.

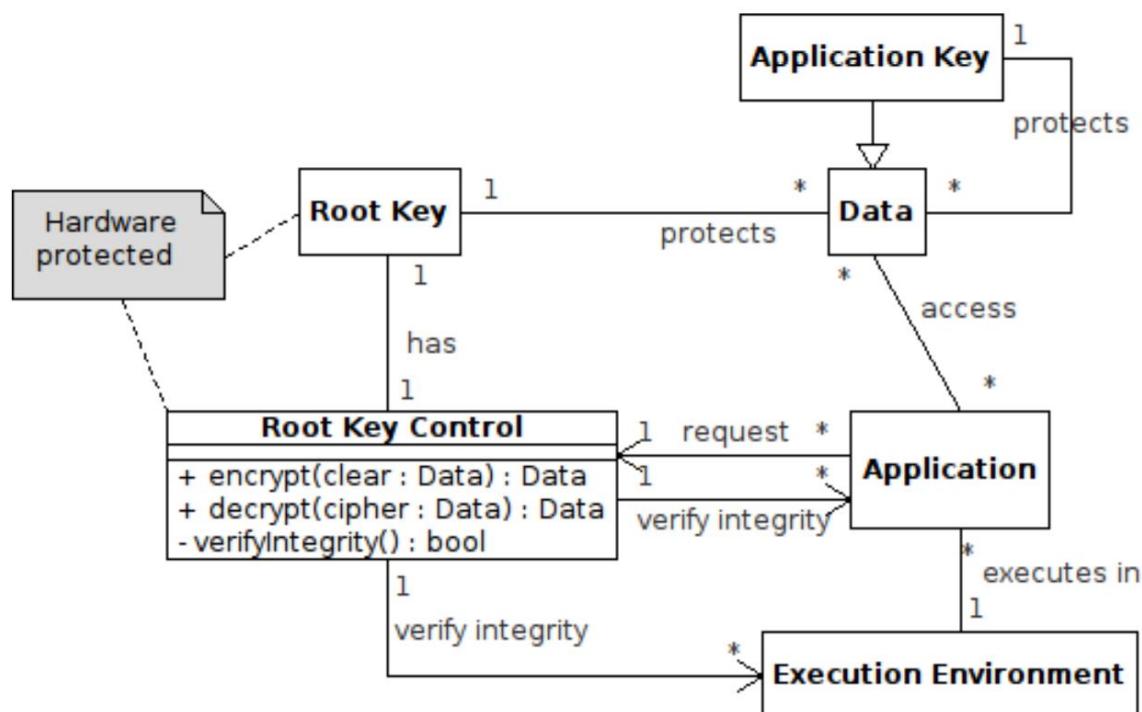


Figure 85: Elements of the Secure Storage pattern.

## 45.7. Dynamics

When an application requests to access Data, the Root Key Control verifies first the integrity of the Application and of the software components of the Execution Environment the Application is executed in (typically, the operating system components). Only if the integrity verification succeeds, Root Key Control uses the Root Key to decrypt (or encrypt) the Data. Note that the Root Key is never passed to an application. Instead, the Root Key Control performs the corresponding encryption and decryption operations and only returns the result.

## **45.8. Implementation**

## **45.9. Example Resolved**

Passwords are stored in secure storage, ensuring that the integrity of the application and its trusted computing base (TCB) are verified before the application can access the passwords. Access is granted only to the authorized password manager, and only if it is running in a secure execution environment on a secure operating system. It is important to include the TCB (here: operating system) into the verification process, because on an insecure system, malware might be able to read the memory of the password manager and hence gain access to the passwords.

## **45.10. Consequences**

The benefits from the secure storage pattern include:

- Only software where the integrity verification succeeded can access the protected data. In combination with a secure operating system, this can prevent malware from accessing sensitive data.
- Data can be stored on a system, such that it can be accessed only when the authorized operating system and software has been started.

The liabilities from the secure storage pattern include:

- Backup strategies become more involved because the encryption keys are protected in hardware. Data must be encrypted with a different key for the backup system, or a mechanism is needed to backup hardware-protected keys to other secure hardware.
- After an update, software cannot access protected data anymore, if no additional mechanisms are in place. Thus, software updates become more difficult.
- This pattern adds complexity and overhead. It needs to be coordinated with the other protection mechanisms.

## **45.11. Known Uses**

There are various implementations of Secure Storage:

- The Cell processor [1] features storage that can only be accessed when the processor is in a “secure state”. In this state, software that is running on the processor has been measured by a secure boot mechanism. This way, the Cell processor can provide secure storage.
- TPM sealing [2] is a mechanism specified by the TCG and implemented in the TPM. With this functionality, a key which can be used only inside the TPM can be restricted in its use by defining specific values for the PCRs. Since the PCRs securely store the recorded measurements from the authenticated boot process, usage of the key can be restricted to software that passes the integrity verification.
- The OMTP TR1 recommendation [3] includes detailed requirements for secure storage on mobile devices.

## **45.12. See Also**

Secure Storage requires Secure Boot to protect the integrity verification data. If Applications and their Execution Environment are also part of the Secure Boot, then Root Key Control can rely on the integrity verification during Secure Boot and does not need to perform it explicitly.

## **45.13. References**

- [1] Shimizu, K. (2006). The cell broadband engine processor security architecture. *IBM DeveloperWorks*.
- [2] Trusted Computing Group. (2007). TCG TPM specification version 1.2, revision 103. <https://www.trustedcomputinggroup.org/specs/TPM>.
- [3] Open Mobile Terminal Platform Consortium. (2009). OMTP advanced trusted environment: OMTP TR1 (v1.1). recommendation document. <http://www.omtp.org/Publications.aspx>.

## **45.14. Sources**

Löhr, H., Sadeghi, A. R., & Winandy, M. (2010, February). Patterns for secure boot and secure storage in computer systems. In 2010 International Conference on Availability, Reliability and Security (pp. 569-573). IEEE.

# 46. Secure GUI System

## 46.1. Intent

Provide a trusted path between the user and the application the user intends to use. Security-critical applications require integrity and confidentiality of user input and displayed output. All input and output devices (keyboard/mouse, display) have to be shared by all applications, but security-critical applications need exclusive access to avoid interference with other applications.

## 46.2. Example

Consider a system login dialog on a graphical window system where the user has to enter a password. The user should be sure that the dialog being displayed actually corresponds to the system login and not to a faked application. Additionally, the data transmitted between the login dialog and the user (keyboard input and displayed graphical output) should not be possible to be eavesdropped.

## 46.3. Context

You need to provide a graphical user interface for several applications with different trust and security requirements for one or more users. You need to ensure that the application the user is in front of the input/output devices is currently interacting with cannot fake to be an arbitrary other (possibly trusted) application. You also need to ensure that the input of the user and the output of an application cannot be eavesdropped or manipulated by other applications.

## 46.4. Problem

Providing security for user interfaces is non-trivial because they often rely on and communicate with several components. The problem is to combine the individual goals (security and usability) in such a way that efficient interaction with the user interface remains possible.

Experience shows that commodity operating systems cannot provide sufficient security for the GUI of applications (e.g. [1]). Applications can impose other applications by adapting their look and feel. An attack on the user's privacy can be performed by applications that eavesdrop everything the user enters, e.g., keyloggers typically install themselves as device drivers to intercept all keyboard events.

In particular, the solution to this problem has to resolve the following forces:

- You need to provide a graphical user interface system that can be used by several applications simultaneously.
- You need the flexibility that applications can draw any content without constraints (application flexibility).
- Users need to know with which application they are currently interacting with (application authentication).

- Users need to be able to invoke interaction with a certain trusted service at any time (trusted path).
- You need protected input/output of GUIs, i.e., unauthorized applications should not be able to eavesdrop or manipulate user input directed to or display output coming from other applications.
- You (optionally) need controlled communication between user interfaces of different applications – for example, in a multi-level security system copy&paste has to adhere to the information flow policy.

## 46.5. Solution

The central idea is to mediate all user input/output through a Secure User Interface (SUI) system, and to separate the content drawn by applications from what is actually displayed on the screen. The SUI controls solely the graphics rendering hardware and the input events from the user input devices (typically, keyboard and mouse). The SUI receives the user input and sends it to the currently active application. Only one application can be active at a certain time, i.e., it has the “input focus”. On the other hand, the SUI receives the graphical output of all applications and multiplexes it to the graphics hardware, composed with a visible labeling of applications according to a policy. Hence, only the SUI decides what is going to be displayed on the screen and how.

Besides the input routing and the visible labeling, the SUI has to detect trusted path invocation, i.e., when the user requests to interact with a trusted service, and implement certain trusted path features, e.g., when the user wants to switch the input focus to another application.

## 46.6. Structure

The SUI system interacts with Applications on the one hand, and Users via Graphics Hardware (screen, graphics processing unit (GPU)) and Input Devices (typically, keyboard and mouse) on the other hand. Applications are defined as interactive programs that want to display a GUI on the screen and receive user input. Depending on the operating system, Applications can be single processes, groups of processes (e.g., belonging to the same security level in an MLS system), or entire virtual machines.

The main elements of the SUI system are the Input Manager, Display Manager, Copy&Paste Manager, a Policy, and the SUI Control (see Figure 86).

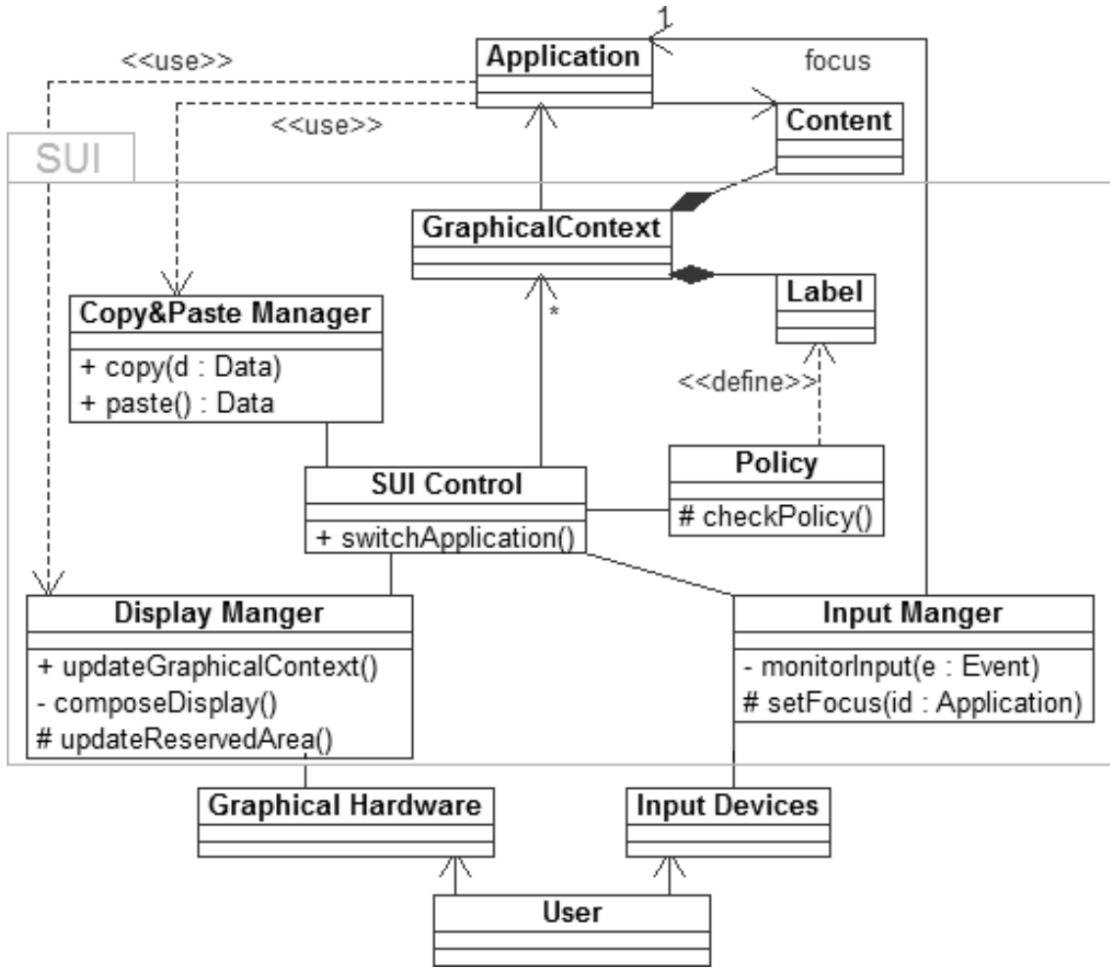


Figure 86: Participating elements of the Secure GUI System pattern.

The Input Manager is responsible for the input routing and detection of trusted path invocation. It monitors all input events and forwards them to the Application which currently has the focus. Certain input events are not forwarded, they are used as Secure Attention Key, e.g., a special key sequence, to invoke trusted path features of the SUI Control.

The Display Manager multiplexes the GUI output and is responsible for drawing visible labeling. Visible labels allow the User to reliably identify Applications, especially the one having the input focus. It is important to note that the visible labeling must be drawn in a way that arbitrary Applications cannot fake it. For example, the Display Manager could only display the content of one application at a time on the screen and keep a reserved area, e.g., on the top of the screen, in which no application can draw. This area is used to display the identity of the current application. The screen could also be divided into fixed regions belonging to different groups of applications, e.g., of the same security level. To allow most flexibility and to display all application windows simultaneously, the Display Manager could apply a window labeling policy [2], i.e., all windows are decorated with colored border where each color corresponds to a security level. In addition to colors, the name of the application (or its security level) can be shown in the window border, and a reserved area on the screen shows the current input focus.

The SUI Control implements trusted path features, in particular, to switch the input focus to another Application.

GraphicalContexts represent the GUI windows or screens of Applications. They are composed of two parts: a Label, which represents the identity of the Application (e.g., its name or security level), and the actual graphical Content. The Content is the only region to which the Application can draw its GUI elements. The Label is under control of the SUI. This prevents that an application can fake the visible identity of another Application since it has no access to the Label, and the Label is finally visually “attached” on the screen by the Display Manager.

The Copy&Paste Manager acts as trusted intermediate component to enable Applications to exchange data in compliance to the information flow policy of the system.

Finally, the Policy of the SUI defines the labeling strategy of the Display Manager, constraints to allow to switch the input focus, constraints to allow copy&paste, and the secure attention key events and corresponding reactions.

Note that the Display Manager and the Input Manager are the only components connected to the hardware input/output devices. This has to be enforced by the underlying operating system (OS), e.g., by allowing access to the corresponding device drivers exclusively to these components. Moreover, the SUI system relies on proper authentication of Applications and Users, which has to be done by the OS as well.

## 46.7. Dynamics

When an application wants to update its GUI, it draws into its Content object and requests to update the corresponding GraphicalContext. Figure 87 shows the corresponding sequence diagram. The important step in this process is the composition of the labeling information with the basic display content provided by the Application. The Application cannot alter the Label itself, as it is defined by the Policy based on the application identity and composed within the GraphicalContext by the Display Manager.

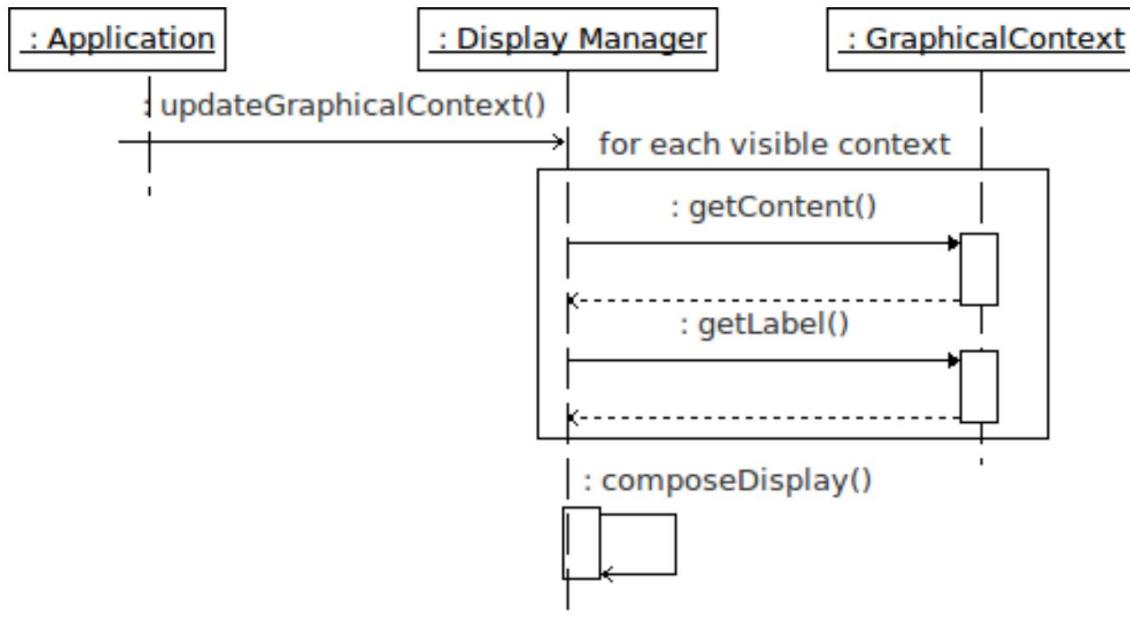


Figure 87: Sequence diagram for updating GraphicalContext

Users can request to switch the input focus by invoking the trusted path to the SUI Control (e.g., via a hot key).

Applications may also request to switch the focus to themselves or another Application, e.g., to request a password input or a confirmation from the user when they do not have the focus. In this case, the SUI Control asks the Policy and determines whether to allow or deny the request (see Figure 88). Alternatively, Policy can contain rules that effectively result in asking the user for a password in order to authorize the switch. Once authorized, SUI Control changes the input focus and updates the reserved area if required.

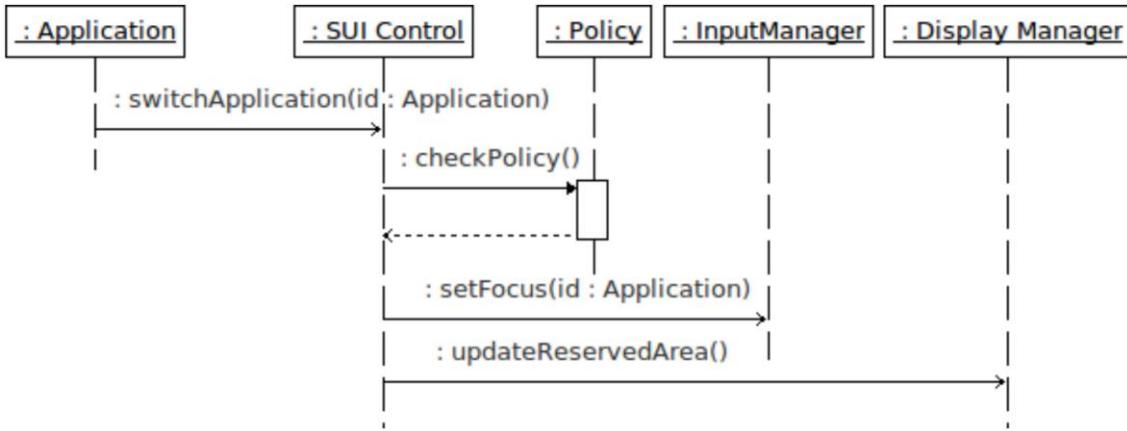


Figure 88: Sequence diagram for switching application

To enable copy&paste, the Policy must define access rules to the Copy&Paste Manager (CPM) which are complying with the information flow policy of the operating system. For example, in an MLS system it is not allowed to write data from high to low security levels. Therefore, all data transfer via GUI elements must go through the CPM. An application can copy (write) data to the CPM, and other Applications may request to paste (read) data from the CPM. The CPM has to enforce the policy on such requests accordingly.

## 46.8. Implementation

## 46.9. Example Resolved

By applying the Secure GUI System pattern, a Policy can be defined to keep a reserved area on the screen to display the active application's name. The Display Manager controls solely the graphics hardware and multiplexes the output to the display. Hence, no application can modify the reserved area in order to change the displayed name. This allows the user to identify the login application and to distinguish it from faked ones. Moreover, since the Input Manager routes the user input only to the one application that has the focus, the input cannot be eavesdropped by any other application.

## 46.10. Consequences

Applying this pattern is expected to result in the following benefits:

- You will get a trusted path for the user input and the application output. This channel is secure against eavesdropping and manipulation by unauthorized applications.

- Security-critical applications need not to implement their own protection mechanisms against faked dialogs because they can rely on the SUI to show the user the identity of applications via labels.
- The resulting SUI system is very flexible since labeling and access decisions can be delegated to the policy.

However, this pattern may also have the following liabilities as consequences:

- The SUI might become a single point of failure. If the SUI does not work correctly, the whole system might become unusable for users, or the security of user intentions is compromised.
- There might be usability issues due to security constraints of the GUI system. For example, when applying a window labeling policy (based on colors), users might need extra training and education to understand the meaning of the colors.
- You need to trust the SUI. For a high assurance system, the components within the SUI (cf. Figure 86) are critical and require high assurance for their development and integrity protection during runtime.
- The increasing usage of graphics accelerator functions in commodity operating systems and 3D game applications may pose an implementation problem since any direct access to the GPU could circumvent the SUI.

Today's GPUs lack of proper virtualization functionality. However, there are approaches to virtualize the 3D OpenGL graphics language [3], [4].

The actual implementation of the labeling in the SUI can be fitted to many scenarios. For example, one can implement different labeling methods for color blind people, and another can use a special braille external interface with separated "secure label zone" for blind people. It is also possible to decouple the label from the screen at all. For example, an additional small display attached to the keyboard could display the security level of the application that has currently the input focus.

Another area of extension is to control audio input/output in the same way as it is done with keyboard inputs. However, details are beyond the scope of this pattern.

## 46.11. Known Uses

The Secure GUI System pattern can be found in several existing implementations:

- Trusted X [5], [6] is a multi-level secure X window system. It encapsulates the untrusted functionality of an ordinary X server and polyinstantiates it for each security level. Trusted X multiplexes the windows of all levels and attaches a colored label on each window to indicate its security level. A reserved area on the screen always shows the level having the input focus. A dedicated trusted application implements the secure copy&paste function and mediates the data transfer according to the information flow policy.
- Solaris TX [7] supports multi-level desktop sessions in a similar way to Trusted X but uses a single trusted desktop system instead of polyinstantiated X servers.
- EWS [8] is a mandatory access control capable window system for the EROS operating system. It consists of a small display server that renders the output of client applications and supports window labeling to indicate the trusted path. It does not

have an X server, instead all drawing logic is in the applications, and shared memory is used to define Content. Secure copy&paste is realized via special invisible windows which are dedicated trusted applications.

- Nitpicker [9] is a small, framebuffer-based secure GUI server on top of the L4 microkernel. It controls the physical display and aggregates the virtual screens of clients, which can be native processes or virtual machines. The server also supports visible labels.
- Green Hills' INTEGRITY [10] is a microkernel-based operating systems that supports multi-level security. It displays the maximum-security level and the current input security level at the top and, respectively, bottom of the screen as a colored bar. Applications in the system are virtual machines (VMs) that run isolated from each other. The VMs can only access virtual devices and cannot draw on the reserved screen areas.
- The SDH architecture [11] divides the screen in separate regions according to security levels. For each region, an untrusted processor draws the Content, and a hardware implemented Display Manager multiplexes the output on the screen. A software-implemented Input Manager functions also as SUI Control, i.e., it switches current security level and input focus simply by moving the mouse pointer from one region on the screen to another.

While the examples above are mainly mandatory access control systems or research prototypes, commodity operating systems would also benefit from applying the Secure GUI System pattern. However, this would require changing the access to graphics and input hardware, i.e., not allow every application to arbitrarily use these devices (e.g., preventing installation of “filter” drivers that can intercept keyboard input). Moreover, the GUI system of a commodity operating system needs to separate the graphical content that applications can draw from what is finally presented on the screen (i.e., providing window labeling and/or a reserved area).

## 46.12. See Also

On an architectural view, the Secure GUI System pattern is a Single Access Point, for users to the graphical interfaces of the system's applications, but also for applications to the user input/output devices, especially keyboard/mouse and the graphical processing unit.

Reference Monitor intermediates access to resources defined by policy. This is similar to the Secure GUI System pattern since the SUI system controls the usage of the graphics output and the user input, i.e., which application receives the input. But the Secure GUI System has additionally to ensure authenticity of the end points. Actually, the Secure GUI System can use the Reference Monitor pattern to implement a policy-driven secure GUI.

The Authenticator pattern can be applied to provide proofs of identity of users and applications, which are requested by the Secure GUI System. For example, digital certificates of program binaries can be used to reliably identify applications, which the operating system has to verify when a process is started.

The SUI has to be executed in an Execution Domain which restricts the usage of the user I/O devices solely to the SUI. Other processes must not be allowed to directly access the GPU, otherwise they could easily circumvent the SUI.

To protect the input sent to an application and the output sent to the graphics hardware, it must be possible to isolate process memory. Otherwise, one process could directly read or modify the memory of another process and, hence, intercept or manipulate the data entered by or presented to the user. Process isolation can be achieved with the Controlled Virtual Address Space pattern.

## 46.13. References

- [1] Spalka, A., Cremers, A. B., & Langweg, H. (2001, June). The fairy tale of what you see is what you sign'-trojan horse attacks on software for digital signatures. In *Proceedings of the IFIP WG* (Vol. 9, No. 11.7, pp. 75-86).
- [2] Epstein, J. J. (1990, December). A prototype for Trusted X labeling policies. In [1990] *Proceedings of the Sixth Annual Computer Security Applications Conference* (pp. 221-230). IEEE.
- [3] Lagar-Cavilla, H. A., Tolia, N., Satyanarayanan, M., & De Lara, E. (2007, June). VMM-independent graphics acceleration. In *Proceedings of the 3rd international conference on Virtual execution environments* (pp. 33-43).
- [4] Smowton, C. (2009, March). Secure 3D graphics for virtual machines. In *Proceedings of the Second European Workshop on System Security* (pp. 36-43).
- [5] Epstein, J., McHugh, J., Orman, H., Pascale, R., Marmor-Squires, A., Danner, B., ... & Rothnie, D. (1993). A High Assurance Window System Prototype 1. *Journal of Computer Security*, 2(2-3), 159-190.
- [6] Epstein, J. (2006, December). Fifteen years after tx: A look back at high assurance multi-level secure windowing. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)* (pp. 301-320). IEEE.
- [7] Faden, G. (2006). Solaris trusted extensions. *Sun Microsystems Whitepaper, April*.
- [8] Shapiro, J. S., Vanderburgh, J., Northup, E., & Chizmadia, D. (2004). Design of the {EROS} Trusted Window System. In *13th USENIX Security Symposium (USENIX Security 04)*.
- [9] Feske, N., & Helmuth, C. (2005, December). A Nitpicker's guide to a minimal-complexity secure GUI. In *21st Annual Computer Security Applications Conference (ACSAC'05)* (pp. 85-94). IEEE.
- [10] Green Hills Software Inc. (2008, November). INTEGRITY PC Technology.  
<http://www.ghs.com/products/rtos/integritypc.html>
- [11] Sherman, R., Dinolt, G., & Hubbard, F. (1991, December). Multilevel secure workstation. U.S. Patent 5,075,884.

## 46.14. Source

- Fischer, T., Sadeghi, A. R., & Winandy, M. (2009, August). A pattern for secure graphical user interface systems. In *2009 20th International Workshop on Database and Expert Systems Application* (pp. 186-190). IEEE.

# **47. Neuralyzer**

## **47.1. Intent**

Describe how to manage the right to be forgotten in Big Data environments. Usually, this right is asked by the data subject, which is the person who wants that the system “forgets” his data. However, there are some scenarios where this action should be done automatically by the system.

## **47.2. Example**

## **47.3. Context**

Generally, Big Data ecosystems have a huge amount of personal and sensitive data stored in different databases and storage systems, supported by a variety of utility software.

## **47.4. Problem**

The right to be forgotten, also known as the right to erasure, is a feature that suggests that it is necessary to find a way to effectively delete data from storage systems. Erasing an individual's data implies to find and delete all his data; possibly scattered in several places. This functionality gets even more complicated in Big Data contexts where not only personal data is stored, but also information inferred from it due to the use of analysis techniques like business intelligence or machine learning.

The solution is constrained by the following forces:

- The gap between regulators and technology. There is a huge gap between the good intentions of the regulators and the complexity of real Big Data environments. It is necessary that our pattern meets both necessities.
- Flexibility. Big Data ecosystems can be used in different scenarios from a hospital to a factory. Different kinds of contexts require different solutions.
- Value. Probably, one of the main characteristics of a Big Data system is the value that can be obtained from this data. Due to the application of the RTBF, this characteristic can be compromised; erasing records may affect the obtained value from the analytics.
- Access Control. Due to the importance of the operation, any erasure or anonymization must be performed by order of the CDO: who will manage all the requests made by the data subjects, and who is the only role authorized to perform the data changes.

## **47.5. Solution**

Our solution focuses on an architecture pattern. Hence, it defines an architecture where the data subject demands the deletion of his data from all the databases of the ecosystem. It is important to highlight that the deletion of the data must be authorized by the data officer.

Then, an entity called Neuralyzer will delete all the data related to the data subject. In order to do that, it will use different techniques, for example, data masking of the data. These decisions must be taken by considering the context of the system and how it can affect the performance of the analytics. Different scenarios and organizations need different solutions, so the decision about which data can be deleted from the system is a complex process in which the CDO together with the top management of the company must consider all possibilities and thereby decide. Further details of the solution will be explained in the following subsections.

## 47.6. Structure

Figure 89 depicts the class model for this pattern. The Subject and the Chief Data Officer represent two interfaces where active entities are authenticated by the Server using an Authenticator Service. Once they are authenticated, they have different objectives. On one hand, the subject makes a request to delete all his personal data from the system. On the other hand, the data officer receives from the server the notification and starts the process to comply with the RTBF. The Neuralyzer is the main class in this pattern, its main goal is to decide which technique should be used depending on the context and the data stored from the user. There are three main categories of techniques represented by three different subclasses: Anonymization, Pseudo-anonymization, and Deletion. Once the Storage is properly modified to comply with the regulation the system should notify the subject that the erasure has been completed. All the operations performed on the data must be recorded in a log file using a Security Logger/Auditor. Table 2 summarizes the main components of the pattern and gives a brief explanation of its purpose.

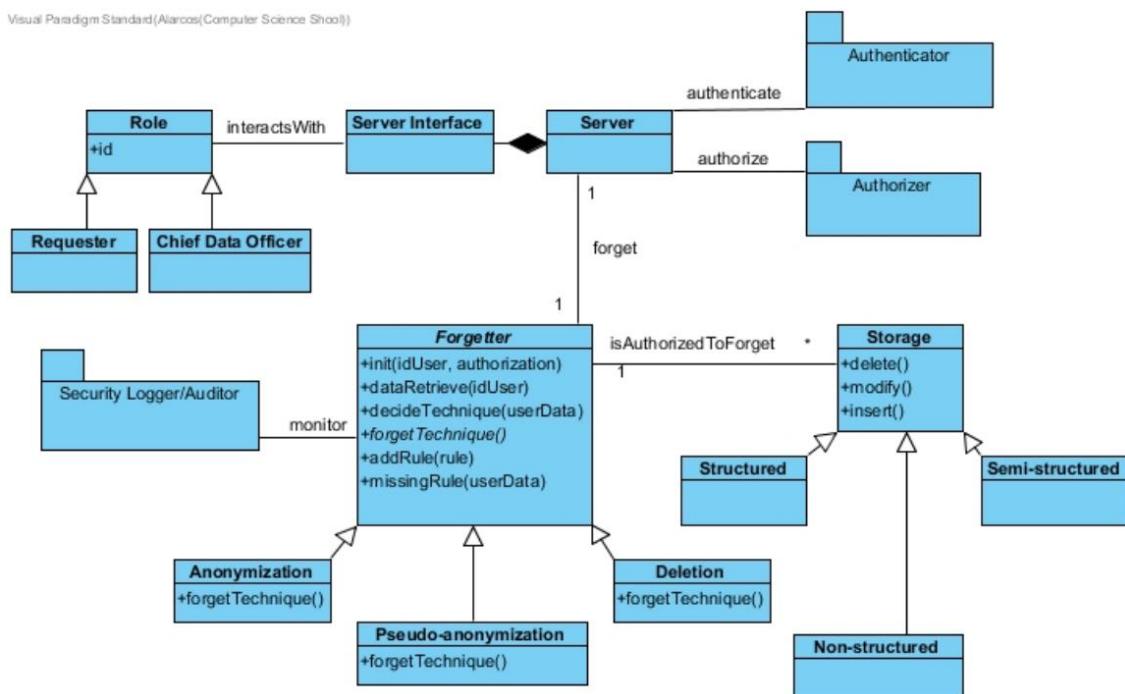


Figure 89: Class model of the Neuralyzer security pattern

Name	Type	Description
Role	Meta-subject	Generalization of the subjects that can perform actions in the pattern.

- Requester	Subject	Subject requesting that their data be erased from the system. They initiate the use of the pattern.
- Chief Data Officer (CDO)	Subject	CDO is responsible for a company's data at the highest level, both from a technological and business point of view, including security. This role adds the forgetting rules into the system based on the context of the company.
Server Interface	Software	Interface that shows all the actions provided by the server.
Server	Software	Server provides the actions related to the RTBF depending on the role that accesses. It acts like a bridge between the roles and the Neuralyzer pattern.
Authenticator	Security pattern	Auxiliary security pattern used to provide authentication to the users.
Authorizer	Security pattern	Auxiliary security pattern used to give specific permission to each role.
Forgetter	Software	The main class of the Neuralyzer pattern. Based on the forget rules implemented by the CDO and the data that should be forgotten, it applies the forget techniques on the storage system. Some cases require to use different techniques at the same time.
- Anonymization	Software	A mechanism to perform the Neuralyzer. Data anonymization is a technique that refers to hiding identity and sensitive data for owners of data records (Zhang, Yang, Liu, & Chen, 2014).
- Pseudoanonymization	Software	A mechanism to perform the Neuralyzer. Data pseudo-anonymization substitutes the identity of the user in such way that additional data is needed to re-identify the data user. In some cases, anonymization is enough to comply with the regulations.
- Deletion	Software	A mechanism to perform the Neuralyzer. Deletion of the subject's data from all the different data sources along the Big Data ecosystem.
Storage	Software	A collection of data that allows it to be accessed, managed, and updated. In Big Data ecosystems, there are typically three ways of storage data: structured, semi-structured, and non-structured data.
- Structured	Software	The structured storage can be considered as the traditional relational databases, for example, MySQL or PostgreSQL; usually, in this kind of stores, they use an SQL similar language to make queries to the data.
- Non-structured	Software	The non-structured stores, usually known as NoSQL databases, are widely used in Big Data ecosystems. In this kind of stores, there are four

		different subtypes that can be identified: graph based (usually used to represent data from social networks; for example, neo4j), columnar (in these stores each key is associated with one or more attributes, unlike the relational databases; for example, HBase or Cassandra), documental (data is stored with a document form, its main advantage is the scalability; for example, MongoDB or CouchDB), and key-value (they use a similar hash table style where each key is associated with a set of values; for example, Apache Accumulo or Riak).
- Semi-structured	Software	A way of storing data that is neither raw data, nor strictly typed as relational database systems; for example, JSON format can be considered as a semi-structured data format.
Security Logger/Auditor	Security pattern	A mechanism to perform the Neuralyzer. Most of the time, the application of the RTBF implies the management of sensitive personal information, for that reason, it is important to keep track of all operations performed in this process.

*Table 2: Components of the Neuralyzer pattern*

## 47.7. Dynamics

Figure 90 shows the use case “Forget a user”, which corresponds to the sequence:

1. Requester requests to be forgotten from the system.
2. Requester is asked for authentication.
3. Requester authenticates in the system.
4. Server receives proof of authentication.
5. Server initiates the Forgetter.
6. Forgetter queries the Storage systems to receive all the data from the requester.
7. Forgetter receives the data related to the requester.
8. Forgetter decides which technique will use depending on the data retrieved and the rules created by the Chief Data Officer.
9. In this case, an anonymization technique is used to forget the data from the user.
10. The anonymization technique is used in the Storage systems.
11. The server notifies the involved users.

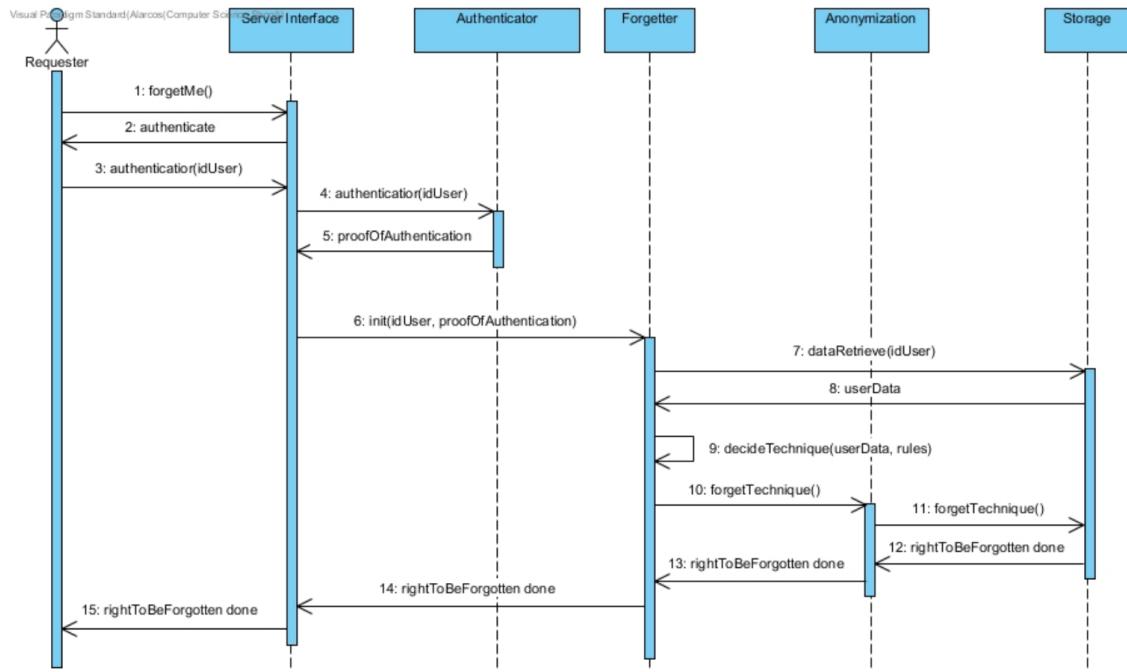


Figure 90: Sequence diagram for the use case "Forget a user"

This scenario can be considered as the “ideal case” in which the data is easily found, and the forgetting rules are perfectly designed. Although this pattern is supposed to work automatically without human interaction, in some cases the CDO will be asked to introduce new rules that meets these characteristics. Those new rules will be added to the system, so they can be reused in the future. This scenario is depicted in Figure 91.

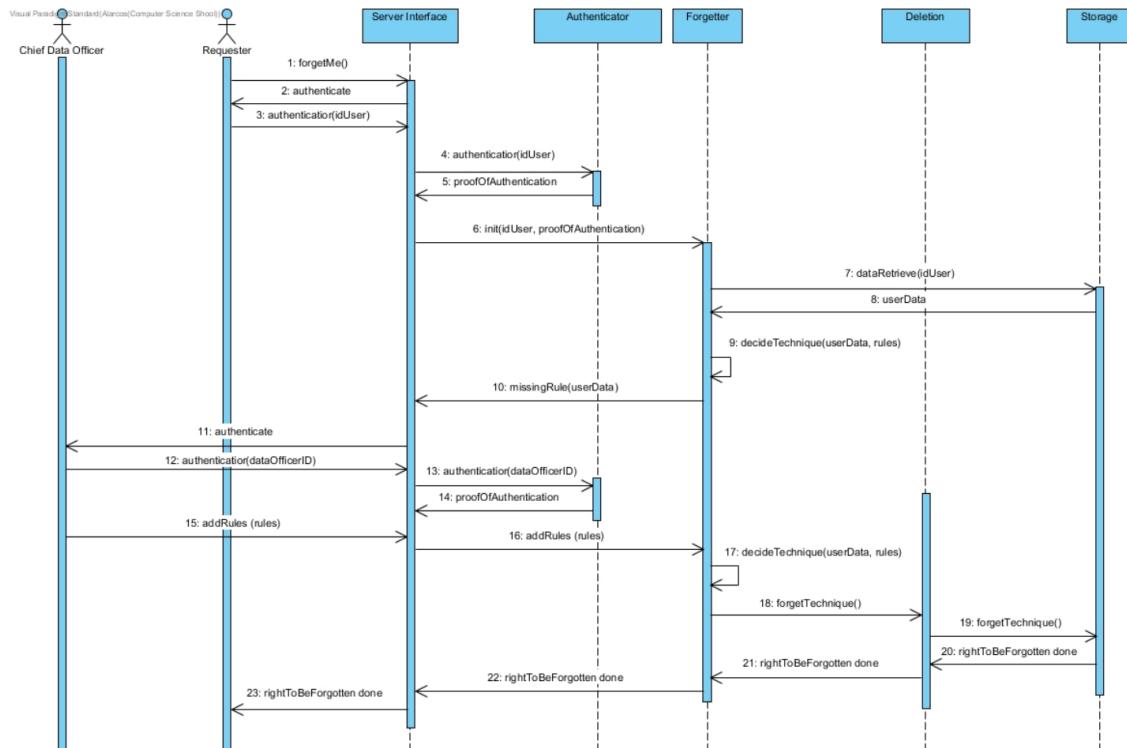


Figure 91: Alternative sequence diagram

## 47.8. Implementation

A possible implementation is shown in Figure 92. In this object diagram, we have a scenario with the two roles involved: the requester (who initiates the request to delete their data), and the Chief Data Officer (who is the person in charge to manage the deletion of the data). We have also added the role of data scientist, which is normally present in this type of system, to show that in this particular case it does not have any kind of right over the data. Furthermore, in this scenario, we have a Big Data system that has a storage system based on HDFS (Hadoop Distributed File System).

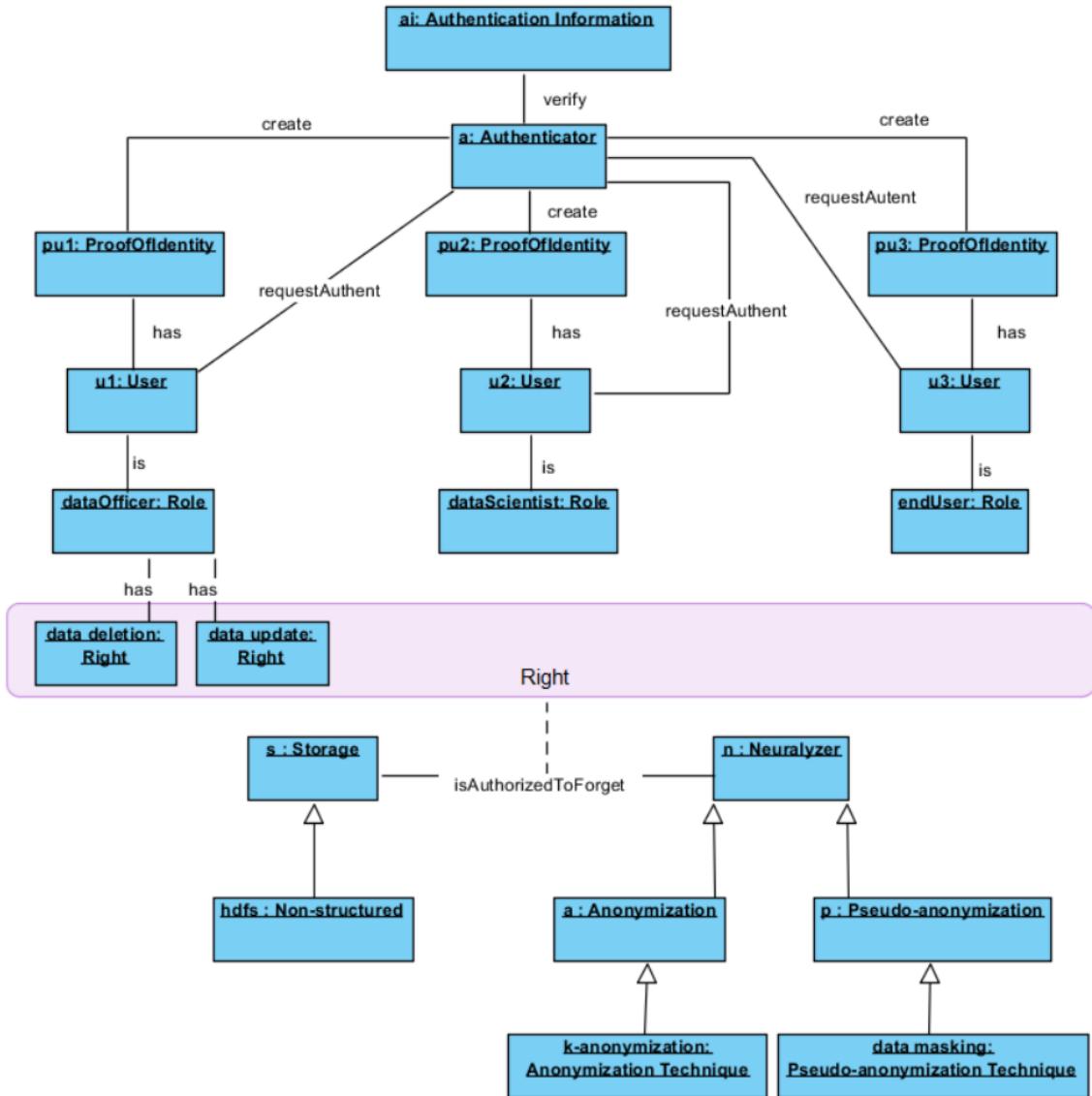


Figure 92: Example of use of the Neuralizer pattern

Also, this is a scenario where there is no need to fully eliminate all the data; instead of doing that it is enough to apply anonymization (for example, k-anonymization) or pseudo-anonymization (for example, data masking) techniques on the data. This example tries to highlight that the pattern depends on the context and regulations that can affect the Big Data environment. It is also affected by the implementation features of the system, for example, the use of HDFS.

## **47.9. Example Resolved**

## **47.10. Consequences**

Advantages derived from the use of this pattern include the following:

- The gap between regulators and technology. The appropriate use of different techniques can reduce this gap. However, it is still important to improve training on these topics for regulators and IT people.
- Flexibility. The pattern includes three ways to face the right to be forgotten problem. Hence, it can be adapted to different contexts.
- Access Control. We can use RBAC to control actions on the databases and restrict access to only the CDO role.

However, the implementation of this pattern can have some liabilities:

- Value. Each time any record of the data being analyzed is deleted, it is highly likely that the results will be affected and vary from the previous time.

## **47.11. Known Uses**

- In [1] the authors explain the main approaches to deal with the right to be forgotten in Artificial Intelligence environments. These solutions can be classified in three categories: anonymization, pseudo-anonymization, and deletion of the data. They also conclude that all these solutions have a relevant impact in the quality of the analysis. Furthermore, in [2] the author also highlights that the best way to comply with the right to be forgotten is the anonymization of the data. For that purpose, he shows some examples related to the GDPR regulation. We created an entity named “Neuralyzer” that generalizes the main kinds of solutions used to comply with the right to be forgotten.
- BankingHub (3). Explains a specific scenario where the right to be forgotten is applied in a bank context. In this case, they tag the different types of data stored and depending on that, they assign a retention period for the data; for example, the migration probability and the hobbies of the subject must be deleted immediately, while the account transactions must remain in the system 10 years after the account gets inactive. This is an example of how the data cannot be deleted by demand only, but also, because of the requirements of the context where the Big Data system performs its operations.

## **47.12. See Also**

Authenticator. When an active entity like a user or system (subject) identifies itself to the system, how do we verify that the subject intending to access the system is who it says it is? Present some information that is recognized by the system as identifying this subject. After being recognized, the requestor is given some proof that it has been authenticated. This pattern is used to authenticate the Requester and the Chief Data Officer in the system.

Role-Based Access Control. Describe how to assign rights based on the functions or tasks of users in an environment in which control of access to computing resources is required. The RBAC pattern is needed to establish the rights that the different roles have over the data.

Security Logger/Auditor. How can we keep track of user's actions in order to determine who did what and when? Log all security-sensitive actions performed by users and provide controlled access to records for Audit purposes. Deleting individual data is a very sensitive operation that must be registered in a log file.

## 47.13. References

- [1] Villaronga, E. F., Kieseberg, P., & Li, T. (2018). Humans forget, machines remember: Artificial intelligence and the right to be forgotten. *Computer Law & Security Review*, 34(2), 304-313.
- [2] Mohammed, J., Khan, C. (2018). Big Data Deidentification, Reidentification and Anonymization. Retrieved from <https://www.isaca.org/Journal/archives/2018/Volume-1/Pages/big-data-deidentification-reidentification-andanonymization.aspx>
- [3] GDPR deep dive—how to implement the ‘right to be forgotten.’ (2017, November 15). Retrieved June 5, 2018, from <https://www.bankinghub.eu/banking/finance-risk/gdpr-deep-dive-implement-right-forgotten>

## 47.14. Source

Moreno, J., Fernandez, E. B., Fernandez-Medina, E., & Serrano, M. A. (2018, October). Neuralyzer: a security pattern for the right to be forgotten in big data. In *Proceedings of the 25th Conference on Pattern Languages of Programs* (pp. 1-9).

# 48. Mutual Authentication

## 48.1. Intent

In a grid environment, parties may need to prove to each other that they are who they say they are, before proceeding with their actual business. This is known as mutual authentication and is accomplished by the parties exchanging their credentials.

## 48.2. Example

A user at site A requests to a simulation service to run a simulation at site B. The simulation may access sensitive data on the user's behalf. Before any simulation actions, the user and the simulation service must first establish two-way trust between each other, since typically, these two sites are in different security domains.

If we allowed the simulation service to serve any user anonymously, we run the risk that simulations could be executed by users with no authorization to access sensitive data at site B, either maliciously or by accident. Similarly, the user wants to first authenticate the simulation service before entrusting it with her simulation request.

## 48.3. Context

A grid environment in which two parties need to establish mutual trust between each other. Credential pattern is used as a precondition to identify the parties.

## 48.4. Problem

Two parties must establish mutual trust between each other, before any further operations can happen. Refer to the two parties as Alice and Bob. Alice requests a service from Bob, assuming she is authorized by Bob to do so. The initial mutual trust must be established. Then Alice is sure that she is communicating with Bob who can provide the service she is requesting, and Bob ensures that it in fact is Alice attempting access and not a third party not authorized to the service.

One solution would be for Bob and Alice to authenticate each other with a username and password as an identity. But this would also require Alice and Bob to know each other's passwords prior to authentication, and somehow keep a copy of the secret password. This copy leaves a security, because once an attacker obtains the passwords, they can steal the identities of Alice and Bob. Thus, the more entities with whom a party communicates, the less secure its identity. Another disadvantage of this solution is the overhead caused by managing copies of the passwords. Therefore, a precondition of this pattern is Credential. More specifically, X.509 certificates from the public key infrastructure (PKI) can be used to implement the Credential pattern.

To solve this problem, the following forces must be balanced:

- The solution should be easy to manage, but to gain this ease of use we cannot sacrifice protection against identity thieves and eavesdroppers.

- The identity of each entity should be persistent in all sessions initiated by it.

## 48.5. Solution

Each entity (verifier) verifies the other entity's (verified) identity by validating the encryption of the verified with her public certificate. Based on public key technology, an entity holds a pair of public/private keys. The public key is contained in the entity's certificate and the private key is only accessible to the entity. If a message is encrypted with a private key, only the corresponding public key can decrypt it.

The parties share a randomly generated message. At either side, the verified encrypts the message with its private key, and the verifier decrypts the encryption using the public key in the certificate of the verified. If the decrypted message matches the random message on hand, the verifier can authenticate the verified. The certificate signed by a certificate authority (CA) represents who the verified is. The CA is the trust anchor of the verifier. As long as the verifier trusts the CA, it would trust the identity of the verified represented by her certificate in the authentication.

## 48.6. Structure

Figure 93 below shows the structure of this pattern. Alice and Bob want to mutually authenticate each other. Alice holds a certificate Cert A which identifies her. Cert A is signed by certificate authority CA A. Bob also holds a certificate Cert B as his identity signed by certificate authority CA B. Alice verifies Cert B to be able to trust Bob, based on her trust to CA B. Bob verifies Cert A to be able to trust Alice, also based on his trust to CA A. Mutual authentication is then established after the two-way trust.

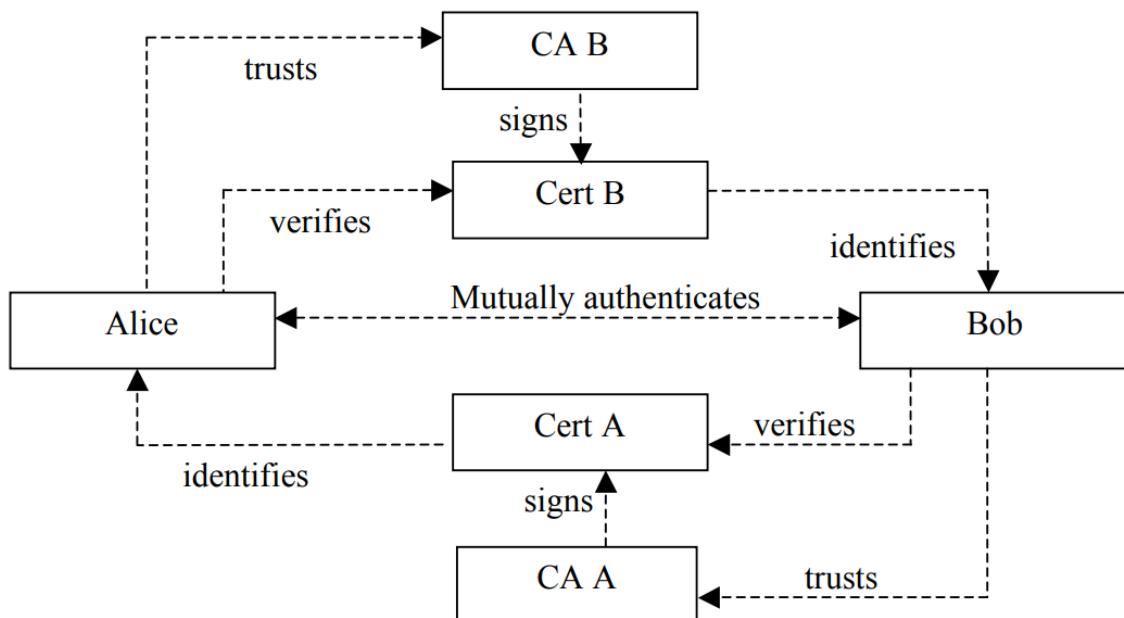


Figure 93: Structure of the Mutual Authentication pattern

## **48.7. Dynamics**

## **48.8. Implementation**

## **48.9. Example Resolved**

The user has a X.509 certificate issued and signed by a CA. So does the simulation. The user and the simulation can both verify each other's certificate and mutual authenticate as described above. Through the authentication, the simulation ensures the user's identity and can further check if she is authorized to access the data. Meanwhile, the user is also sure the simulation service is the one she is requesting.

## **48.10. Consequences**

The following benefits may be expected from applying this pattern:

- The implementation of public key cryptography makes the authentication reliable against adversaries.
- X.509 certificates are widely used in many organizations. The existing public key frameworks such as PKI have taken good care of the certificate issuing and management work.
- X.509 certificates are distinct based on their Subject Name and/or serial number, and no two certificate holders can be authenticated as the same entity, so the certificates are persistent and uniform in all sessions. Therefore, authentication and authorization policies can be defined on them for granting access.
- No one can forge the certificates since the CA private key for signing them is only kept secret for the CA.

The following liabilities may arise from applying this pattern:

- It is significantly more expensive to implement than a lightweight mechanism based on passwords.
- To authenticate an entity, the user must trust the CA that signs its certificate. This “cold start” problem can be solved by using a CA hierarchy. A root CA signs the certificates of all the involved CAs, so parties now only need to both trust the root CA. There can also be several trusted root CAs.
- Mutual authentication is slower compared to one-side only authentication and anonymity.

## **48.11. Known Uses**

Mutual Authentication appears before grid computing emerges, so it is not limited to grids. It is used on client/server applications where clients hold certificates as identities. In Large Hadron Collider (LHC) Computing Grid (LCG) project [1], the authentication layer is based on

Globus Grid Security Infrastructure (GSI). Each grid user (and grid service) is in possession of an X509 format certificate. Certificates are issued by a network of Certification Authorities (CA) approved by the project. LCG enables services to operate on behalf of the user (such as for batch job execution when the user is not present, or for provisioning of data from one storage location to another), and Mutual Authentication is required between the services and the user before delegation relationships are established [2].

## 48.12. See Also

Mutual Authentication refines Known Partner pattern. Credential pattern is used to identify the parties who want to mutual authenticate each other.

## 48.13. References

- [1] LCG. (n.d.). homepage. LHC Computing Grid Project. <https://wlcg.web.cern.ch/LCG/>
- [2] Neilson, I. & LCG Deployment Group. (2004). CERN. Jisc.  
[http://www.jisc.ac.uk/uploaded\\_documents/CERN\\_%20PositionPaper.doc](http://www.jisc.ac.uk/uploaded_documents/CERN_%20PositionPaper.doc)

## 48.14. Source

Lu, M., & Weiss, M. (2008). Patterns for grid security. In *Mini-PLoP at conference on object-oriented programming systems, languages, and applications, OOPSLA*.

# 49. Short Lifespan

## 49.1. Intent

When a party delegates its rights to another party by issuing a proxy certificate, it may not trust that party indefinitely, and the proxy credentials may be compromised. Limiting the proxy certificate lifetime to be short can mitigate the effects of a compromise or misuse.

## 49.2. Example

A user Alice at site A delegates her rights to a simulation service running at site B by issuing a proxy certificate and the corresponding private/public key pair to the simulation service. The simulation service may request computational resources at site C owned by Charlie for 12 hours for Alice's computation use. An adversary Malice attacks site B and gets a copy of the private key related to the proxy certificate. Malice may in turn use the resources at site C for days for a hacker program, but everybody thinks its Alice using the resources.

## 49.3. Context

A grid in which parties can be authenticated as acting on other's behalf by using Credential Delegation.

## 49.4. Problem

When a party delegates its rights to another party by issuing a proxy certificate, it may not trust that party indefinitely, and the proxy credentials may be compromised. Allowing the grantee to act unconditionally on the grantor's behalf is risky. The grantor may trust the grantee for the particular task only, but it is hard to restrict the grantee from using the proxy certificate for other tasks accidentally or deliberately. Some other users of the grantee's machine may compromise the proxy certificate in the future to carry out some mischievous deeds on the grantor's behalf.

For easy usage, the private key proxy credentials are not protected by password or PIN as end entity certificate credentials, as described in Single Sign-on. One solution is to add protection on the private key proxy credentials, so it is hard for adversaries to compromise the proxy credentials. But it greatly complexes the delegation processes and makes Single Sign-on impossible.

To solve this problem, the following forces must be balanced:

- The solution must limit the grantees from misusing the proxy certificates without sacrificing the easiness of delegation.
- Even if the credential is compromised, the adversaries cannot make much use of the credentials

## **49.5. Solution**

Limit the lifetime for the proxy certificate to be short. When creating a X.509 proxy certificate, the proxy issuer can set a validity period by adding an expiry time. For every delegation task, the proxy issuer estimates how long it is expected to last and set the expiry time accordingly. Usually, a proxy certificate lifespan is set to be as short as a few hours. The default setting in GSI is 12 hours. Once the proxy certificates expire, the delegation is no longer trusted by anybody, so the tasks invoked with misused or compromised certificates stop.

## **49.6. Structure**

## **49.7. Dynamics**

## **49.8. Implementation**

## **49.9. Example Resolved**

When signing the proxy certificate for the simulation service, Alice sets an expiry time to be 12 hours since the issuing. When the simulation is done using the resources at site C, the proxy certificate expires. Even if Malice compromises the proxy certificate before the expiry time, he can only use it for a small amount of time. The effects of a compromise or misuse are limited to minimum.

## **49.10. Consequences**

The following benefits may be expected from applying this pattern:

- The grantees are limited from misusing the proxy certificates without sacrificing the easiness of delegation.
- Even if the credentials are compromised, the adversaries cannot make much use of them.

The following liabilities may arise from applying this pattern:

- The pattern relies on setting a suitable lifetime of the proxy certificates, but in fact it is very difficult to predict a job requiring time in a distributed grid environment. Because a VO component may leave the VO unexpectedly due to network problems or user sign off, and it is hard to estimate the time waiting in the queues for required resources.
- Short lifetime causes issues for long-running jobs. See Credential Renewal pattern.

## **49.11. Known Uses**

The Large Hadron Collider (LHC) Computing Grid (LCG) project (using EGEE) is built on Globus Toolkit, so it follows the setting in GSI to limit the proxy certificate lifetime to be short. The default is 12 hours as in GSI. [1]

## **49.12. See Also**

The pattern is based on Credential Delegation for issuing proxy certificates.

## **49.13. References**

- [1] Neilson, I. & LCG Deployment Group. (2004). CERN. Jisc.  
[http://www.jisc.ac.uk/uploaded\\_documents/CERN\\_%20PositionPaper.doc](http://www.jisc.ac.uk/uploaded_documents/CERN_%20PositionPaper.doc)

## **49.14. Source**

Lu, M., & Weiss, M. (2008). Patterns for grid security. In *Mini-PLoP at conference on object-oriented programming systems, languages, and applications, OOPSLA*.

# 50. Gridmap Authorization

## 50.1. Intent

A user (subject) requesting resources (object) in a different domain should be authorized or denied access according to his privilege. A gridmap can be used to map a global grid user subject to a predefined local identity and thus mapped to a local access control policy defined for the identity.

## 50.2. Example

A user at site A requests to run a simulation service on a service container at site B. The simulation service may – on the user’s behalf – need to access a resource on another server at Site C. Typically, these three sites are in different security domains, and may use different local security mechanisms (Windows, Unix). Figure 94 below summarizes this scenario.

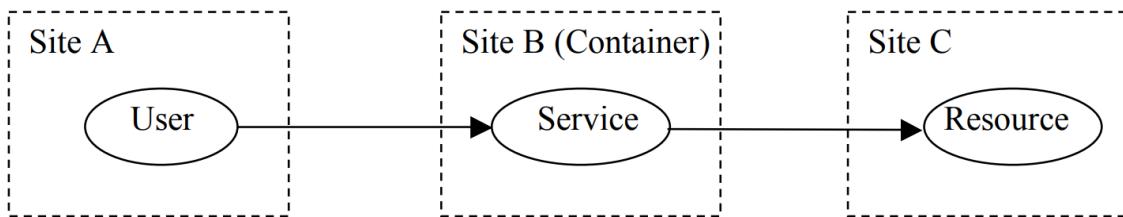


Figure 94: Gridmap Authorization example problem

## 50.3. Context

A grid in which a user from one domain needs to access resources on another domain. The user is using Credential Delegation to delegate his rights to a service.

## 50.4. Problem

A user (subject) requesting resources (object) in a different domain should be authorized or denied access according to his privilege. To the good of the resource providers, all access control decisions are made locally to the resource owner by its system administrators. User domain and resource domain are subject to different security, management, and usage policies. Because the resource pool is large and dynamic, it is impractical to modify every local resource to accommodate interdomain access. So, the problem is actually to map the user from a foreign domain to the authorization policies for access control to local resources.

One solution would be to define an API that can be used by grid containers to match the user privileges to all the resources, but it has an obvious drawback, that is every grid container needs to implement the API. Besides, the containers have to obtain a copy of the access control policies for all the users to access every possible resource. For a big number of users and resources, the user-resource combination of possible polices could be a huge number.

To solve this problem, the following forces must be balanced:

- The parties may use any kind of system (Windows, UNIX), so the solution must be platform independent.
- The solution application code should be application independent.
- It should be easy to grant, modify and deprive the access rights of a grid user in response to changes in his duties or responsibilities.
- The access control process should be transparent to the user requesting resources.

## 50.5. Solution

Use a gridmap to map a global grid user subject to a predefined local identity and thus mapped to a local access control policy defined for the identity. The gridmap is basically an ACL (Access Control List) of grid users that are allowed access to the resource. It maps the grid users by their distinguished name from the X.509 certificates to a local username. See the sample gridmap in Figure 95. Obviously, a gridmap is just a text-based mapping table, so it works on any kind of platform and any application.

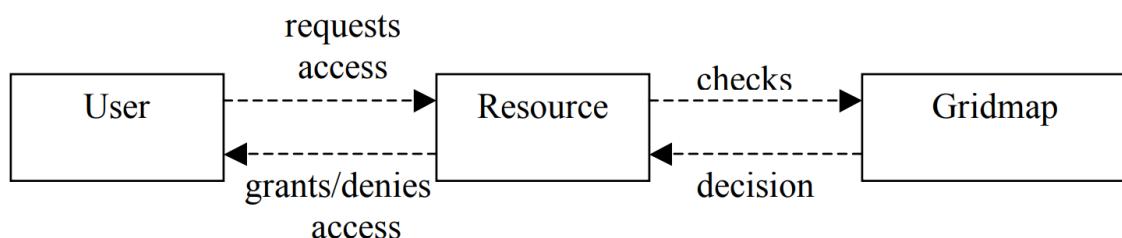
If the user is not listed in the file, access to the resource is denied. Once the user is mapped to a local identity, the user's privilege to the local resources relies solely on local policy management and enforcement mechanisms. Gridmap files are maintained manually by the resource system administrators.

# Distinguished name	Local
#	username
"/C=US/O=Globus/O=NPACI/OU=SDSC/CN=Rich Gallup"	rpg
"/C=US/O=Globus/O=NPACI/OU=SDSC/CN=Richard Frost"	frost
"/C=US/O=Globus/O=USC/OU=ISI/CN=Carl Kesselman"	u14543
"/C=US/O=Globus/O=ANL/OU=MCS/CN=Ian Foster"	itf

*Figure 95: A sample gridmap file*

## 50.6. Structure

Figure 96 below demonstrates the structure of this pattern. The user or the service delegating the user sends an access request to a resource. The resource then checks a local gridmap to see if the user's global name is mapped to a predefined local username. The result is the decision to access the resource. The resource will then grant or deny access accordingly.



*Figure 96: Structure of the Gridmap Authorization pattern*

## 50.7. Dynamics

## 50.8. Implementation

## 50.9. Example Resolved

The user has a X.509 certificate as an identity. Figure 97 summarizes how the example is resolved. The distributed name (DN) of the certificate serves as a gridwide name. The simulation service at site B forwards the DN to site C on the user's behalf. If the user has access to the resource, the DN is mapped to a username local to site C according to the gridmap. Access will be granted or denied according to the local policy for that username.

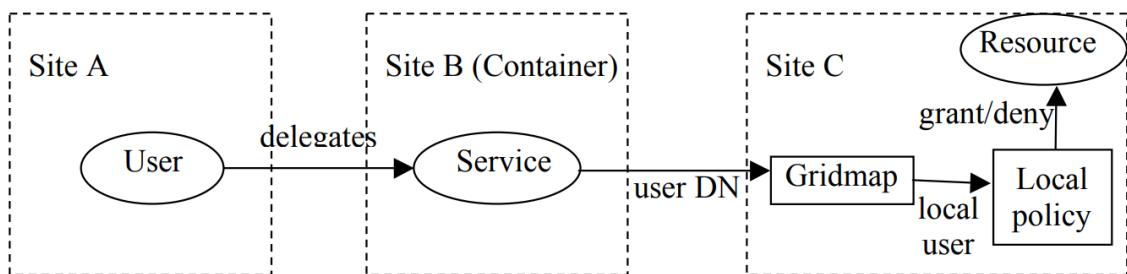


Figure 97: Gridmap Authorization example resolved

## 50.10. Consequences

The following benefits may be expected from applying this pattern:

- Authorization decision making and enforcement are all at object local, so the solution applies to any kind of platform.
- Access control jobs are left to the object local system that has the best knowledge of the local policies.
- The resource system administrators can grant and modify access rights as do so to local users, they can also deprive access by simply remove the corresponding mapping from the gridmap.
- The authorization processes are transparent to the grid users as they do not require user interactions.

The following liabilities may arise from applying this pattern:

- Every participating site must keep and maintain a grid map file.
- Each personnel or policy change requires changing policy at all participating sites.
- Any failure in changing policy at any site causes inconsistency in the VO policy.
- The pattern performance greatly depends on other authorization solutions implemented at the object local system, because the policy decision point (PDP) and policy enforcement point (PEP) are all at object local.

## 50.11. Known Uses

The mechanism of keeping a gridmap file at resources exists in Globus Toolkit since its first generation. A first interim solution adopted by the EDG [1] consisted in using Lightweight

Directory Access Protocol (LDAP) servers to manage the authorization information at the VO level and developing the mkgridmap utility to allow resources to automatically download this information and generate the “gridmap-file” [2]. The Virtual Organization Membership Service (VOMS) [3] is an evolution of the gridmap generating and management by providing a fine-grained control of the access.

## 50.12. See Also

Gridmap Authorizaiton refines the Authorization pattern. Credential Delegation is usually used for a service to request access on the grid user’s behalf.

## 50.13. References

- [1] Hoschek, W., Jaen-Martinez, J., Samar, A., Stockinger, H., & Stockinger, K. (2000, December). Data management in an international data grid project. In *International Workshop on Grid Computing* (pp. 77-90). Springer, Berlin, Heidelberg.
- [2] Alfieri, R., Cecchini, R., Ciaschini, V., dell’Agnello, L., Frohner, A., Lőrentey, K., & Spataro, F. (2005). From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Generation Computer Systems*, 21(4), 549-558.
- [3] Alfieri, R., Cecchini, R., Ciaschini, V., Gianoli, A., Spataro, F., Bonnassieux, F., ... & Lhorentey, K. (2003). Managing dynamic user communities in a grid of autonomous resources. *arXiv preprint cs/0306004*.

## 50.14. Source

Lu, M., & Weiss, M. (2008). Patterns for grid security. In *Mini-PLoP at conference on object-oriented programming systems, languages, and applications, OOPSLA*.

# 51. Abstract IDS

## 51.1. Intent

The ABSTRACT IDS pattern allows monitoring of all traffic as it passes through a network, and its analysis it to detect possible attacks and trigger an appropriate response.

## 51.2. Example

Our company has a firewall to control traffic from the Internet. However, we are still plagued by viruses and other attacks that penetrate the firewall. The attacks can be existing attacks or new attacks. We need to improve our defense against such attacks.

## 51.3. Context

Nodes for local systems that need to communicate with each other using the Internet or another insecure network.

## 51.4. Problem

An attacker may try to infiltrate our system through the Internet and misuse our information by reading or modifying it. We need to know when an attack is happening and take appropriate response.

The solution to this problem must resolve the following forces:

- Communication. The system is usually more secure if we have a closed network. However, in today's world it is better and more realistic to use the Internet or other insecure network to reduce costs, which may subject our network to security threats.
- Real time behavior. Attacks should be detected before the attack completes its purpose, so that we can preserve our assets and save time and money. It is difficult to detect an attack when it is happening, but such detection is imperative if we are to react timely and appropriately.
- Incomplete security. Security measures such as encryption, authentication and so on may not protect all our systems, because they do not cover all possible attacks.
- Non-suspicious users. Protecting our system through a firewall is quick and easy. However, request coming from a non-suspicious address (that is, one permitted by a firewall) could still be harmful and should be monitored further.
- Flexibility. Hard coding the type of attack can be done easily. But it will be hard and time-consuming to adapt to attack patterns that change constantly

## 51.5. Solution

Each request to access the network is analyzed to check whether it conforms to the definition of an attack. If we detect an attack, an alert is raised, and some countermeasures may be taken.

The ABSTRACT IDS pattern defines the basic features of any intrusion detection system (IDS). An abstract pattern defines only fundamental, implementation-independent functions and threats [1]. Concrete patterns add functionalities and threats and consider the characteristics of their specific concrete environment. In this case, the abstract functions are realized by concrete IDSs that operate based on known attack signatures or based on abnormal behavior or anomaly in the network: SIGNATURE-BASED IDS or BEHAVIOR-BASED IDS. We present here patterns for all these three types of IDS.

Figure 98 shows the typical placement of an IDS in a network, complementing a firewall. The firewall filters requests for services, and the IDS further checks for suspicious patterns in request sequences. If a suspicious pattern is detected, the network operator is alerted, and the firewall may block some or all traffic.



Figure 98: Possible placement of network IDS to complement a firewall

## 51.6. Structure

Figure 99 shows the class diagram for the ABSTRACT IDS pattern. A Client requests some service from the Server. The IDS intercepts this request and sends it to an EventProcessor. The EventProcessor processes the event so that the AttackDetector can analyze the event and implement some method of detection using information from AttackInformation. When an attack is detected, a Response is created.

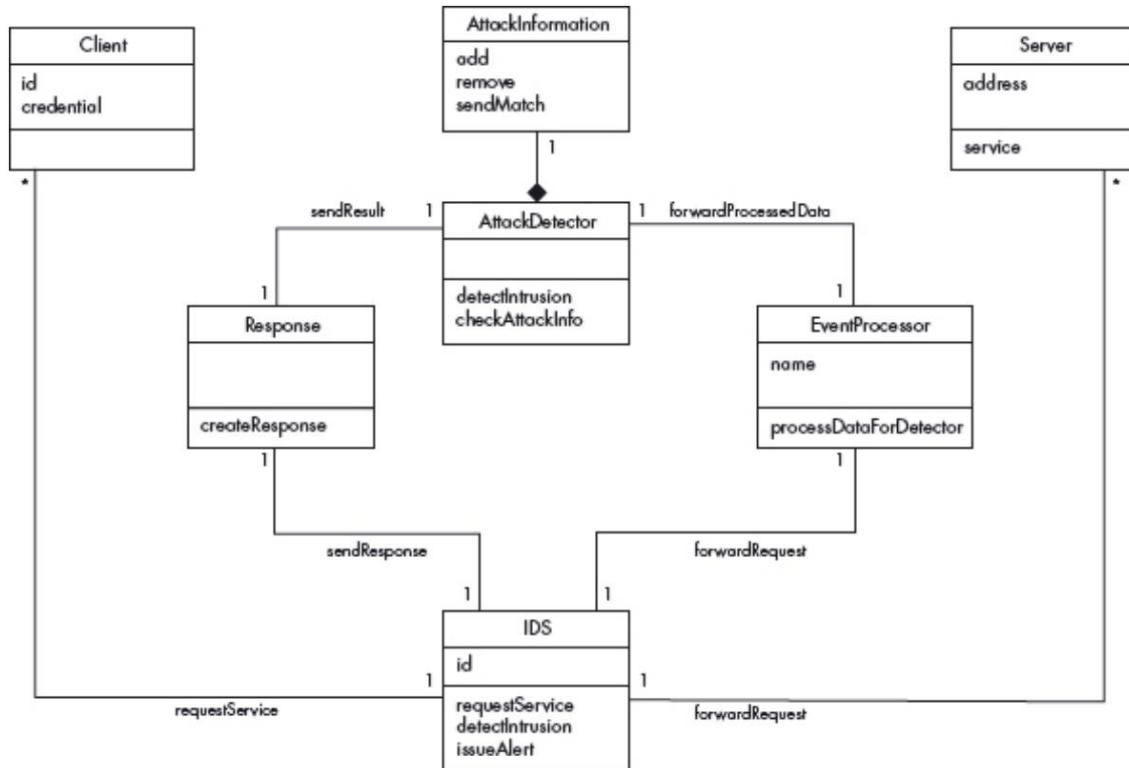


Figure 99: Class diagram for the ABSTRACT IDS pattern

## 51.7. Dynamics

We describe the dynamic aspects of the ABSTRACT IDS pattern using a sequence diagram for the following use case.

Use Case:	Detect an Intrusion – Figure 100
Summary	The Client requests a service from the Host. The IDS intercepts the message and checks whether the request is an attack or not and raises a response.
Actors	Client, Server.
Precondition	We have attack information available.
Description	<ol style="list-style-type: none"> <li>4. A Client makes a service request to the Host.</li> <li>5. The IDS send the request event to an EventProcessor.</li> <li>6. The EventProcessor processes the event data so that the AttackDetector can interpret the event.</li> <li>7. The AttackDetector tries to detect whether this request is an attack or not by comparing with the available information in the AttackInformation.</li> <li>8. If an attack is detected, a Response is created. <ul style="list-style-type: none"> <li>• The AttackInformation may not be able to detect an attack (a false negative).</li> <li>• The AttackInformation may indicate an attack when no attack is present (a false positive).</li> </ul> </li> </ol>
Alternate Flows	
Postcondition	If an attack is detected, suitable preventive measures may be applied.

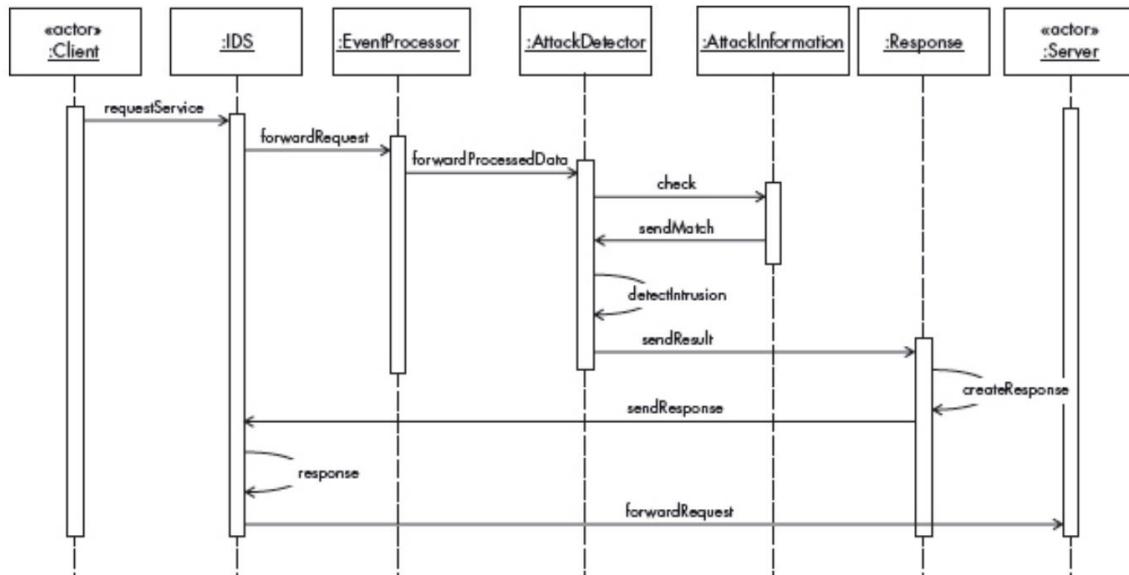


Figure 100: Sequence diagram for the use case 'Detect an intrusion'

## 51.8. Implementation

We need to create a database with attack information so that we can check against this database and decide whether an attack is happening. The incoming event is compared against the database and a decision is made whether the incoming event is an attack or not. The concrete versions of this pattern use different types of information to detect attacks.

The Common Intrusion Detection Framework (CIDF) is a working group created by DARPA in 1998 that is mainly oriented towards creating a common framework in the IDS field. CIDF defined a general IDS architecture based on the consideration of four types of functional modules as shown in Figure 101 [2].

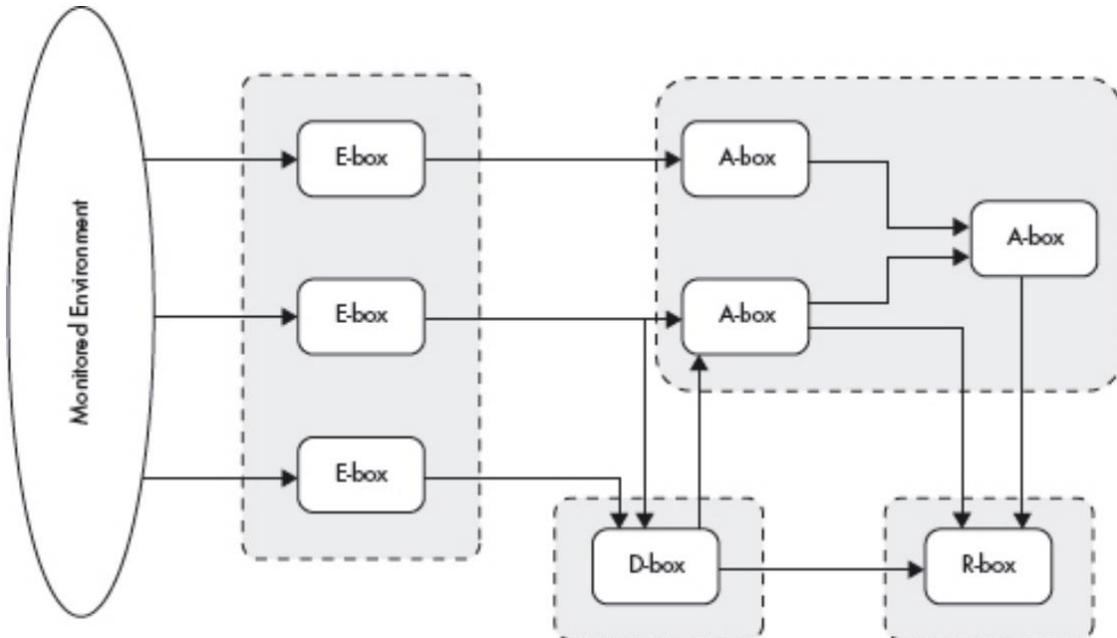


Figure 101: General CIDF architecture for IDS systems (from [Gar09])

- E blocks ('event-boxes'). This block is composed of sensor elements that monitor the target system, thus acquiring information events to be analyzed by other blocks.
- D blocks ('database-boxes'). These are elements intended to store information from E blocks for subsequent processing by A and R boxes.
- A blocks ('analysis-boxes'). Processing modules for analyzing events and detecting potential hostile behavior.
- R blocks ('response-boxes'). The main function of this type of block is the execution, if any intrusion occurs, of a response to thwart the detected menace.

## 51.9. Example Resolved

## 51.10. Consequences

The ABSTRACT IDS pattern offers the following benefits:

- Communication. If we can detect most attacks, we can safely use the Internet or other insecure networks to access other systems.
- Real time behavior. Attacks can be detected when the attack happens and the system alerted, which saves both time and money in recovery measures, and may prevent misuse of assets. Attacks can be detected in real time if they have sufficient and appropriate information.

- Incomplete security. The IDS provides be an added layer of security in addition to encryption, authentication and so on.
- Non-suspicious users. A request coming from a non-suspicious address (permitted by a firewall) is further inspected and analyzed.
- Flexibility. The detection information can be modified to include new attacks or new behavior.

The pattern also has the following potential liabilities:

- Some attacks may be so fast that it may be hard to recognize them in real time.
- Attack patterns are closely tied to a given environment (operating system, hardware architecture and so on) and cannot be applied easily to other systems. This means we need to define detection information tailored to an environment.
- There is some overhead in the addition of IDSs to a system.
- The concrete versions of these systems have additional liabilities.

## **51.11. Variants**

- IDS can be either behavior (rule) based or can be based on anomalies (abnormal behavior). There are significant differences in the use and effectiveness of these two approaches. See SIGNATURE-BASED IDS pattern and BEHAVIOR-BASED IDS pattern.
- A hybrid model of both the signature based and behavior-based IDS together is now available: a behavior-based IDS detects the anomalies in traffic and then compares the anomalies with an attack signature in a signature-based IDS.
- According to the resources they monitor, IDS systems are divided into two categories: host-based IDS systems and network-based IDS systems. Host-based IDS systems are installed locally on host machines and evaluate the activities and access to key servers within the host. Network-based IDS systems inspect the packets passing through the network [3].

## **51.12. Known Uses**

NID is a freely available hybrid intrusion detection package. It monitors network traffic and scans for the presence of known attack signatures, as well as deviations from normal network behavior [4].

## **51.13. See Also**

Firewall patterns can be added to complement the IDS. Firewalls usually deny requests made by unknown addresses. They can protect against attacks coming from distrusted sources and can block the addresses from where an attack originates. The response class could be implemented as a Strategy pattern [5].

## **51.14. References**

[1] Fernandez, E. B., Washizaki, H., & Yoshioka, N. (2008, October). Abstract security patterns. In *Proceedings of the 15th Conference on Pattern Languages of Programs* (pp. 1-2).

- [2] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2), 18-28.
- [3] Depren, O., Topallar, M., Anarim, E., & Ciliz, M. K. (2005). An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert systems with Applications*, 29(4), 713-722.
- [4] C. Grace. (n.d.). Understanding Intrusion Detection Systems. IT Journalist PC Network Advisor – Tutorial. <http://www.techsupportalert.com/pdf/t1523.pdf>
- [5] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

## 51.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 52. Abstract Virtual Private Network

## 52.1. Intent

The ABSTRACT VIRTUAL PRIVATE NETWORK pattern describes how to set up a secure channel between two endpoints using cryptographic tunneling, with authentication at each endpoint. An endpoint is an interface exposed by a communicating unit (user site or network).

## 52.2. Example

Our company has employees all over the world. Because of cost, we decided to use the Internet to communicate. However, we are having problems because their orders are hacked, and the attackers get access to customers' credit card numbers and other details. Our staff want to be sure they are talking to other employees and must be able to send secure messages to discuss prices, discounts and so on.

## 52.3. Context

Users scattered in many fixed locations, who need to communicate securely with each other using the Internet or some other insecure network. In such a network attackers may intercept messages and try to read, modify, or replay them.

## 52.4. Problem

In today's world, companies have offices all over the world and a lot of people work remotely. They need a secure connection to other specific nodes so that confidential work can be performed securely. Their communication can be intercepted by attackers, who may get access to private information and may even modify the messages. How can we establish a secure channel for the end users of a network so that they can exchange messages through fixed points using an insecure network?

The solution to this problem must resolve the following forces:

- We need to use the Internet or other insecure networks to reduce cost, but in turn subjecting our network to numerous threats.
- Only registered users should access the institution's endpoints.
- We need to make sure that the users with which we are communicating are the right ones, otherwise confidentiality may be compromised.
- The number of users remotely connected may be growing: the system should be scalable.
- Because different users or institutions require different levels of security, the system should be flexible enough to accommodate different ways of providing security and different degrees of security.
- In some cases, we also need to support authorization to access specific resources in the endpoints.
- The system should be easy to set up and use, otherwise users and administrators will not want to use it.

- The system should not impose a heavy performance penalty, otherwise it will not be used all the time.
- The pattern should be adaptable to the needs and constraints of different protocol layers.

## 52.5. Solution

Protect communications by establishing a cryptographic tunnel between endpoints on one of the layers of the communication protocol. Add authentication functions at each endpoint.

Figure 102 shows the case in which site A is talking to site B over the Internet using routers R1 and R2, respectively. The secure connection is established through one of the Internet layers.

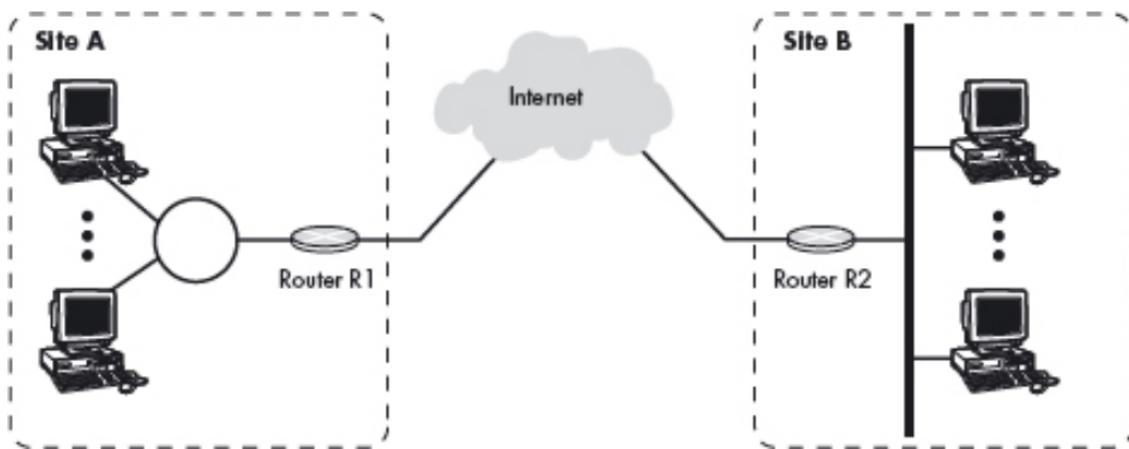


Figure 102: Two sites communicating through the Internet [For04b]

## 52.6. Structure

Figure 103 shows the class diagram for the ABSTRACT VIRTUAL PRIVATE NETWORK pattern, in which a SecureChannel can be established between a Client and a NetworkEndPoint. EndPoints communicate with other EndPoints. A user is authenticated by an AUTHENTICATOR pattern. AUTHENTICATOR and Secure Channel are patterns, composed typically of several classes, and are shown using the UML symbol for package.

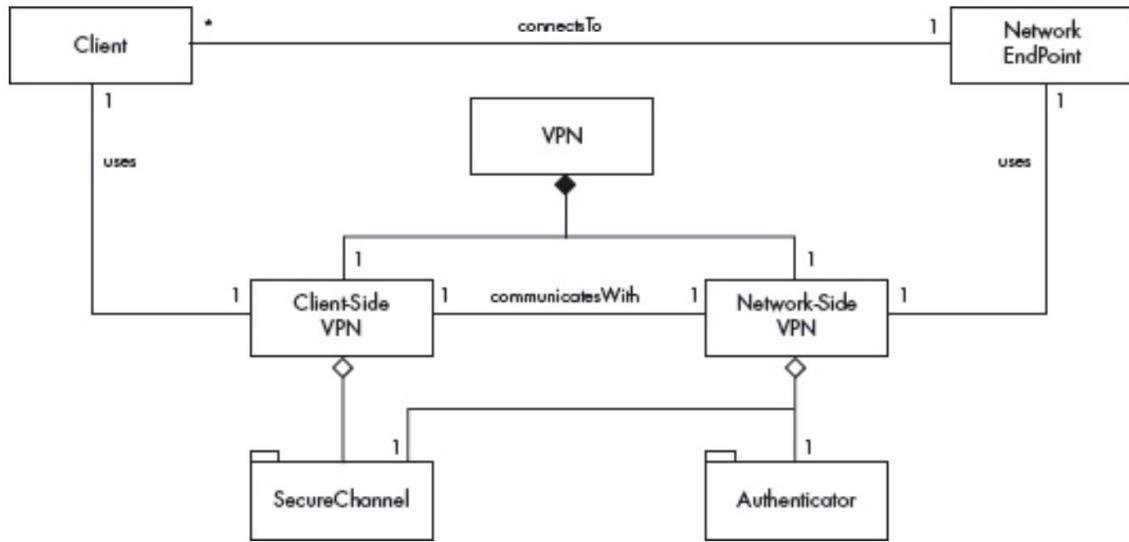


Figure 103: Class diagram for the ABSTRACT VIRTUAL PRIVATE NETWORK pattern

## 52.7. Dynamics

The sequence diagram of Figure 104 shows a use case in which an end user tries to access an endpoint in a network, endPoint2, from another endpoint, endPoint1. The Authenticator at endPoint1 authenticates the user. The Authenticator creates a Token as proof of authentication, which can be used to establish a SecureChannel. This channel allows secure access to endPoint2.

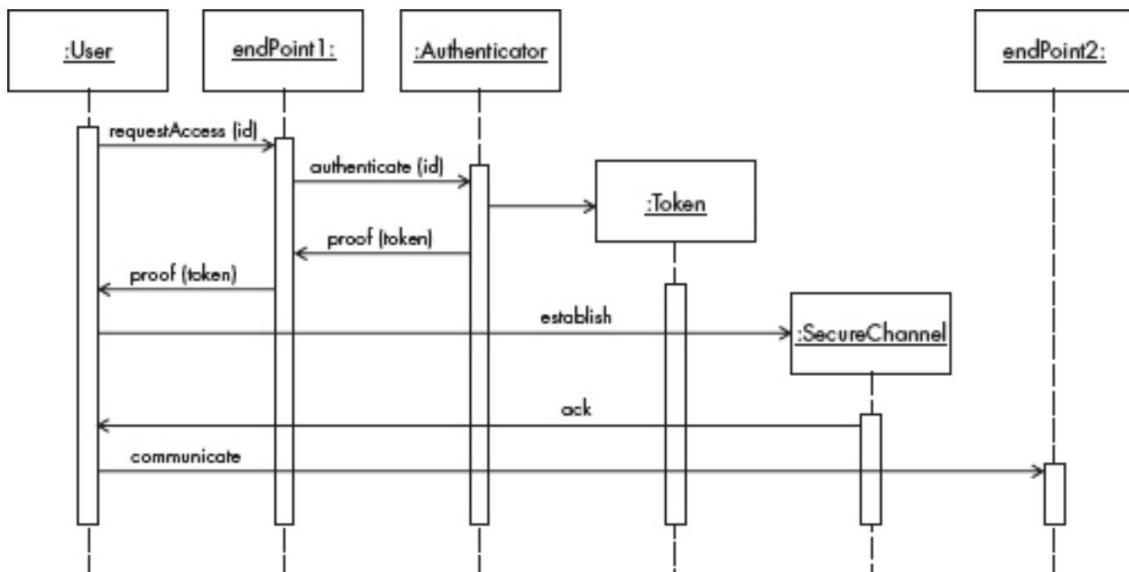


Figure 104: Sequence diagram for accessing an endpoint

## 52.8. Implementation

First define the endpoints which the VPN will reach. Consider the architectural layer where the communications should be secured, according to the needs of the applications.

After this decision, use the concrete VPN pattern at the corresponding level, IP, or TCP. See the corresponding patterns (IPSEC VPN, and TLS VPN) for help in making this decision [1]. Both endpoints must share a public key system for authentication and must have appropriate software packages running on them.

## 52.9. Example Resolved

Now the users can be authenticated at the endpoints, which ensures that they are communicating with their own employees. User messages are now protected from external attacks when sent over the secure channel.

## 52.10. Consequences

The ABSTRACT VIRTUAL PRIVATE NETWORK pattern offers the following benefits:

- We can use the Internet or other insecure networks to reduce cost.
- Cryptography can protect our messages from being read or modified by attackers.
- Authentication at endpoints ensures that only registered users can access the secure channel.
- Mutual authentication between end users is possible.
- The system can accommodate new links for new users by just replicating the access software.
- We can use any cryptographic algorithm to establish the secure channel, which allows us to make trade-offs between security and cost.
- We can add authorization to access specific resources at each endpoint.
- We can add a logging system for the users logging in at the endpoints for use in audits.
- The VPN is transparent to the users, who are authenticated by their local endpoints.
- The VPN is a client-server architecture that is easy to configure.
- We can have different versions of the pattern that can use the specific features of each protocol layer.

The pattern also has the following potential liabilities:

- If the VPN connection is compromised, the attacker could get full access to the internal network. Authorization can restrict this access, however.
- Because of encryption, VPN traffic is invisible to IDS monitoring. If the IDS probe is outside the VPN server, as is often the case, then the IDS cannot see the traffic within the VPN tunnel. Therefore, if a hacker gains access to the end node of the VPN, they can attack the internal systems without being detected by the IDS.
- In the case of VPN with a private end user, the remote computer used by the private user is vulnerable to outside attacks, which in turn can attack the network to which it is connected.
- There is some overhead in the encryption process.

## **52.11. Variants**

Virtual private networks can be established at the application layer (XML or application VPN); TLS (SSL) VPNs are established at the transport layer. IPSec VPNs are established at the IP layer [1].

## **52.12. Known Uses**

Citrix provides a site-to-site SSL VPN connection for remote users to log into the secure network, as well as access applications on the company (secure) network [2].

Cisco VPN uses an IPSec VPN and provides authorization [3].

Nokia provides a VPN connection for Nokia mobile users.

## **52.13. See Also**

Firewalls can be added to each endpoint to filter inputs. They can protect against some types of attacks coming from untrusted sources.

IDS patterns can be added in each of the network layers to detect attacks in real time.

The VPN uses the Secure Channel pattern that in turn uses cryptography to protect its messages.

The Authenticator pattern can authenticate users and nodes.

Access Control/Authorization can be added at each site to control access to specific resources.

## **52.14. References**

[1] Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

[2] Citrix. (n.d.). Explore Citrix Products for Enterprise and Medium Business. Retrieved June 21, 2010, from <http://www.citrix.com/English/ps2/products/product.asp?contentID=15005>.

[3] Cisco. (n.d.). Cisco Systems: Products and Technologies > Cisco Intrusion Detection. Retrieved from <http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/>.

## **52.15. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 53. Asymmetric Encryption

## 53.1. Intent

Asymmetric encryption provides message confidentiality by keeping information secret in such a way that it can only be understood by intended recipients who have the access to the valid key. In asymmetric encryption, a public/private key pair is used for encryption and decryption respectively.

## 53.2. Example

Alice wants to send a personal message to Bob. They have not met each other to agree upon a shared key. Alice wants to keep the message secret, since it contains personal information. Eve can intercept Alice's messages and may try to obtain the confidential information.

## 53.3. Context

Applications that exchange sensitive information over insecure networks.

## 53.4. Problem

Applications that communicate with external applications interchange messages that may contain sensitive information. These messages can be intercepted and read by imposters during transmission. How can we send sensitive information securely over insecure channels?

The solution to this problem must resolve the following forces:

- Confidentiality. Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information the message contains. Hiding information also makes replaying of messages by an attacker harder to perform.
- Reception. The hidden information should be revealed conveniently to the receiver.
- Protocol. We need to apply the solution properly, or it will not be able to withstand attacks (there are several ways to attack a method of hiding information).
- Performance. The time to hide and recover the message should be acceptable.
- Key distribution. Two parties may want to communicate to each other, but they have not agreed on a shared key: we need a way to send messages without establishing a common key.

## 53.5. Solution

Apply mathematical functions to a message to make it unreadable to those that do not have a valid key.

This approach uses a key pair: private and public key. The sender encrypts (E) the message (M) using the receiver's public key (PuK), which is accessible by anyone. The result of this process is cipher text (C):

$$C = E_{Puk}(M)$$

On the other side, the receiver decrypts (D) the cipher text (C) using their private key (PrK) to recover the plain message (M):

$$M = D_{Prk}(C)$$

### 53.6. Structure

Figure 105 shows the class diagram for the ASYMMETRIC ENCRYPTION pattern. A Principal may be a user or an organization that is responsible for sending or receiving messages. The principal may have the roles of Sender or Receiver. A Sender may send a Message and/or an EncryptedMessage to a Receiver with which it shares a secret key.

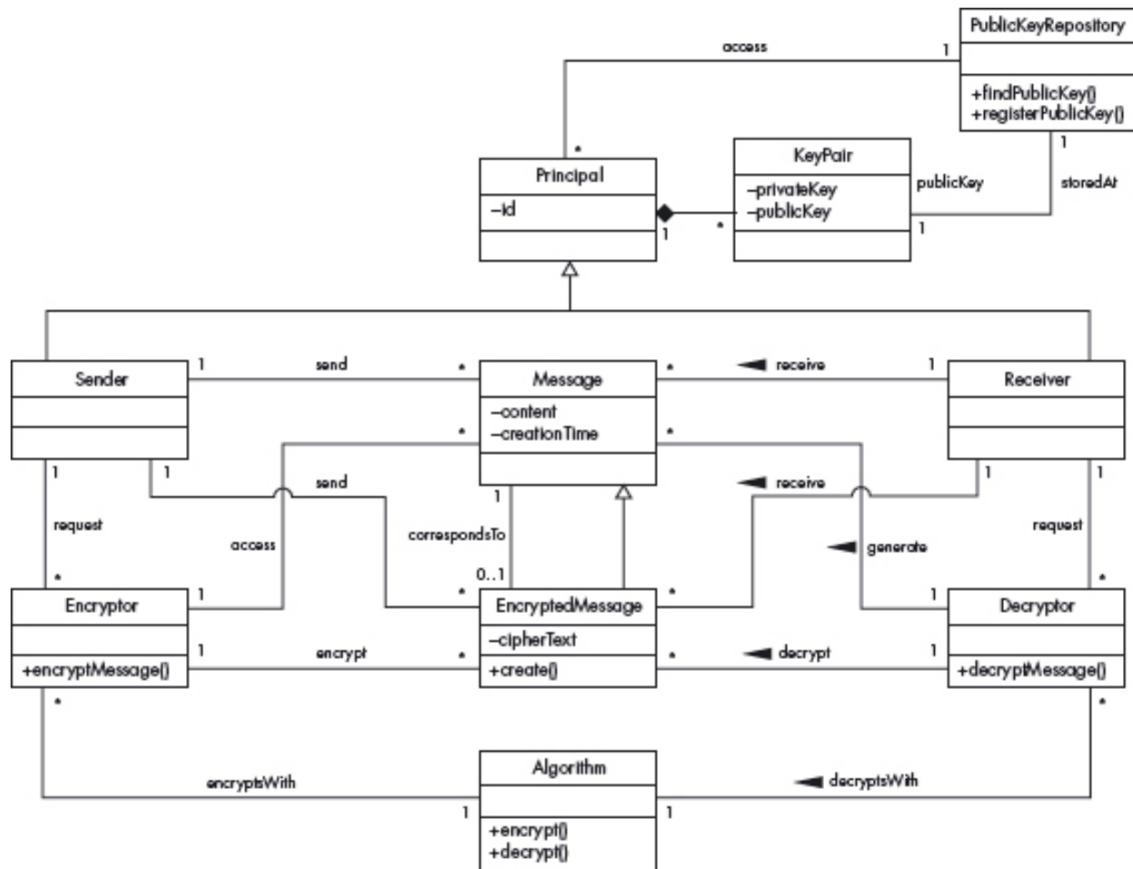


Figure 105: Class diagram for the ASYMMETRIC ENCRYPTION pattern

A Principal has one or more KeyPairs that are composed of a private key, kept secret by its owner, and a public key, which is publicly published. PublicKeyRepository is a repository that contains a list of public keys where users can register and/or access public keys. These two keys are mathematically related, so while one encrypts, the other decrypts. However, it is not feasible to deduce a private key from its corresponding public key.

The Encryptor creates the EncryptedMessage that contain the cipher text using the public key of the Receiver provided by the sender, while the Decryptor deciphers the encrypted data into its original form using its private key. Both the Encryptor and Decryptor use the same Algorithm to encipher and decipher a message.

### 53.7. Dynamics

We describe the dynamic aspects of the ASYMMETRIC ENCRYPTION pattern using sequence diagrams for the following use cases: ‘Encrypt a message’ and ‘Decrypt a message’.

Use Case:	Encrypt a Message – Figure 106
Summary	A Sender wants to encrypt a message.
Actors	A Sender.
Precondition	The Sender has access to the Receiver’s public key. Both Sender and Receiver have access to a repository of algorithms. The message has already been created by the Sender.
Description	<ol style="list-style-type: none"> <li>1. A Sender sends the message, the Receiver’s public key, and the algorithm identifier to the Encryptor.</li> <li>2. The Encryptor ciphers the message using the algorithm specified by the Sender.</li> <li>3. The Encryptor creates the EncryptedMessage that includes the cipher text.</li> </ol>
Postcondition	The message has been encrypted and sent to the Sender.

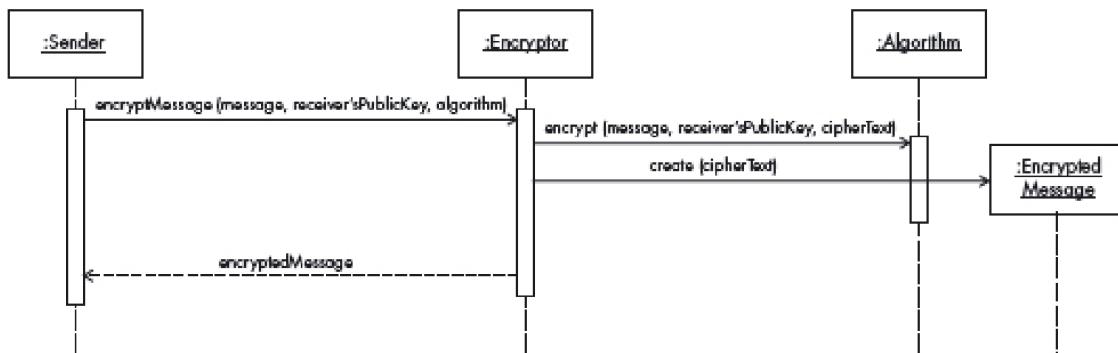


Figure 106: Sequence diagram for the use case ‘Encrypt a message’

Use Case:	Decrypt an Encrypted Message – Figure 107
Summary	A Receiver wants to decrypt an encrypted message from a Sender.
Actors	A Receiver.
Precondition	Both the Sender and Receiver have access to a repository of algorithms.
Description	<ol style="list-style-type: none"> <li>1. A Receiver sends the encrypted message and their private key to the Decryptor.</li> <li>2. The Decryptor deciphers the encrypted message using the Receiver’s public key.</li> <li>3. The Decryptor creates the Message that contains the plain text obtained from the previous step.</li> <li>4. The Decryptor sends the plain text Message to the receiver.</li> </ol> <ul style="list-style-type: none"> <li>• If the key used in step 2 is not mathematically related to the key used for encryption, the decryption process fails.</li> </ul>
Alternate Flow	
Postcondition	The encrypted message has been deciphered and delivered to the Receiver.

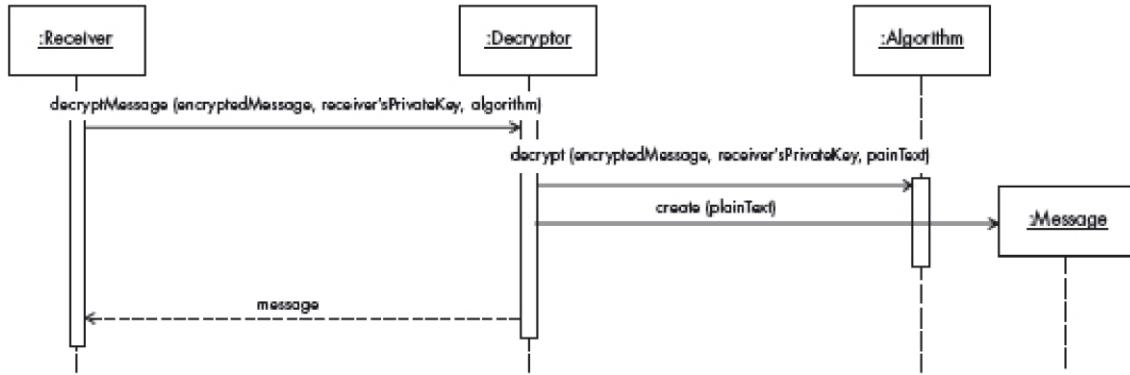


Figure 107: Sequence diagram for the use case 'Decrypt an encrypted message'

### 53.8. Implementation

- Use the Strategy pattern [1] to select different encryption algorithms.
- The designer should choose well-known algorithms such as RSA [2].
- Encryption can be implemented in different applications, such as in e-mail communication, distribution of documents over the Internet, or web services. In these applications we are able to encrypt an entire document. However, in web services we can encrypt parts of a message.
- Both the sender and the receiver have to previously agree what cryptographic algorithms they support.
- A good key pair generator is very important. It should generate key pairs for which the private key cannot be deduced from the public key.

### 53.9. Example Resolved

Alice now can look up Bob's public key and encrypt the message using this key. Since Bob keeps his private key secret, he is the only one who can decrypt Alice's message. Eve cannot understand the encrypted data since Eve does not have access to Bob's private key.

### 53.10. Consequences

The asymmetric encryption pattern offers the following benefits:

- Asymmetric encryption does not require a secret key to be shared among all the participants. Anyone can look up the public key in the repository and send messages to the owner of the public key.
- Only recipients that possess the corresponding private key can make the encrypted message readable again.
- The strength of a cryptosystem is based on the secrecy of a long key [Sta06]. The cryptographic algorithms are known to the public, so the private key should be kept protected from unauthorized users.
- It is possible to select from several encryption algorithms the one suitable for the application's needs.
- Encryption algorithms that take an acceptable time to encrypt messages exist.

The pattern also has the following potential liabilities:

- Cryptography operations are computationally intensive and may affect the performance of the application. Asymmetric encryption is slower than symmetric encryption. It is best to use a combination of both algorithms: asymmetric encryption for key distribution, and symmetric encryption for message exchange.
- Encryption does not provide data integrity. The encrypted data can be modified by an attacker: other means, such as hashing, are needed to verify that a message has not been changed.
- Encryption does not prevent a replay attack, because an encrypted message can be captured and resent without being decrypted. It is recommended to use another security mechanism, such as timestamps or Nonces, to prevent this attack.
- This pattern assumes that a public key belongs to the person who they claim to be. How can we know that this person is not impersonating another? To confirm that someone is who they say they are, we can use certificates issued by a certification authority (CA). If the CA is not trustworthy, we may lose security.

### **53.11. Known Uses**

ASYMMETRIC ENCRYPTION has been widely used in different products.

- GNuPG [3] is free software that secures data from eavesdroppers.
- Java Cryptographic Extension [4] supports a variety of algorithms, including asymmetric encryption.
- The .NET framework [5] provides several classes to perform asymmetric encryption and decryption.
- XML Encryption [6] is one of the foundation web services security standards that defines the structure and process of encryption for XML messages. This standard supports both types of encryption: symmetric and asymmetric encryption.
- Pretty Good Privacy (PGP) uses asymmetric encryption and decryption as one of its process to secure e-mail communication [7].

### **53.12. See Also**

- The Secure Channels pattern supports the encryption/decryption of data. This pattern describes encryption in more general terms: it does not distinguish between asymmetric and symmetric encryption.
- The Strategy pattern [1] describes how to separate the implementation of related algorithms from the selection of one of them. This pattern can be used to select an encryption algorithm dynamically.
- Predicate-based encryption is a family of public key encryption schemes; patterns for them are described in [8].

### **53.13. References**

[1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

- [2] Rivest, R. L., Shamir, A., & Adleman, L. M. (2019). A method for obtaining digital signatures and public key cryptosystems. In *Secure communications and asymmetric cryptosystems* (pp. 217-239). Routledge.
- [3] GnuPG. (n.d.). The GNU Privacy Guard. from <http://www.gnupg.org/>
- [4] Sun Microsystems Inc. (n.d.). Java Cryptography Extension (JCE). From <http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html>
- [5] Microsoft Corporation. (2007, November). .NET Framework Class Library. from <http://msdn.microsoft.com/enus/library/ms229335.aspx>
- [6] W3C. (2002, December 10). XML Encryption Syntax and Processing. From <http://www.w3.org/TR/xmlenc-core/>
- [7] Wikipedia contributors. (n.d.). Pretty Good Privacy. Wikipedia. from [https://en.wikipedia.org/wiki/Pretty\\_Good\\_Privacy](https://en.wikipedia.org/wiki/Pretty_Good_Privacy)
- [8] de Muijnck-Hughes, J., & Duncan, I. (2012, June). Thinking towards a pattern language for predicate based encryption crypto-systems. In *2012 IEEE Sixth International Conference on Software Security and Reliability Companion* (pp. 27-32). IEEE.

### 53.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **54. Authenticator**

## **54.1. Intent**

When a subject identifies itself to the system, the authenticator pattern allows verification that the subject intending to access the system is who or what it claims to be.

## **54.2. Example**

The computer system at Melmac State University has legitimate users who use it to host their files. However, there is no way to ensure that a user who is logged in is a legitimate user. When Melmac was a small school and everybody knew everybody, this was acceptable and convenient. Now the school is bigger and there are many students, faculty members and employees who use the system. Some incidents have occurred in which users impersonated others and gained illegal access to their files.

## **54.3. Context**

Computer systems that contain resources that may be valuable because they include information about business plans, user medical records and so on. We only want subjects that have some reason to gain access to our system to be able to do so.

## **54.4. Problem**

How can we prevent imposters from accessing our system? A malicious attacker could try to impersonate a legitimate user to gain access to their resources. This could be particularly serious if the user impersonated has a high level of privilege. How do we verify that a user intending to access the system is legitimate?

The solution to this problem must resolve the following forces:

- Flexibility. A variety of users require access to the system and a variety of system units exist with differently sensitive assets. We need to be able to handle all this variety appropriately, or we risk security exposures.
- Dependability. We need to authenticate users in a reliable and secure way. This means using a robust protocol and a way to protect the results of authentication. Otherwise, users may skip authentication, or illegally modify its results, exposing the system to security violations.
- Cost. There are trade-offs between security and cost: more secure systems are usually more expensive.
- Performance. If authentication needs to be performed frequently, performance may become an issue.
- Frequency. We should not make subjects authenticate frequently.

## 54.5. Solution

Use a single point of access to receive the interactions of a subject with the system and apply a protocol to verify the identity of the subject. The protocol used may be simple or complex, depending on the needs of the application.

## 54.6. Structure

Figure 108 shows the class diagram for this pattern. A Subject requests access to the system. The Authenticator receives this request and applies a protocol using some AuthenticationInformation. If the authentication is successful, the Authenticator creates a ProofOfIdentity, which is assigned to the subject to indicate that they are legitimate.

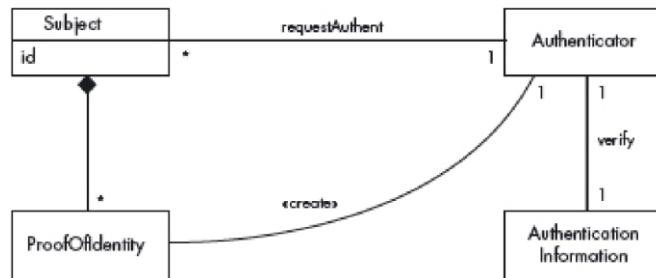


Figure 108: Class diagram for the AUTHENTICATOR pattern

## 54.7. Dynamics

Figure 109 shows the dynamics of the authentication process. A subject User requests access to the system through the Authenticator. The Authenticator applies some authentication protocol, for example by verifying some information presented by the subject, and as a result a ProofOfIdentity is created and assigned to the subject.

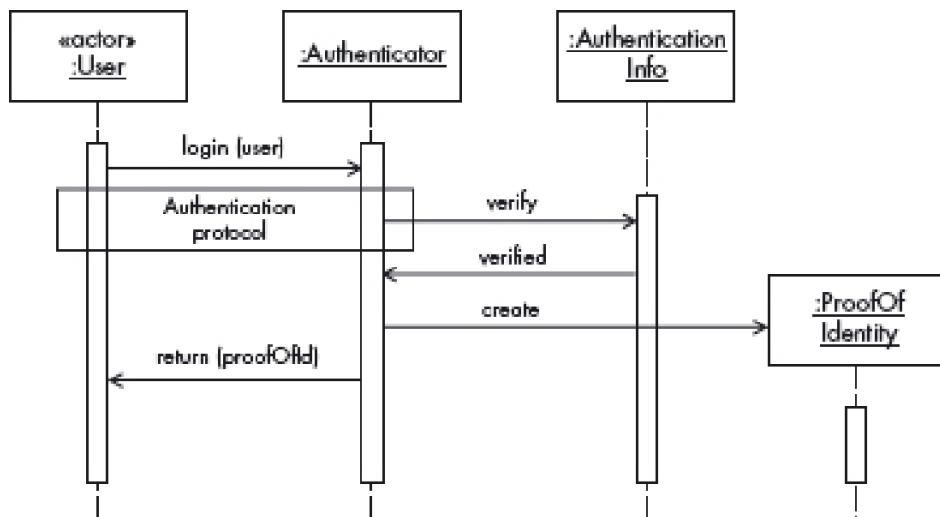


Figure 109: Sequence diagram for the use case 'Authenticate subject'

## **54.8. Implementation**

In centralized systems, the operating system controls the creation of a session in response to the request by a subject, typically a user. The authenticated user (represented by processes running on their behalf) is then allowed to access resources according to their rights.

Some database systems have their own authentication systems, even when running on top of an operating system. There are many ways to implement authentication protocols, described in specific authentication patterns, for example in REMOTE AUTHENTICATOR/AUTHORIZER. Sensitive resource access, for example data in financial systems, requires more elaborate process authentication than systems that handle lower-value assets. Distributed systems may implement this function in specialized servers.

## **54.9. Example Resolved**

Melmac State University implemented an authentication system and now only legitimate users can access their system. Illegal access to files has stopped.

## **54.10. Consequences**

The AUTHENTICATOR pattern offers the following benefits:

- Flexibility. Depending on the protocol and the authentication information used, we can handle any types of users and we can authenticate them in diverse ways.
- Dependability. Since the authentication information is separated, we can store it in a protected area, to which all subjects may have at most read-only access.
- Cost. We can use a variety of algorithms and protocols of different strength for authentication. The selection depends on the security and cost trade-offs. Three varieties include something the user knows (passwords), something the user has (ID cards), or something the user is (biometrics).
- Authentication can be performed in centralized or distributed environments.
- Performance. We can produce a proof of identity to be used in lieu of further authentication. This improves performance.
- Frequency. Using a token means that we do not need to authenticate users.

The pattern also has the following potential liabilities:

- The authentication process takes some time: the overhead depends on the protocol used.
- The general complexity and cost of the system increase with the level of security.
- If the protocol is complex, users waste time and get annoyed.

## **54.11. Variants**

Single Sign On. Single Sign On (SSO) is a process whereby a subject verifies their identity, and the results of this verification can be used across several domains and for a given amount of time [1]. The result of the authentication is an authentication token, used to qualify all future accesses by the user

## **54.12. Known Uses**

- Commercial operating systems use some form of authentication, typically passwords, to authenticate their users.
- RADIUS (Remote Authentication Dial-In User Service) provides a centralized authentication service for network and distributed systems [2] [3] – see REMOTE AUTHENTICATOR/AUTHORIZER pattern.
- The SSL authentication protocol uses a PKI arrangement for authentication.
- SAML, a web services standard for security, defines one of its main uses as a way to implement authentication in web services.
- E-commerce sites such as eBay and Amazon.

## **54.13. See Also**

- The Distributed Filtering and Access Control framework includes authentication [4].
- REMOTE AUTHENTICATOR/AUTHORIZER provides facilities for authentication and authorization when accessing shared resources in a loosely coupled distributed system.
- CREDENTIAL pattern provides a secure means of recording authentication and authorization information for use in distributed systems.
- Single Access Point pattern.
- Check Point pattern.

## **54.14. References**

- [1] Dalton, C., & King, C. (2001). *Security Architecture: Design, Deployment and Operations*. osborne-McGraw-Hill.
- [2] Garfinkel, S., & Spafford, G. (2002). *Web security, privacy & commerce*. " O'Reilly Media, Inc.".
- [3] Hassell, J. (2002). *RADIUS: securing public access to private resources*. " O'Reilly Media, Inc.".
- [4] Hays, V., Loutrel, M., & Fernandez, E. B. (2000). The object filter and access control framework. In *Procs. Pattern Languages of Programs (PLoP2000) Conference*.

## **54.15. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 55. Behavior-Based IDS

## 55.1. Intent

The BEHAVIOR-BASED IDS pattern describes how to check every request for access against patterns of network traffic in order to detect possible deviations from normal behavior (anomaly) that may indicate an attack and trigger appropriate responses.

## 55.2. Example

A company uses a public network for its applications. The network is exposed to security threats, especially a variety of unknown attacks. Their business could be in jeopardy if their customers realize that their system is not secure enough.

## 55.3. Context

Any network application where the temporal behavior of network traffic is repetitive and predictable.

## 55.4. Problem

Whenever data is accessed from the Internet or other external networks, there is always a possibility that this access can be harmful to the network. We need to detect possible attacks while they are occurring.

The solution to this problem must resolve the following forces:

- New attacks. In today's world networks are constantly bombarded with new attacks that do not have a specific attack signature. We need to detect these kinds of attacks.
- Real-time. We need to detect attacks in real time while they are happening, and not after the attack has happened and it is too late to recover from it.
- Increased vulnerability. Some networks, such as mobile networks, are more vulnerable to unknown attacks because of their mobile nature.

## 55.5. Solution

Observe the traffic over a network and try to find deviations from normal or expected behavior. Any deviation from normal behavior is treated as a sign of intrusion.

## 55.6. Structure

Figure 110 shows the class diagram for this pattern. A Client requests some service from the system. The IDS intercepts this request and sends it to an EventProcessor. The EventProcessor processes the event data as301 needed by the AttackDetector, which involves establishing profiles of normal behavior that can be compared with the current behavior in BehaviorProfileInformation and passes the processed data to the AttackDetector. When an attack is detected, a Response is created.

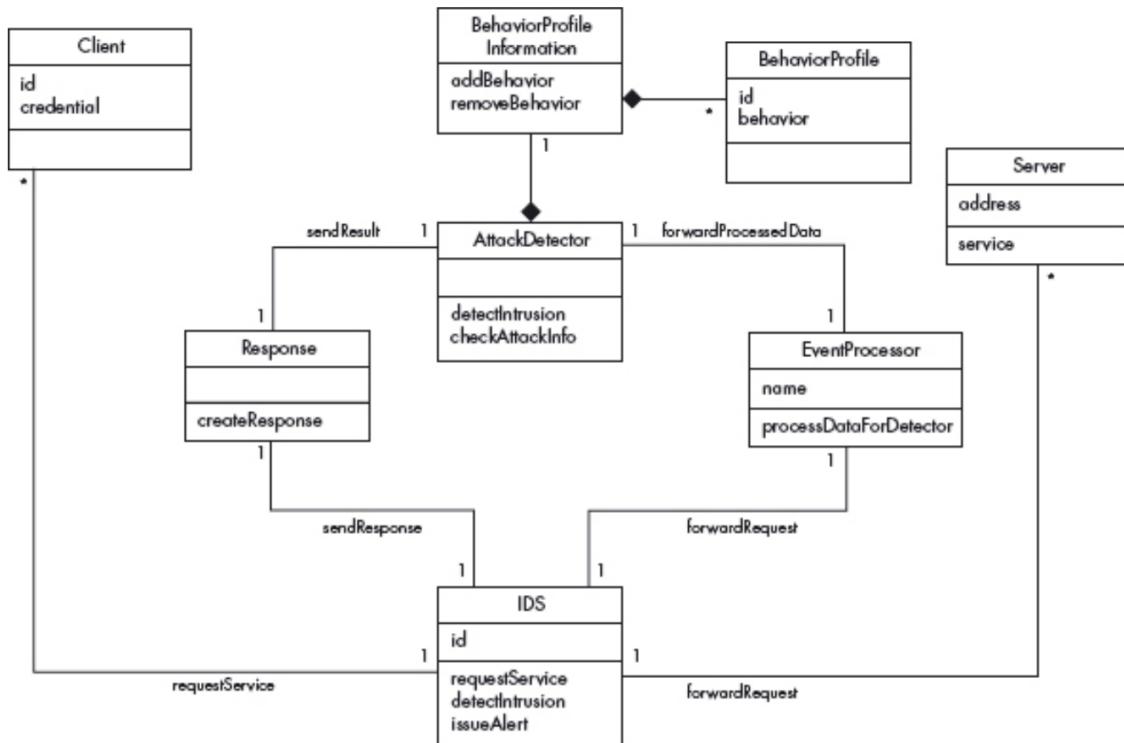


Figure 110: Class diagram for the BEHAVIOR-BASED IDS pattern

## 55.7. Dynamics

We present the dynamic aspects of the BEHAVIOR-BASED IDS pattern using sequence diagrams for the following use case.

Use Case:	Detect an Intrusion – Figure 111
Summary	The Client requests a service from the Host. The behavior-based IDS intercepts the message and determines whether the behavior of the request matches a normal behavior profile. If it does not, an attack is suspected, and a response is raised.
Actors	Client, Server.
Precondition	A set of normal behavior profiles is available.
Description	<ol style="list-style-type: none"> <li>Client makes a service request to the Host.</li> <li>The IDS send the request event to an EventProcessor.</li> <li>The EventProcessor processes the event as required by the AttackDetector and passes the processed event data to the AttackDetector.</li> <li>The AttackDetector tries to detect whether this request is an attack or not by comparing the behavior profile of the request with the available behavior profiles in the BehaviorProfileInformation.</li> <li>If a match is detected, a Response is created. <ul style="list-style-type: none"> <li>The BehaviorProfileInformation may not be able to detect an attack (a false negative).</li> <li>The BehaviorProfileInformation can match and may indicate an attack when no attack is present (a false positive).</li> </ul> </li> </ol>
Alternate Flows	
Postcondition	If an attack is detected while it is happening, suitable preventive measures can be adopted.

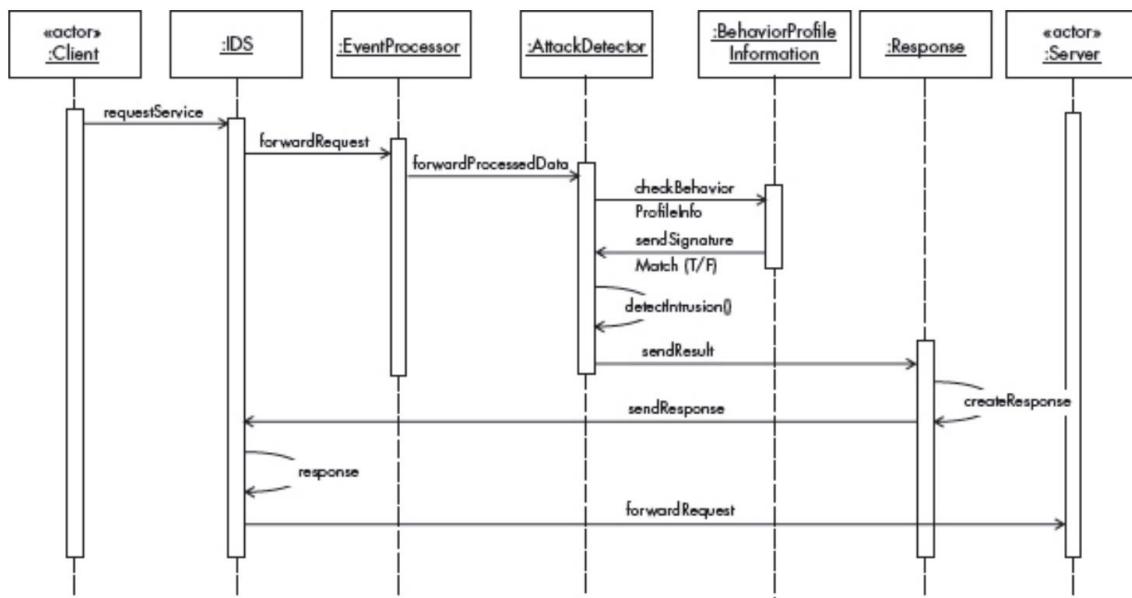


Figure 111: Sequence diagram for the use case 'Detect an intrusion'

## 55.8. Implementation

Examples of techniques used for anomaly detection in practice are:

- Genetic algorithms. In this approach, applications are modeled in terms of system calls for different conditions, such as normal behavior, error conditions and attack conditions. A typical genetic algorithm involves two steps. The first step involves coding the input population of the algorithm. The second step involves finding a fitness function to test each individual of the population against some evaluation criteria. In the learning process each event sequence of node behavior forms a gene. Fitness is calculated for a collection of genes. If genes with required fitness cannot be found in the current generation, new sets of genes are evolved through crossover and mutation. The process of evolution continues until genes with the required fitness are found. The detection process involves defining vectors for event data and methods of testing whether the vector indicates an intrusion or not [1].
- Protocol verification. The basis for this approach is the fact that most intruders use irregular or unusual protocol fields, which are not handled properly by application systems [2].
- Statistical models. These can be either multivariate models or models based on available statistics such as threshold measures or mean and standard deviations of the profile. Clustering analysis where clusters represent similar activities or user patterns is also sometimes used [2].

## 55.9. Example Resolved

We added an intrusion detection system to our network. Now all traffic is checked against a normal behavior profile to see whether the access request is an anomaly and hence a possible attack. We are now able to detect many new attacks that do not have a known signature and prevent them.

## 55.10. Consequences

The BEHAVIOR-BASED IDS pattern offers the following benefits:

- New attacks. Detection can be effective against new attacks that could cause abnormal behavior in the network traffic. For example, we can identify an attack with a specific behavior, such as when a usually passive web server tries to connect to a large number of addresses, it could be the result of a worm attack.
- Real time. This kind of IDS works well with network traffic that exhibits a normal behavior and where it will be easier to detect an abnormal behavior pattern for the network.
- Increased vulnerability. This kind of IDS is usually good in wireless networks, which are more vulnerable due to their mobile nature.

The pattern also has the following potential liabilities:

- It generates a lot of false positives. Many anomalies detected are not attacks but could be just unusual behaviors of users.
- It cannot be implemented in networks that do not have a predictable traffic pattern.
- The technology adopted for one network is not easily portable to another system and can be different from system to system in a network, as normal behavior for one system is usually not the normal behavior for another system.
- If the attacker carries out an attack by mimicking regular traffic or normal behavior, the attack may go undetected.

## 55.11. Known Uses

- Cisco IPS 4200 Series utilizes detection techniques including stateful pattern recognition, protocol parsing, heuristic detection, and anomaly detection [3].
- AirTight's wireless IPS automatically detects, classifies, blocks, and locates wireless threats using behavior analysis. They use a genetic algorithm to establish normal behaviors [4].

Some other uses of anomaly based IDSs are given in Table 3 [5].

Name	Manufacturer	Hybrid	Response	Anomaly-Related Techniques
AirDefense Guard	AirDefense, Inc.	Y	Y	Detection, correlation, and multi-dimensional detection
Barbedwire's IDS Softblade	Barbedwire Technologies	Y	Y	Protocol analysis, pattern matching
BreachGate WebDefend	Breach Security	Y		Behavior-based analysis, statistical analysis, Using correlation functions
Bro	Lawrence Berkeley National Laboratory	Y	Y	Application-level semantics, event analysis, pattern matching, protocol analysis

Table 3: Network-based IDS platforms with anomaly detection functionalities, according to the manufacturer's information [Gar09]

The Hybrid column indicates hybrid detection, and the Response column indicates that some kind of response mechanism is also available.

## 55.12. See Also

- This pattern is used in conjunction with the SIGNATURE-BASED IDS pattern.
- Firewall patterns are usually used together with the IDS in a network.
- The response class could be implemented as a Strategy pattern [6].

## 55.13. References

- [1] Raja, P. K. (2006). Wireless node behavior based intrusion detection using genetic algorithm.
- [2] Verwoerd, T., & Hunt, R. (2002). Intrusion detection techniques and approaches. *Computer communications*, 25(15), 1356-1365.
- [3] Cisco. (n.d.). Cisco Systems: Products and Technologies > Cisco Intrusion Detection. from <http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/>
- [4] Airlight. (n.d.). Airtight Networks: WLAN Intrusion Prevention. from <http://www.airtightnetworks.com/home/solutions/wireless-intrusion-prevention.html>
- [5] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., & Vázquez, E. (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, 28(1-2), 18-28.
- [6] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

## 55.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **56. Circle of Trust**

## **56.1. Intent**

The CIRCLE OF TRUST pattern allows the formation of trust relationships among service providers to allow their subjects to access an integrated and more secure environment.

## **56.2. Example**

Our university had different ways to access different services. For each service, one needed a different protocol and to remember a different password. This was cumbersome and prone to error; it was also insecure because people would write their multiple passwords on their office bulletin boards.

## **56.3. Context**

Service providers that provide services to consumers (subjects) over large systems such as the Internet.

## **56.4. Problem**

In such large open environments, subjects are typically registered (have accounts) with unrelated services. Subjects may have no relationships with many other services available in the open environment. It may be cumbersome for the subject to deal with multiple accounts, and it may not be secure to build new relationships with other services since identity theft or violation of privacy can be performed by rogue services. How can we take advantage of relationships between service providers to avoid the inconvenience of multiple logins and to select trusted services?

The solution to this problem must resolve the following forces:

- Service providers are numerous on public networks. It can be cumbersome, indeed impossible, for each service provider to define relations with every other provider.
- The service providers' infrastructures for their subjects' login may be implemented using different technologies.

## **56.5. Solution**

Each service provider establishes business relationships with a set of other service providers. These relationships are materialized by the existence of operational and possibly business agreements between services. Such relationships are trust relationships since each service provider expects the other to behave according to the operating agreements. Therefore, a circle of trust is a set of service providers that have business and operational relationships with each other – that is, that trust each other. Operational agreements could include information about whether they can exchange information about their subjects, for example, and how and what type of information can be exchanged. Business agreements could describe how to offer

cross-promotions to their subjects. Service providers need to agree on operational processes before their users can benefit from the seamless environment.

## 56.6. Structure

Figure 112 shows a UML class diagram with an OCL constraint describing the structure of the solution. A CircleOfTrust includes several ServiceProviders, who trust each other, as described by the OCL expression.

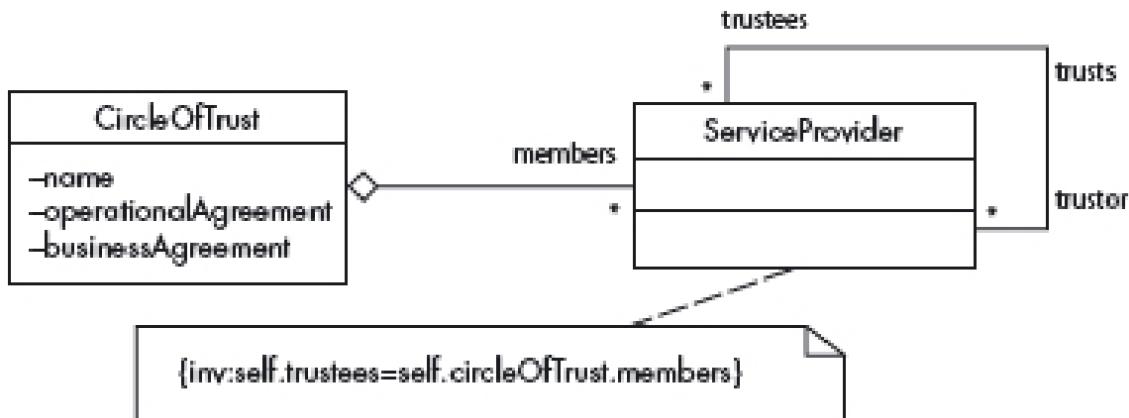


Figure 112: Class diagram for the CIRCLE OF TRUST pattern

## 56.7. Dynamics

## 56.8. Implementation

Operational agreements should include a means to concretely enable trust, such as the sharing of a secret key or the secure distribution of certificates. The providers have to exchange credentials through some kind of external channel to trust and recognize each other.

## 56.9. Example Resolved

Our university can now trust the identity providers who are members of our federation, so we do not have to worry that they will misuse our information.

## 56.10. Consequences

The CIRCLE OF TRUST pattern offers the following benefits:

- The subjects can interact more securely with (potentially unknown) trusted services from the circle of trust.
- The services can provide a seamless environment to the subjects and can exchange information about their users by using operating agreements that describe what common technologies to use.

Liabilities include keeping the participants synchronized in the presence of changes.

## **56.11. Known Uses**

- Identity federation systems and models such as the LIBERTY ALLIANCE IDENTITY FEDERATION standard.
- PayPal has a variety of partners that trust their payment system, including Verifone and Equinox [1].

## **56.12. See Also**

- IDENTITY PROVIDER pattern and IDENTITY FEDERATION pattern.
- LIBERTY ALLIANCE IDENTITY FEDERATION pattern [2].
- [3] formally discusses the CIRCLE OF TRUST.
- CREDENTIAL pattern.

## **56.13. References**

[1] Paypal partners with Verifone and Equinox to accept mobile payments in store. (n.d.).

VentureBeat. from <http://venturebeat.com/2012/05/24/paypal-partners-with-verifoneequinox-to-accept-mobilepayments-in-store/>

[2] Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

[3] Nizamani, H. A., & Tuosto, E. (2011). Patterns of Federated Identity Management Systems as Architectural Reconfigurations. *Electronic Communications of the EASST*, 31.

## **56.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 57. Controlled Access Session

## 57.1. Intent

The CONTROLLED ACCESS SESSION pattern describes how to provide a context in which a subject (user, system) can access resources with different rights without need to reauthenticate every time they access a new resource.

## 57.2. Example

Lisa is a secretary in a medical organization who sometimes helps with patient tests in the laboratory. As a secretary she has access to patients' information such as name, address, social security number and so on. This is necessary so that she can bill them and their insurance companies. In the lab she has access to anonymized patient test results. Combining the accesses provided by her two jobs in one window allows her to associate test results and patients' names, which violates patient privacy.

## 57.3. Context

Any environment in which we need to control access to computing resources and in which users can be classified according to their jobs, groups, departments, assignments, or tasks.

## 57.4. Problem

A given user may be authorized to access a system because they need to perform several functional activities. However, for a particular access, only those privileges should be active that are necessary to perform the intended task. This is an application of the principle of least privilege and is necessary to prevent the user from misusing the system, either intentionally, accidentally by performing an error, or without knowledge and tricked to do so, for example through a Trojan Horse attack. Additionally, it potentially restricts damage in the case of session hijacking: a successful attack process would not have all the privileges of a user available, only the active subset.

The solution to this problem must resolve the following forces:

- Subjects may have many rights directly or indirectly through the execution contexts they need for their tasks. Using all of them at one time may result in conflicts of interest and security violations. We need to restrict the use of those rights depending on the application or task the subject is performing.
- In the context of an interaction, we can make access to some functions implicit, thus facilitating the use of the system and preventing errors that may result in vulnerabilities. For example, some editors or other tools could be implicitly available in some sessions.
- It is not convenient to make subjects reauthenticate every time they request a new resource. Once the subject is authenticated, this condition should remain valid during the whole session.

## 57.5. Solution

Define a unit of interaction, a session, which has a limited lifetime, for example between login and logoff of a user, or between the beginning and the end of a transaction. When a user logs on and after authentication, the session activates some execution contexts with only a subset of the authorizations they possess. This should be the minimum subset that is needed for the user or transaction to perform the intended task. Only those rights are available within the session. A subject can be in several sessions at the same time; however, in every session only the necessary rights are active.

## 57.6. Structure

Figure 113 shows the class model of the CONTROLLED ACCESS SESSION pattern. The classes Subject and Session have obvious meanings. The class ExecutionContext contains the set of active rights that the subject may use within the session.

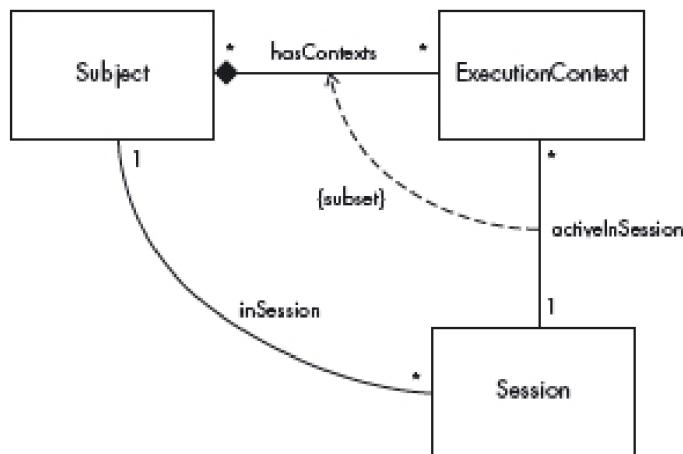
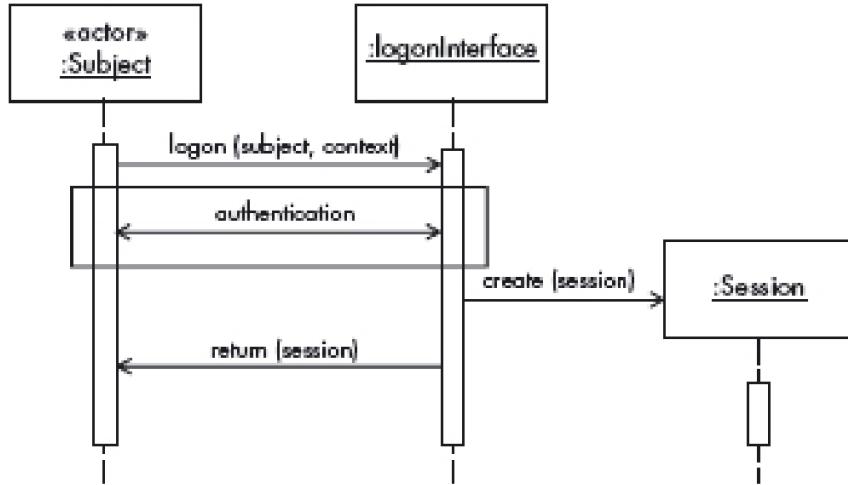


Figure 113: Class model for the CONTROLLED ACCESS SESSION pattern

## 57.7. Dynamics

Figure 114 shows the use case ‘Open (activate) a session’. A subject logs on and the logon interface authenticates it. The box with a double arrow indicates some authentication dialog or protocol. After the subject is authenticated, the interface creates a session object and returns a handle to the subject.



*Figure 114: Sequence diagram for the use case ‘Open a session’*

## 57.8. Implementation

Based on institution and application policies, define which contexts (implying specific rights) should be used in each task and grant them to the corresponding subject. The rights should be selected using the least privilege principle, and there should be no contexts with excessive rights; for example, the administrator rights should be divided into smaller sets.

## 57.9. Example Resolved

Lisa can log on a secretary or as a lab assistant, but she cannot combine these activities in one session. Now she cannot relate test results to patients’ names.

## 57.10. Consequences

The CONTROLLED ACCESS SESSION pattern offers the following benefits:

- We can give only the necessary rights to each execution context, according to its function, and we can invoke in a session only those contexts that are needed for a given task.
- We can exclude combinations of contexts that might result in possible access violations or conflicts of interest.
- Any functions can be made implicit in a session.
- Once a subject starts a session it doesn’t have to be reauthenticated: its status is kept by the session.

The pattern also has the following potential liabilities:

- If we need to apply fine-grained access, it might be inefficient to include many contexts in order to perform complex activities.
- Using sessions may be confusing to the users.

## 57.11. Known Uses

- Session Access is part of the RBAC standard proposal by NIST, which has been adopted by the American National Standards Institute, International Committee for Information Technology Standards (ANSI/INCITS) as ANSI INCITS 359-2004 [1].
- Multics [2] used execution contexts (based on projects) to limit access rights.
- Session Access is implemented in the security module CSAP [3] of the Webocrat system in conjunction with an RBAC policy.
- Views in relational databases can be used to define sets of rights. Controlling the use of views by users can control their use of rights in sessions. This is done for example in Oracle and DB2, where SQL can be used to define restricted views [4].

## 57.12. See Also

- The Access Session pattern is used in the SESSION-BASED ROLEBASED ACCESS CONTROL pattern and Attribute-Based Access Control patterns.
- The Security Session pattern intents to prevent a user from having to be reauthenticated every time they access a new object.
- Abstract Session [5] is a pattern with a similar objective to Security Session: when an object's services are invoked by clients, the server object may have to maintain state for each client. The server creates a session object that encapsulates state information for the client and returns a pointer to the session object.

However, none of these patterns considers limitation of rights. Our pattern is an extension of these patterns, concentrating all its security functions and emphasizing the function of a session as a limiter of rights.

## 57.13. References

- [1] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3), 224-274.
- [2] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc..
- [3] Dridi, F., Fischer, M., & Pernul, G. (2003, May). CSAP—An Adaptable Security Module for the E-Government System Webocrat. In *IFIP International Information Security Conference* (pp. 301-312). Springer, Boston, MA.
- [4] Elmasri, R., & Navathe, S. (2003). *Fundamentals of Database Systems*. 4th edition. Addison-Wesley.
- [5] Pryce, N. (2001). Abstract session: An object structural pattern. In *Design patterns in communications software* (pp. 191-208).

## 57.14. Source

- Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **58. Credential**

## **58.1. Intent**

The CREDENTIAL pattern provides a secure means of recording authentication and authorization information for use in distributed systems.

## **58.2. Example**

Suppose we are building an instant messaging service to be used by members of a university community. Students, teachers, and staff of the university may communicate with each other, while outside parties are excluded. Members of the community may use computers on school grounds, or their own systems, so the client software is made available to the community and is installed on the computers of their choice. Any community member may use any computer with the client software installed.

The client software communicates with servers run by the university in order to locate active participants and to exchange messages with them. In this environment, it is important to establish that the user of the client software is a member of the community, so that communications are kept private to the community. Further, when a student graduates, or an employee leaves the university, it must be possible to revoke their communications rights. Each member needs to be uniquely and correctly identified, and a member's identity should not be forgeable.

## **58.3. Context**

Systems in which the users of one system may wish to access the resources of another system, based on a notion of trust shared between the systems.

## **58.4. Problem**

In centralized computer systems, the authentication and authorization of a principal can be handled by that system's operating system, middleware and/or application software; all attributes of the principal's identity and authorization are created by and are available to the system. With distributed systems this is no longer the case. A principal's identity, authentication, and authorization on one system does not carry over to another system. If a principal is to gain appropriate access to another system, some means of conveying this information must be introduced.

More broadly, this is a problem of exchanging data between trust boundaries. Within a given trust boundary, a single authority is in control, and can authenticate and make access decisions on its own. If the system is to accept requests from outside its own authority/trust boundary, the system has no inherent way of validating the identity or authorization of the entity making that request. How then do we allow external users to access some of our resources?

The solution to this problem must resolve the following forces:

- Privacy. The user must provide enough information to grant authorization, without being exposed to intrusive data mining.
- Persistence. The information must be packaged and stored in a way that survives travel between systems, while allowing the data to be kept private.
- Authentication. The data available must be sufficient for identifying the principal to the satisfaction of the accepting system's requirements, while disallowing others from accessing the system.
- Authorization. The data available must be sufficient for determining what actions the presenting principal is permitted to take within the accepting system, while also disallowing actions the principal is not permitted to take.
- Trust. The system accepting the credential must trust the system issuing the credential.
- Generation. There must be entities that produce the credentials such that other domains recognize them.
- Tamper freedom. It should be very difficult to falsify the credential.
- Validity. The credential should have an explicit temporal validity.
- Additional documents. It might be necessary to use the credential together with other documents.
- Revocation. It should be possible to revoke the credential conveniently.

## **58.5. Solution**

Store authentication and authorization data in a data structure external to the systems in which the data is created and used. When presented to a system, the data (credential) can be used to grant access and authorization rights to the requester. For this to be a meaningful security arrangement, there must be an agreement between the systems which create the credential (credential authority) and the systems which allow their use, dictating the terms and limitations of system access.

## **58.6. Structure**

In Figure 115 the principal is an active entity such as a person or a process. The principal possesses a Credential, representing its identity and its authorization rights. A Credential is a composite describing facts about the rights available to the principal. The Attribute may flag whether it is presently enabled, allowing principal control over whether to exercise the right implied by the Credential. An expiration date allows control over the duration of the rights implied by the attribute.

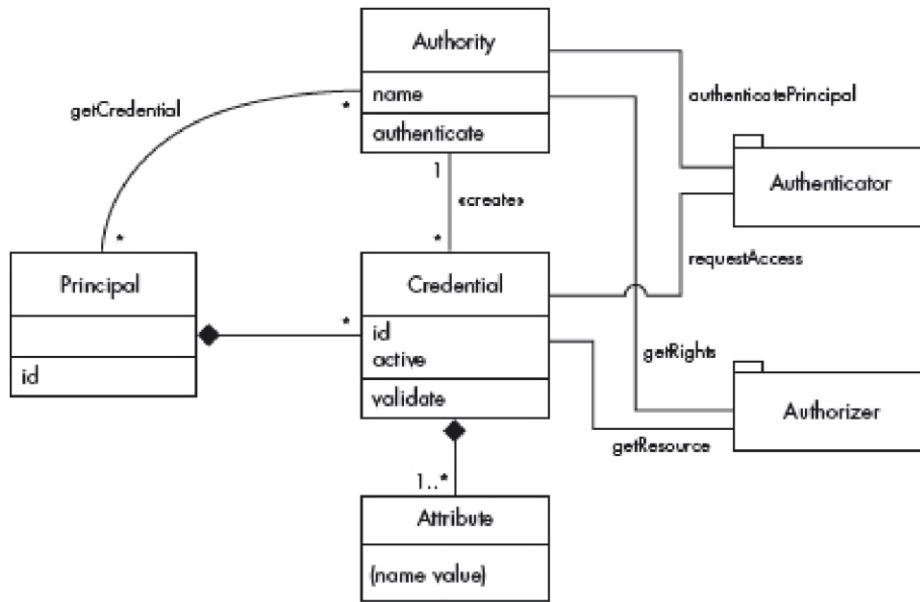


Figure 115: Class diagram for the CREDENTIAL pattern

A Credential is issued by an Authority and is checked by an Authenticator or an Authorizer. Specialization of a Credential is achieved through setting Attribute names and values.

Some specializations of Attributes are worth mentioning. Identity, created by setting an attribute name to, say, ‘username’ and the value to the appropriate username instance, shows that the subject has been authenticated and identified as a user known to the Authenticator. Privilege, named after the intended privilege, implies some specific ability granted to the subject. Group and Role can be indicated in a similar fashion to Identity.

## 58.7. Dynamics

There are four primary use cases:

1. Issue credential, by which a credential is granted to the principal by an authority.
2. Principal authentication, where an authenticator accepts a credential provided to it by a principal and makes an access decision based on the credential.
3. Principal authorization, in which the principal is allowed access to specific items.
4. Revoke credential, in which a principal’s credential are invalidated.

### 58.7.1. Issue Credential – Figure 116

The principal presents itself and any required documentation of its identity to an Authority. Based upon its rules and what it ascertains about the Principal, the Authority creates and returns a Credential. The returned data may include an identity credential, group and role membership credential attribute, and privilege credential attributes. As a special case, the Authority may generate a defined ‘public’ Credential for Principals not previously known to the system. This Credential is made available to Authenticators which reference this Authority.

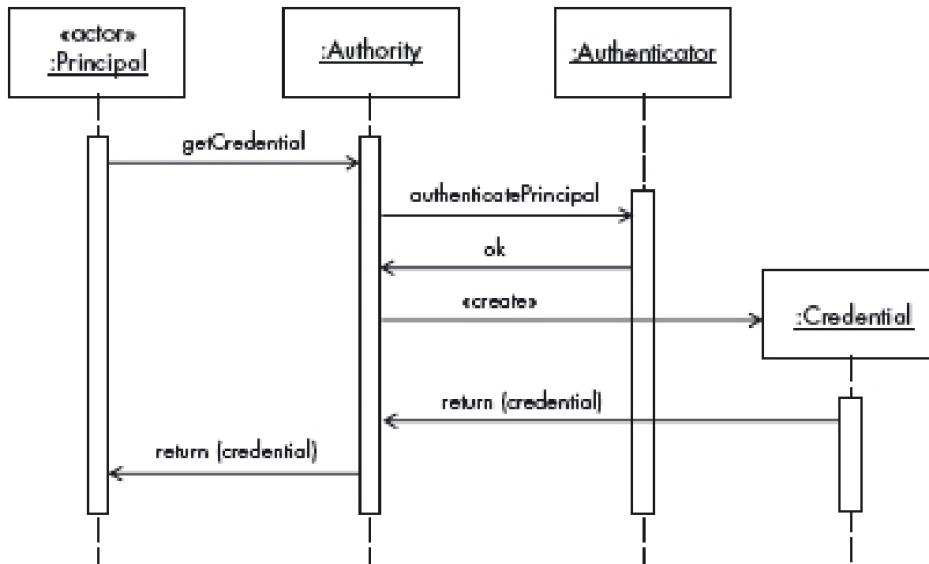


Figure 116: Sequence diagram for the use case 'Issue credential'

### 58.7.2. Principal Authentication – Figure 117

The principal requests authentication at an Authenticator, supplying its name and authentication Credential. The Authenticator checks the Credential and makes an access decision. There are different phases and strengths of check that may be appropriate for this step, discussed in the Implementation section. It is necessary for the Authenticator to be established in conjunction with the original authority. Not shown in the sequence diagram, but it is also optionally possible to forward the authentication request and credentials to the authority for verification.

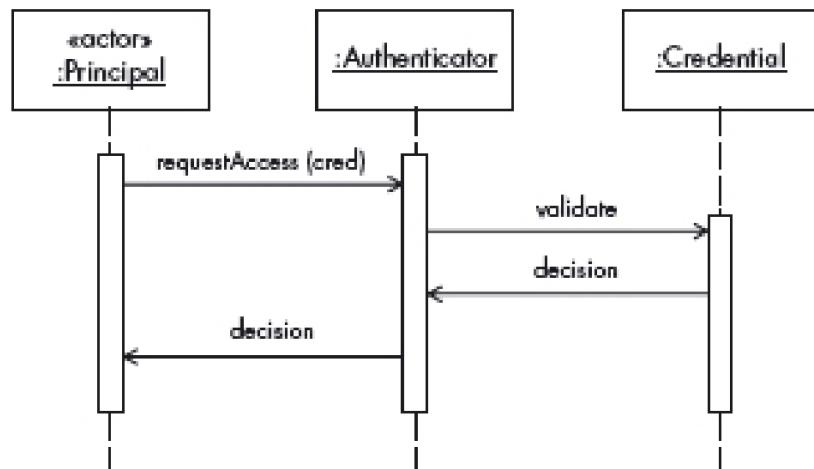


Figure 117: Sequence diagram for the use case 'Principal authentication'

### 58.7.3. Revoke Credential

If it is determined that a given principal should no longer have access to the system, or that a principal's credentials have been stolen or forged, the authority can issue a revocation message to each authenticator and authorizer. Once this message has been received, the authentication and authorization subsystems reject future requests from the affected

credentials. If the principal is still authorized to use the system, new credentials must be issued.

## 58.8. Implementation

The most significant factor in implementing the CREDENTIAL pattern is to determine the nature of the agreement between the participating systems. This begins with consideration of the functions to be provided by the system to which credentials will give access, the potential users of those functions, and the set of rights that are required for each user to fulfill its role. Once these are understood, a clear representation of the subjects, objects and rights can be developed. This representation forms the basis for storing credentials in some persistent medium and sets the terms of authentication and authorization. It also forms the basis for portability, as persisted data may be placed on portable media for transmission to the location(s) of its use.

The problem with a clear representation of security rights is that potential attackers can read them as well as valid participants in the systems in question. In the physical world, anti-forgery devices for credentials use techniques such as embedding the credential data in media that is too expensive to be worth forging for the benefit received: drivers' licenses and other ID cards, passports and currency all are based on the idea that it is too complex and costly for the majority of users to create realistic fakes.

In the digital world, however, copies are cheap. There are two common means of addressing this. One is to require that credentials be established and used within a closed context, and encrypting the communications channels used in that context. The other is to encrypt the credentials when they are issued, and to set up matching decryption on the authenticating system. This further subdivides into 'shared secret' systems, in which the issuing and accepting systems share the cryptographic keys necessary to encrypt and decrypt credentials, and 'public key' systems, in which participating systems can establish means for mutual encryption/decryption without prior sharing. These design choices are part of the terms set by the authority agreement under which the credentials apply. The authenticator must use the same scheme as the authority. Kerberos tokens and X.509 certificates are examples of this that require more specific approaches [1].

As a simple example of 'shared secret' systems, consider a typical online banking authority and authentication setup; at sign-up, the customer verifies their identity to the bank, the authority. As part of the bank's processing, it creates customer data on its website, and allows the customer to create a username and password to gain access to the account. This data is stored on the bank's web server, which serves as the authenticator. The customer later presents their credentials through a browser to the web server, which authenticates under the authority of the bank.

When implementing the 'principal authentication' use case, there are different phases and strengths of check that may be appropriate. For example, when entering my local warehouse club, I need only show a card that looks like a membership card to the authenticator standing at the door. When it comes time to make a purchase, however, the membership card is checked for validity, expiration date and ownership of the person presenting it. In general, the authenticator is responsible for checking the authenticity of the credentials themselves (anti-forgery), whether they belong to their bearer, and whether they constitute valid access to the requested object(s).

## **58.9. Example Resolved**

The university created a credential authority, ‘IM Registration’. It gave it the responsibility of verifying identity and granting a username and password, in the form of an ID card, to university community members when they join the university community. This login embodies the authority of the granting agency and that of the identity of the subject as verified by the agency. The university defined policies such that members were encouraged to keep their login information private.

The client software is coded to implement an authenticator when someone wants to start a session. Access is granted or denied based on the results of the authentication. Checks on the servers ensure that the member’s credential is not expired.

## **58.10. Consequences**

The CREDENTIAL pattern offers the following benefits:

- Fine-grained authentication and authorization information can be recorded in a uniform and persistent way.
- A credential from a trusted authority can be considered proof of identity and of authorization.
- It is possible to protect credentials using encryption or other means.

The pattern has the following potential liabilities:

- It might be difficult to find an authority that can be trusted. This can be resolved with chains (trees) of credentials, by which an authority certifies another authority.
- Making credentials tamper-resistant incurs extra time and complexity.
- Storing credentials outside of the systems that use them leaves system authentication and authorization mechanisms open to offline attack.

## **58.11. Known Uses**

This pattern is a generalization of the concepts embodied in X.509 Certificates, CORBA Security Service’s Credentials [2], Windows security tokens [3], SAML assertions [4] and the Credential Tokenizer pattern [5]. Capabilities, as used in operating systems, are another implementation of the idea.

Passports are a non-technical example of the problem and its solution. Countries must be able to distinguish between their citizens, citizens of nations friendly and unfriendly to them, trading partners, guests, and unwanted people. There may be different rules for how long visitors may stay, and for what they may engage in while they are in the country. Computer systems share some of these traits: they must be able to distinguish between members and non-members of their user community; non-members may be eligible or ineligible to gain system access or participate in transactions.

## **58.12. See Also**

- Metadata-Based Access Control [6] describes a model in which credentials can be used to represent subjects.
- The CREDENTIAL pattern complements Security Session by giving an explicit definition of that pattern's 'session object', as extracted from several existing platforms.
- The Authenticator pattern and the REMOTE AUTHENTICATOR/AUTHORIZER pattern describe types of authenticator.
- An Authorizer is a concrete version of the abstract concept of Reference Monitor.
- Delegation of credentials is discussed in [7].
- [5] describes a Session Object pattern that 'abstracts encapsulation of authentication and authorization credentials that can be passed across boundaries'. They seem to be talking about access rights rather than credentials: credentials abstract authentication and authorization rights.
- The CIRCLE OF TRUST pattern allows the formation of trust relationships among service providers in order for their subjects to access an integrated and more secure environment. CREDENTIALS can be used for identification in a circle of trust.

## **58.13. References**

- [1] Lopez, J., Oppliger, R., & Pernul, G. (2004). Authentication and authorization infrastructures (AAIs): a comparative survey. *Computers & Security*, 23(7), 578-590.
- [2] Anderson, R. (2001). CORBA Security Service Specification. OMG. from <http://www.omg.org/docs/formal/02-03-11.pdf>
- [3] Brown, K. (2005). The .NET Developer's Guide to Windows Security. Addison-Wesley.
- [4] Hughes, J., & Maler, E. (2005). Security assertion markup language (saml) v2. 0 technical overview. *OASIS SSTC Working Draft sstc-saml-tech-overview-2.0-draft-08*, 13.
- [5] Steel, C., & Nagappan, R. (2006). *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Pearson Education India.
- [6] Priebe, T., Fernández, E. B., Mehlau, J. I., & Pernul, G. (2004). A pattern system for access control. In *Research Directions in Data and Applications Security XVIII* (pp. 235-249). Springer, Boston, MA.
- [7] Weiss, M. (2006). Credential delegation: Towards grid security patterns. In *Proceedings of the Nordic Conference on Pattern Languages of Programs* (pp. 65-70).

## **58.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 59. Digital Signature with Hashing

## 59.1. Intent

The DIGITAL SIGNATURE WITH HASHING pattern allows a principal to prove that a message was originated from it. It also provides message integrity, by indicating whether a message was altered during transmission.

## 59.2. Example

Alice in the sales department wants to send a product order to Bob in the production department. The product order does not contain sensitive data such as credit card numbers, so it is not important to keep it secret. However, Bob wants to be certain that the message was created by Alice, so he can charge the order to her account. Also, because this order includes the quantity of items to be produced, an unauthorized modification to the order will make Bob manufacture the wrong quantity of items. Eve is a disgruntled employee who can intercept the messages and may want to attempt this kind of modification to hurt the company.

## 59.3. Context

People or systems often need to exchange documents or messages through insecure networks and need to prove their origin and integrity. Stored legal documents need to be kept without modification and with indication of their origin. Software sent by a vendor through the Internet is required to prove its origin.

We assume that those exchanging documents have access to a public key system where a principal possesses a key pair: a private key that is secretly kept by the principal, and a public key that is in a publicly accessible repository. We assume that there is a mechanism for the generation of these key pairs and for the distribution of public keys; that is, a public key infrastructure (PKI).

## 59.4. Problem

In many applications we need to verify the origin of a message (message authentication). Since an imposter may assume the identity of a principal, how can we verify that a message came from a particular principal? Also, messages that travel through insecure channels can be captured and modified by attackers. How can we know that the message or document that we are receiving has not been modified?

The solution to this problem must resolve the following forces:

- For legal or business reasons we need to be able to verify who sent a particular message. Otherwise, we may not be sure of its origin, and the sender may deny having sent it (repudiation).
- Messages may be altered during transmission, so we need to verify that the data is in its original form when it reaches its destination.

- The length of the signed message should not be significantly larger than the original message, otherwise we would waste time and bandwidth.
- Producing a signed message should not require large computational power or take a long time

## 59.5. Solution

Apply properties of public key cryptographic algorithms to messages in order to create a signature that will be unique for each sender [1]. The message is first compressed (hashed) to a smaller size (digest), then encrypted using the sender's private key. When the signed message arrives at its target, the receiver verifies the signature using the sender's public key to decrypt the message. If it produces a readable message, it could only have been sent by this sender. The receiver then generates the hashed digest of the received message and compares it with the received hashed digest: if it matches, the message has not been altered.

This approach uses public key cryptography in which one key is used for encryption and the other for decryption. To produce a digital signature (SIG), we encrypt (E) the hash value of a message ( $H(M)$ ) using the sender's private key (PrK):

$$SIG = E_{PrK}(H(M))$$

We recover the hash value of the message ( $H(M)$ ) by applying decryption function D to the signature (SIG) using the sender's public key (PuK). If this produces a legible message, we can be confident that the sender created the message, because they are the only one who has access to their private key. Finally, we calculate the hash value of the message as:

$$H(M) = D_{PuK}(SIG)$$

If this value is the same as the message digest obtained when the signature was decrypted, then we know that the message has not been modified.

It is clear that the sender and receiver should agree to use the same encryption and hashing algorithms.

## 59.6. Structure

Figure 118 shows the class diagram for the DIGITAL SIGNATURE WITH HASHING pattern. A Principal may be a process, a user or an organization that is responsible for sending or receiving messages. This Principal may have the roles of Sender or Receiver. A Sender may send a plain Message and/or a SignedMessage to a receiver.

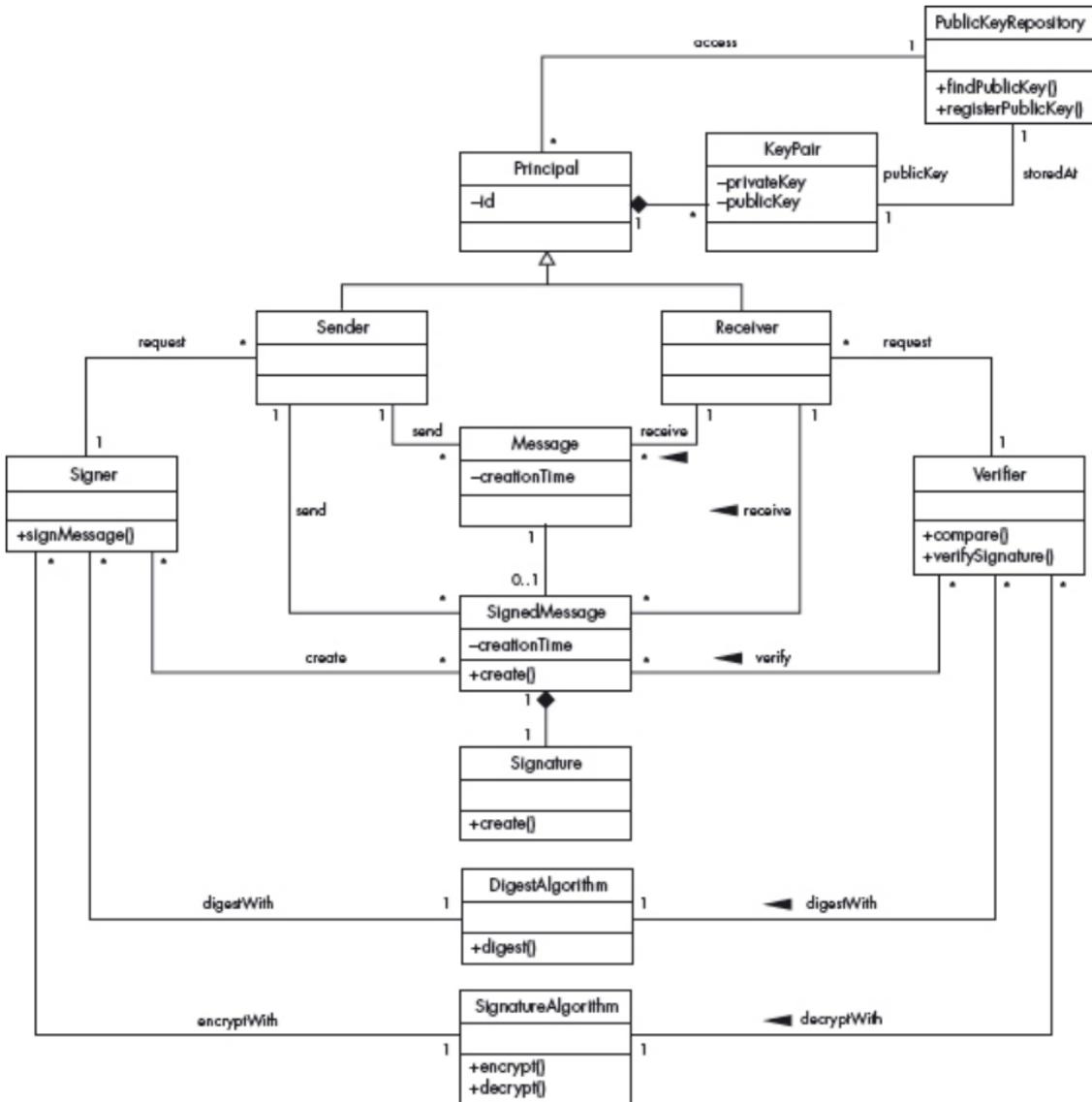


Figure 118: shows the class diagram for the DIGITAL SIGNATURE WITH HASHING pattern.

The **KeyPair** entity contains two keys, public and private, that belong to a **Principal**. The public key is registered and accessed through a repository, while the private key is kept secret by its owner. **PublicKeyRepository** is a repository that contains public keys. The **PublicKeyRepository** may be located in the same local network as the **Principal**, or on an external network.

The **Signer** creates the **SignedMessage** that includes the **Signature** for a specific **message**. On the other side, the **Verifier** checks that the **Signature** within the **SignedMessage** corresponds to that message. The **Signer** and **Verifier** use the **DigestAlgorithm** and **SignatureAlgorithm** to create and verify a signature respectively. The **DigestAlgorithm** is a hash function that condenses a message to a fixed length called a hash value, or message digest. The **SignatureAlgorithm** encrypts and decrypts messages using public/private key pairs.

## 59.7. Dynamics

We describe the dynamic aspects of the DIGITAL SIGNATURE WITH HASHING pattern using sequence diagrams for the use cases ‘Sign a message’ and ‘Verify a signature’.

Use Case:	Sign a Message – Figure 119
Summary	A Sender wants to sign a message before sending it.
Actors	A Sender.
Precondition	A Sender has a public/private pair key.
Description	<ol style="list-style-type: none"> <li>1. A Sender sends the message and its private key to the Signer.</li> <li>2. The Signer calculates the hash value of the message (digest) and returns it to the Sender.</li> <li>3. The Signer encrypts the hash value using the Sender's private key with the SignatureAlgorithm. The output of this calculation is the digital signature value.</li> <li>4. The Signer creates the Signature object that contains the digital signature value.</li> <li>5. The Signer creates the SignedMessage that contains the original message and the Signature.</li> </ol>
Postcondition	A SignedMessage object has been created.

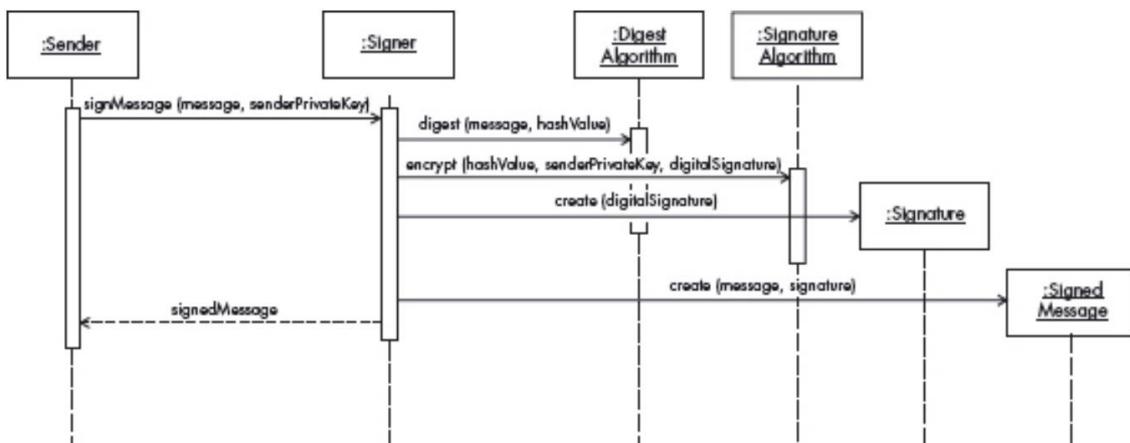


Figure 119: Sequence diagram for the use case 'Sign a message'

Use Case:	Verify a Signature – Figure 120
Summary	A Receiver wants to verify that the signature corresponds to the received message.
Actors	A Receiver.
Precondition	None.
Description	<ol style="list-style-type: none"> <li>1. A Receiver retrieves the Sender's public key from the PublicKeyRepository.</li> <li>2. A Receiver sends the signed message and the Sender's public key to the Verifier.</li> <li>3. The Verifier decrypts the signature using the Sender's public key with the SignatureAlgorithm.</li> <li>4. The Verifier calculates the digest value of the message.</li> <li>5. The Verifier compares the outputs from step 3 and 4.</li> <li>6. The Verifier sends an acknowledgement to the Receiver that the signature is valid.</li> </ol> <ul style="list-style-type: none"> <li>• The outputs from steps 3 and 4 are not the same. In this case, the verifier sends an acknowledgement to the receiver that the signature failed.</li> </ul>
Alternate Flows	
Postcondition	The signature has been verified.

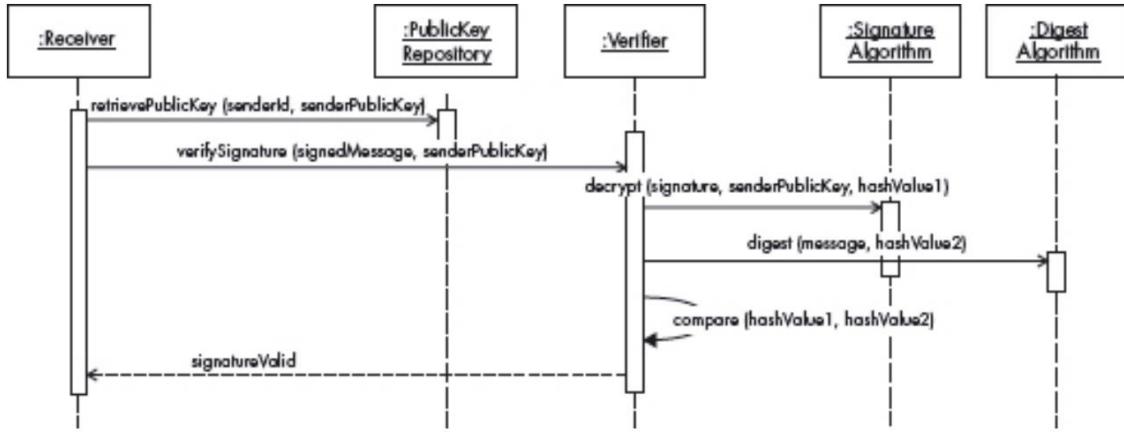


Figure 120: Sequence diagram for the use case 'Verify a signature'

## 59.8. Implementation

- Use the Strategy pattern [2] to select different hashing and signature algorithms. The most widely used hashing algorithms are MD5 and SHA1. These and others are discussed in [1].
- A good hashing algorithm produces digests that are very unlikely to be produced by other meaningful messages, meaning that it is very hard for an attacker to create an altered message with the same hash value. The message digest should be encrypted after being signed to avoid man-in-the-middle attacks, where someone who captures a message could reconstruct its hash value.
- Two popular digital signature algorithms are RSA [3,4] and Digital Signature Algorithm (DSA) [1,5].
- The designer should choose strong and proven algorithms to prevent attackers from breaking them. The cryptographic protocol aspects, for example key generation, are as important as the algorithms used.
- The sender and receiver should have a way of agreeing on the hash and encryption algorithms used for a specific set of messages. (XML documents indicate which algorithms they use, and pre-agreements are not necessary in this case.)
- Access to the sender's public key should be available from a public directory or from certificates presented by the signer.
- Digital signatures can be implemented in different applications, such as in e-mail communication, distribution of documents over the Internet, or web services. For example, it is possible to sign an email's contents, or any other document's content, such as a PDF. In both cases, the signature is appended to the e-mail or document. When digital signatures are applied in web services, they are also embedded within XML messages. However, these signatures are treated as XML elements, and they have additional features, such as signing parts of a message, or external resources, which can be XML or any other data type.
- When certificates are used to provide the sender's public key, there must be a convenient way to verify that the certificate is still valid [6].
- There should be a way of authenticating the signer software [7], as an attacker who gains control of a user's computer could replace the signing software with their own software.

## **59.9. Example Resolved**

Alice and Bob agree on the use of a digital signature algorithm, and Bob has access to Alice's public key. Alice can then send a signed message to Bob. When the message is received by Bob, he verifies that the signature is valid using Alice's public key and the agreed signature algorithm. If the signature is valid, Bob can be confident that the message was created by Alice. If the hash value is correct, Bob also knows that Eve has not been able to modify the message.

## **59.10. Consequences**

The DIGITAL SIGNATURE WITH HASHING pattern offers the following benefits:

- Because a principal's private key is used to sign the message, the signature can be validated using its public key, which proves that the sender created and sent the message.
- When a signature is validated using a principal's public key, the sender cannot deny that they created and sent the message (nonrepudiation). If a message is signed using another private key that does not belong to the sender, the validity of the signature fails.
- If the proper precautions are followed, any change in the original message will produce a digest value that will be different (with a very high probability) from the value obtained after decrypting the signature using the sender's public key.
- A message is compressed into a fixed length string using the hash algorithm before it is signed. As a result, the process of signing is faster, and the signed message is much shorter.
- The available algorithms that can be used for digital signatures do not require very large amounts of computational power and do not take large amounts of time.

The pattern also has the following potential liabilities:

- We need a well-established public key infrastructure that can provide reliable public keys. Certificates issued by a certification authority are the most common way to obtain this [1].
- Both the sender and the receiver have to previously agree what signature and hashing algorithms they support. (This is not necessary in XML documents, because they are self-describing.)
- Cryptographic algorithms create some overhead (time, memory, computational power), which can be reduced but not eliminated.
- The required storage and computational power may not be available, for example in mobile devices.
- Users must implement the signature protocol properly.
- There may be attacks against specific algorithms or implementations [7]. These are difficult to use against careful implementations of this pattern.
- This solution only allows one signer for the whole message. A variant or specialization, such as the XML SIGNATURE pattern [8], allows multiple signers.
- Digital signatures do not provide message authentication, and replay attacks are possible [6]. Nonces or time stamps could prevent this type of attack.

## 59.11. Known Uses

Digital signatures have been widely used in different products.

- Adobe Reader and Acrobat [9] have an extended security feature that allows users to digitally sign PDF documents.
- CoSign [10] digitally signs different types of documents, files, forms, and other electronic transactions.
- GNuPG [11] digitally signs e-mail messages.
- The Java Cryptographic Architecture [12] includes APIs for digital signature.
- Microsoft .NET [13] includes APIs for asymmetric cryptography such as digital signature.
- XML Signature [14] is one of the foundation web services security standards that defines the structure and process of digital signatures in XML messages.

## 59.12. See Also

- Asymmetric encryption pattern.
- Generation and distribution of public keys [15].
- Certificates are issued by a certificate authority (CA) that digitally signs them using its private key. A certificate carries a user's public key and allows anyone who has access to the CA's public key to verify that the certificate was signed by the CA.
- The Strategy pattern [2] describes how to separate the implementation of related algorithms from the selection of one of them.

## 59.13. References

- [1] Stallings, W. (2006). *Cryptography and network security principles and practices* 4th edition.
- [2] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [3] Rivest, R. L., Shamir, A., & Adleman, L. M. (2019). A method for obtaining digital signatures and public key cryptosystems. In *Secure communications and asymmetric cryptosystems* (pp. 217-239). Routledge.
- [4] RSA. (n.d.). RSA Security, PKCS #1: RSA Cryptography Standard. from <http://www.rsa.com/rsalabs/node.asp?id=2125>
- [5] NIST. (2000, January 27). Federal Information Processing Standard, Digital Signature Standard. from <http://csrc.nist.gov/publications/fips/fips186-2/fips186-2-change1.pdf>
- [6] W3C. (2001, February). SOAP Security Extensions: Digital Signature. W3C NOTE 06. <http://www.w3.org/TR/SOAP-dsig/>
- [7] Wikipedia contributors. (n.d.). Digital signature. Wikipedia. [https://en.wikipedia.org/wiki/Digital\\_signature](https://en.wikipedia.org/wiki/Digital_signature)
- [8] Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

- [9] Adobe. (n.d.). About certificate signatures in Adobe Acrobat.  
<https://helpx.adobe.com/acrobat/kb/certificate-signatures.html>
- [10] Arx. (n.d.). Digital Signature Solution (Standard Electronic Signatures).  
<http://www.arx.com/products/cosign-digital-signatures.php>
- [11] The GNU Privacy Guard. (n.d.). GnuPG. <https://www.gnupg.org/>
- [12] Sun Microsystems Inc. (n.d.). Java SE Security.  
<http://java.sun.com/javase/technologies/security/>
- [13] Microsoft Corporation. (2007, November). .NET Framework Class Library.  
<http://msdn.microsoft.com/enus/library/ms229335.aspx>
- [14] W3C Working Group. (2008). XML Signature Syntax and Processing (2nd edition).  
<http://www.w3.org/TR/xmldsig-core>
- [15] Lehtonen, S., & Pärssinen, J. (2002, June). Pattern Language for Cryptographic Key Management. In *EuroPLoP* (pp. 245-258).

## 59.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **60. Identity Federation**

## **60.1. Intent**

The IDENTITY FEDERATION pattern allows the formation of a dynamically created identity within an identity federation consisting of several service providers. Therefore, identity and security information about a subject can be transmitted in a transparent way for the user among service providers from different security domains.

## **60.2. Example**

Having a centralized identity is not much good unless we can use it in many places. We need some structure to allow this behavior.

## **60.3. Context**

We have several security domains in a distributed environment that trust each other. A security domain is a set of resources (web services, applications, CORBA services and so on) in which administration of security is performed by a unique entity, which typically stores identity information about the subjects known to the domain. Subjects can perform actions in one or more security domains.

## **60.4. Problem**

There may be no relationship between some of the security domains accessed by a subject. Thus, subjects may have multiple unrelated identities within each security domain. Consequently, they may experience multiple and cumbersome registrations, authentications, and other identity related tasks prior to accessing the services they need.

How can we avoid the inconvenience of multiple registrations and authorizations across security domains?

The solution to this problem must resolve the following forces:

- The identity of a user can be represented in a variety of ways in different domains.
- Parts of a subject's identity within a security domain may include sensitive information that should not be disclosed to other security domains.
- The identity and security-related information in transit between two security domains should be kept confidential, so that eavesdropping, tampering or identity theft cannot be realized.
- A subject may want to access a security domain's resources in an anonymous way.

## **60.5. Solution**

Service providers, which are normally part of a security domain in which the local identity of subjects is managed by an identity provider, form identity federations by developing offline

operating agreements with other service providers from other security domains. In particular, they can agree about their privacy policies.

In a security domain, the local identity associated with a user consists of a set of attributes. Some of those attributes can be marked as confidential and should not be passed to other security domains. A federated identity is created gradually and transparently by gathering some of a subject's attributes from its local identities within an identity federation. Therefore, identity and security information about a subject can be transmitted between service providers from the same identity federation transparently to the user. In particular, its authentication status can be propagated to perform single sign-on within the identity federation.

Figure 121 illustrates an example of how security domains and identity federations can coexist: a security domain is typically a circle of trust within an organization, whereas an identity federation is a circle of trust whose members can come from different organizations.

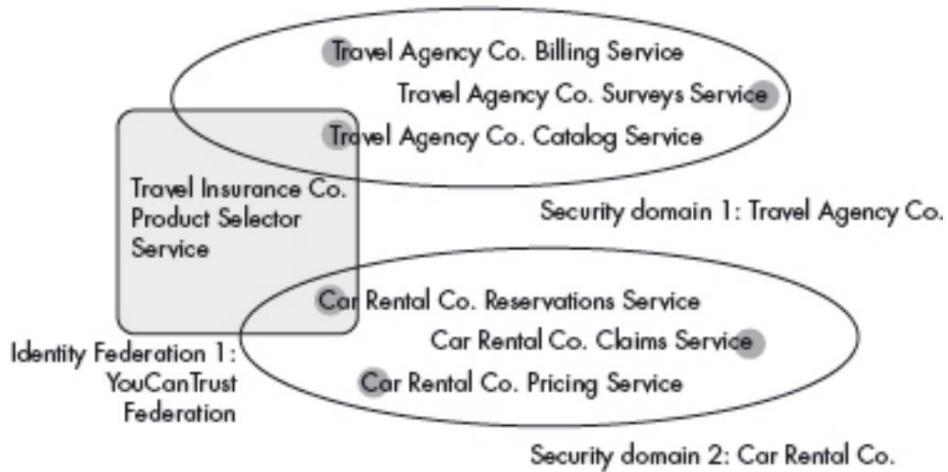
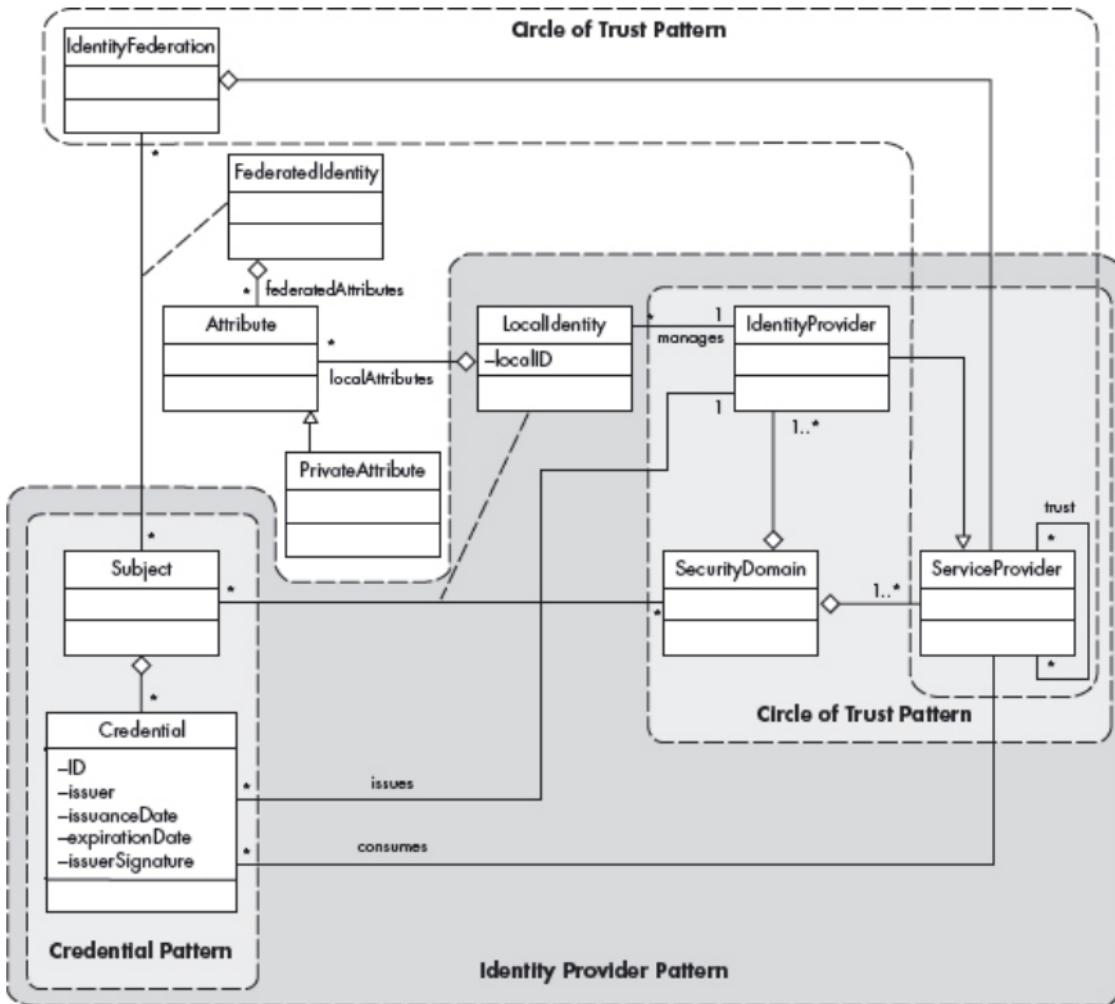


Figure 121: Federation and domains

## 60.6. Structure

Figure 122 shows a UML class diagram with an OCL constraint describing the structure of the solution. An IdentityFederation consists of a set of ServiceProvider which provide services to Subject. A Subject has multiple LocalIdentities with some ServiceProvider. A LocalIdentity can be described as a set of Attributes of the Subject.



```

context FederatedIdentity
Inv: forall(p | self.federatedAttributes -> includes(p)) Implies
    self.subject.localIdentity.localAttributes -> includes(p))
Inv: self.federatedAttributes -> excludes(
    self.subject.localIdentity.localAttributes.oclAsType)
    [PrivateAttribute])
  
```

Figure 122: Class diagram for the IDENTITY FEDERATION pattern

A Subject can have several FederatedIdentities. This FederatedIdentity is composed of a union of attributes from the Localidentities of the IdentityFederation. An IdentityProvider is responsible for managing the Localidentities within a SecurityDomain and can authenticate any Subjects on behalf of any ServiceProvider of the IdentityFederation. A Subject has been issued a set of Credentials that collect information about its authentication status and its identity within a SecurityDomain.

## 60.7. Dynamics

We illustrate the dynamic aspects of the IDENTITY FEDERATION pattern by showing sequence diagrams for two use cases: ‘Federate two local identities’ (Figure 123) and ‘Single sign on’ (Figure 124). The first use case describes how a local entity invites another to join, and their mutual authentication. The second use case shows how a subject, after receiving a credential from an identity provider, uses it to access a new domain.

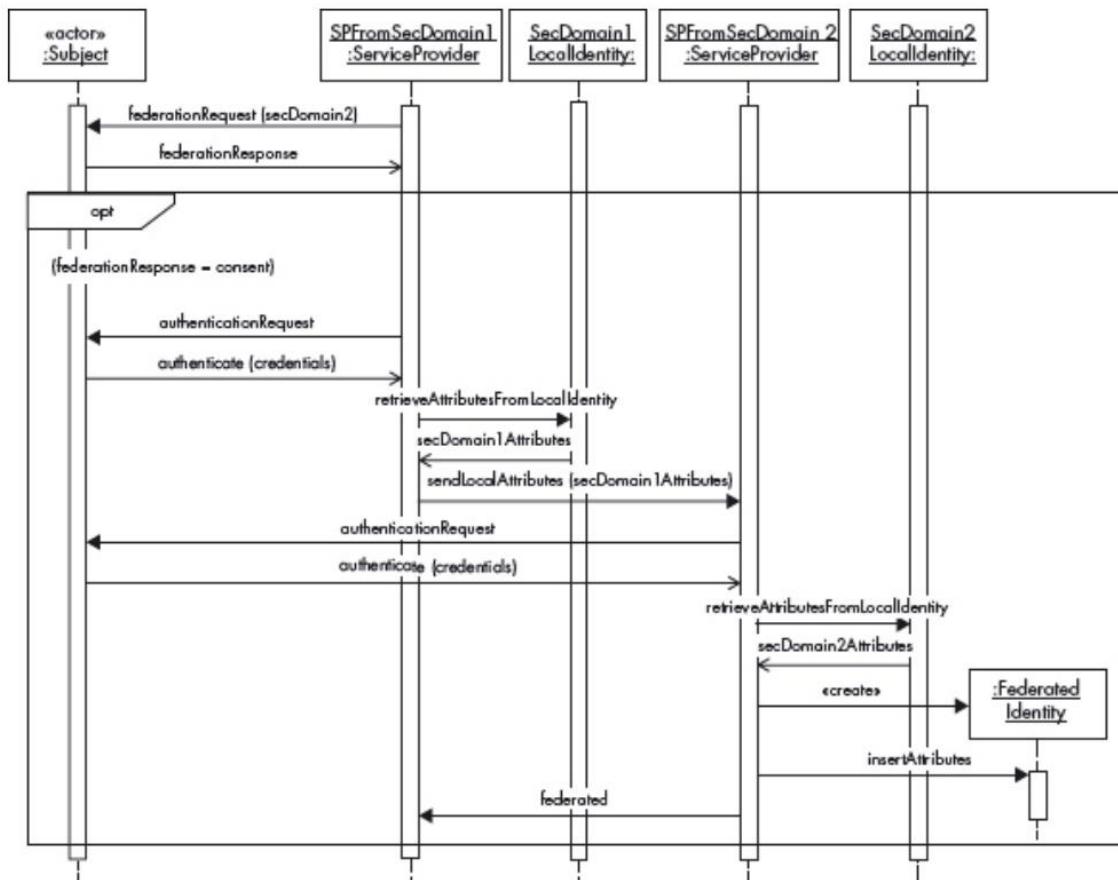


Figure 123: Sequence diagram for the use case 'Federate two local identities'

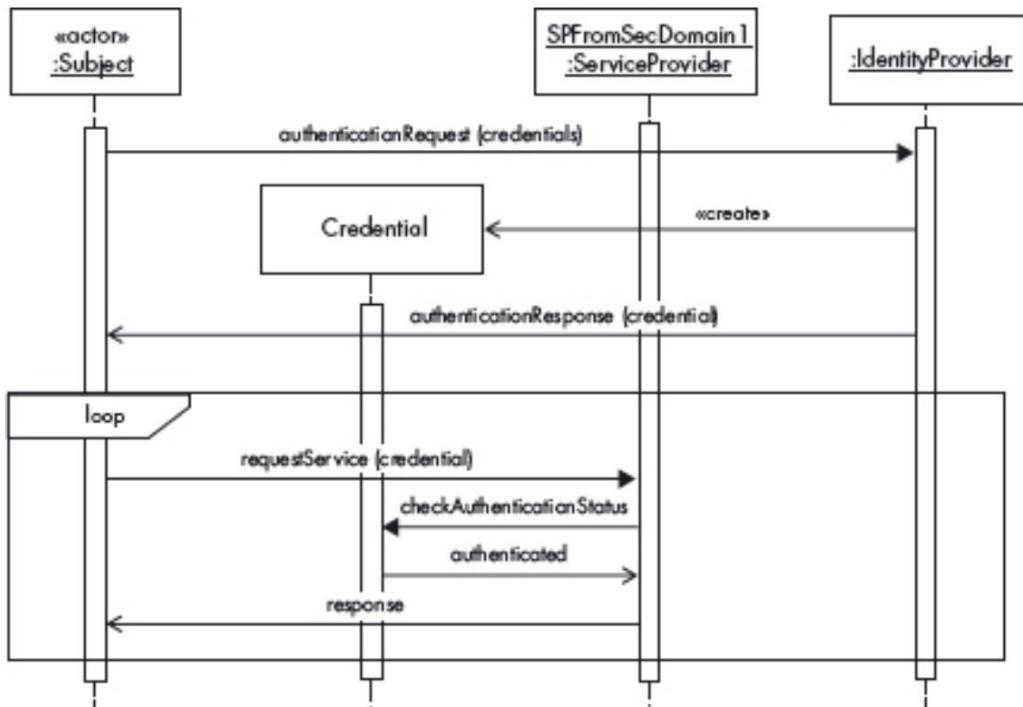


Figure 124: Sequence diagram for the use case 'Single sign on'

## **60.8. Implementation**

The identity federation can be structured hierarchically or in a peer-to-peer manner. The most basic federation could be based on bilateral agreements between two service providers. In the LIBERTY ALLIANCE IDENTITY FEDERATION pattern [1], an Identity Federation has an Identity Provider responsible for managing the federated identity, whereas Shibboleth [2] defines a club as a set of service providers that has reached some operating agreements.

In the LIBERTY ALLIANCE IDENTITY FEDERATION model, the identity provider proposes incentives for other service providers to affiliate with them and federate their local identities. Furthermore, no attribute can be classified as private: privacy is achieved by letting the user provide their consent each time its identity is federated.

## **60.9. Example Resolved**

Having implemented an IDENTITY FEDERATION, we can visit different services using the same identity for all of them.

## **60.10. Consequences**

The IDENTITY FEDERATION pattern offers the following benefits:

- Subjects can access resources within the identity federation in a seamless and secure way without reauthenticating in each new domain.
- Many different representations of the identity of a user can be consolidated under the same federated identity.
- Subjects can classify some of their attributes as private. Therefore, an identity provider can identify which attributes it should transmit to other parties and which it should not.
- Parts of the security credentials issued about a subject can be encrypted, so that the subject's privacy can be protected.
- The security credential can be signed, so that its integrity and authenticity is protected, and attackers cannot forge security tokens or change some of the subject's attributes.
- A subject can access a security domain's resources in an anonymous way, since not all attributes from a local identity (such as a name) are required to be federated.

The pattern also has some potential liabilities:

- Service providers need to have some kind of agreement before their identities can be federated. They have to exchange credentials through some external channel to trust and recognize each other.
- Even when a subject's sensitive information is classified as private, a security domain can still disclose the subject's private information secretly to other parties, thus violating the subject's privacy.
- A security token can be stolen and presented by an attacker, resulting in identity theft. This is alleviated by the use of expiration dates and unique IDs for credentials.
- In spite of an expiration date and the unique ID feature of a credential, which guarantees its freshness, the unconditional revocation of a credential is not addressed in this solution.

## **60.11. Known Uses**

- WS-Federation [3] is a proposed standard allowing web services to federate their identities.
- Liberty Alliance is a standard that allows services to federate into Identity Federations [4].
- Microsoft Account (previously Microsoft Wallet, Microsoft Passport, .NET Passport, Microsoft Passport Network and Windows Live ID) is an identity federation that lets users log in to a variety of web sites using only one account [5].
- Shibboleth is an open solution for realizing identity federation among enterprises [2].

## **60.12. See Also**

- LIBERTY ALLIANCE IDENTITY FEDERATION is a specialization of this pattern [1].
- CREDENTIAL pattern.
- A Single Sign On pattern is shown in [6].
- A formal description of identity management patterns is given in [7].

## **60.13. References**

- [1] Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.
- [2] InCommon. (n.d.-b). Shibboleth Project. <http://shibboleth.internet2.edu/>
- [3] Ajaj, O., & Fernandez, E. B. (n.d.). A pattern for the WS-Federation standard for web services. in preparation
- [4] Liberty Alliance Project. (n.d.). <http://www.projectliberty.org/>
- [5] Microsoft Architecture Journal. (n.d.). Identity and Access. Journal 16
- [6] Steel, C., & Nagappan, R. (2006). *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Pearson Education India.
- [7] Nizamani, H. A., & Tuosto, E. (2011). Patterns of Federated Identity Management Systems as Architectural Reconfigurations. *Electronic Communications of the EASST*, 31.

## **60.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 61. Identity Provider

## 61.1. Intent

The IDENTITY PROVIDER pattern allows the centralization of the administration of subjects' identity information for a security domain.

## 61.2. Example

Having applied the CIRCLE OF TRUST pattern, we can trust the identity providers who are members of our federation, but we still have a variety of identities.

## 61.3. Context

One or several resources, such as web services, CORBA services, applications and so on that are accessed by a predetermined set of subjects. The subjects and resources are typically from the same organization.

## 61.4. Problem

Each application or service may implement its own code for managing subjects' identity information, leading to an overloading of implementation and maintenance costs that may lead to inconsistencies across the organization's units.

## 61.5. Solution

The management of the subjects' information for an organization is centralized in an IDENTITY PROVIDER, which is responsible for storing and propagating parts of the subjects' information (that form their identity) to the applications and services that need it.

We define a security domain as the set of resources whose subjects' identities are managed by the IDENTITY PROVIDER. Typically, the IDENTITY PROVIDER issues a set of credentials to each subject that will be verified by the accessed resources. Notice that the security domain is a special kind of CIRCLE OF TRUST within an organization.

## 61.6. Structure

Figure 125 shows a UML class diagram describing the structure of the solution. This pattern combines the CIRCLE OF TRUST, making explicit its Resources and specializing ServiceProvider to the IdentityProvider. Subjects are identified using CREDENTIALs given by a specific identity provider.

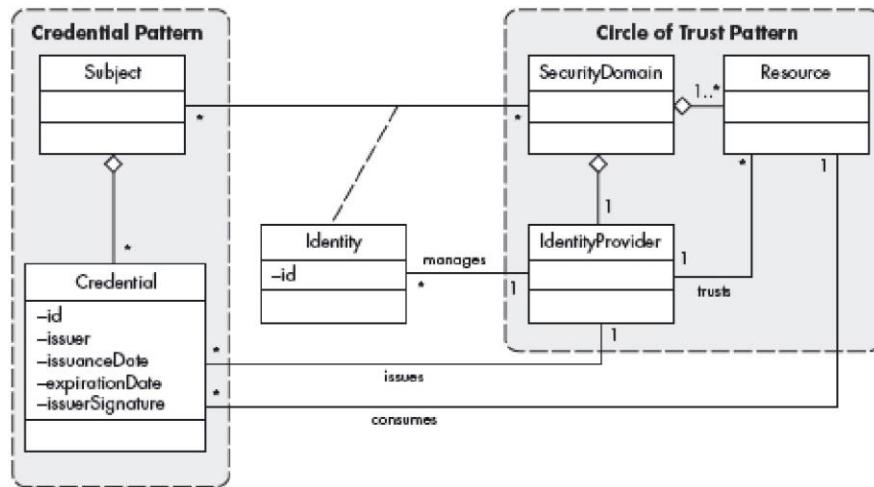


Figure 125: Class diagram for the IDENTITY PROVIDER pattern

## 61.7. Dynamics

## 61.8. Implementation

## 61.9. Example Resolved

Now that we can trust the identity providers who are members of our federation and have a centralized identity managed by a specific identity provider, we do not need multiple identities.

## 61.10. Consequences

The IDENTITY PROVIDER pattern offers the following benefits:

- Maintenance costs are reduced.
- The system is consistent in terms of its users.

It also suffers from the following liability:

- The IDENTITY PROVIDER can be a bottleneck in the organization's network.

## 61.11. Known Uses

- Identity management products, such as IBM Tivoli [1], Sun One Identity Server [2], Netegrity's SiteMinder and WSO2 [3].
- SAP NetWeaver Identity Management [4] manages user access to applications by providing a central mechanism for provisioning users in accordance with their current business roles. It also supports related processes, such as password management, self-service, and approvals workflow.

- The Empower Identity Manager is oriented to cloud computing [5].

## 61.12. See Also

IDENTITY FEDERATION uses this pattern, as well as the CREDENTIAL and CIRCLE OF TRUST patterns.

## 61.13. References

[1] IBM. (n.d.). Tivoli Federated Identity Manager.

<http://www306.ibm.com/software/tivoli/products/federated-identity-mgr/>

[2] Sun. (n.d.). Java System Access Manager.

[http://www.sun.com/software/products/access\\_mngr/](http://www.sun.com/software/products/access_mngr/)

[3] WSO2. (n.d.). Identity Server. <http://wso2.com/products/identity-server>

[4] SAP. (n.d.). Netweaver Identity Manager.

<http://www.sap.com/platform/netweaver/components/idm/index.epx>

[5] Empower Identity Manager. (n.d.).

[http://www.identitymanagement.com/?\\_kk=identity%20management&\\_kt=d37d8c67-315a4919-abfc-41011051bd9e&gclid=CNrgw7ylng8CFcNa7Aod90Shaw](http://www.identitymanagement.com/?_kk=identity%20management&_kt=d37d8c67-315a4919-abfc-41011051bd9e&gclid=CNrgw7ylng8CFcNa7Aod90Shaw)

## 61.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 62. Layered Operating System Architecture

## 62.1. Intent

The LAYERED OPERATING SYSTEM ARCHITECTURE pattern allows the overall features and functionality of the operating system to be decomposed and assigned to hierarchical layers. This provides clearly defined interfaces between each section of the operating system and between user applications and the operating system functions. Layer  $i$  uses the services of a lower layer  $i-1$  and does not know of the existence of a higher layer,  $i+1$ .

## 62.2. Example

Our operating system is very complex, and we would like to separate different aspects in order to handle them in a more systematic way. Complexity brings vulnerability. We also want to control the calls between operating system components and services to improve security and reliability. Finally, we would like to hide critical modules. We tried a modular architecture, but it did not have enough structure to do all this systematically.

## 62.3. Context

A variety of applications with diverse requirements that need to execute together sharing hardware resources.

## 62.4. Problem

Unstructured modules, as in the MODULAR OPERATING SYSTEM ARCHITECTURE pattern, have the problem that all modules can reach all other modules, which facilitates attacks. We need to conceal the existence of some critical modules.

The solution to this problem must resolve the following forces:

- Interfaces should be stable and well-defined. Going through any interface could imply authorization checks.
- Parts of the system should be exchangeable or removable without affecting the rest of the system. For example, we could have modules that perform more security checks than others.
- Similar responsibilities should be grouped, to help understandability and maintainability. This contributes indirectly to improved security.
- We should control module visibility to avoid possible attacks from other modules.
- Complex components need further decomposition. This makes the design simpler and clearer and also improves security.

## 62.5. Solution

Define a hierarchical set of layers and assign components to each layer. Each layer presents an abstract machine (a set of operations) to the layer above it, hiding the implementation details of the lower layers.

## 62.6. Structure

Figure 126 shows a class diagram for the LAYERED OPERATING SYSTEM ARCHITECTURE pattern. LayerN represents the highest level of abstraction, and Layer1 is the lowest level of abstraction. The main structural characteristic is that the services of LayerN are used only by LayerN+1. Each layer may contain complex entities consisting of different components.

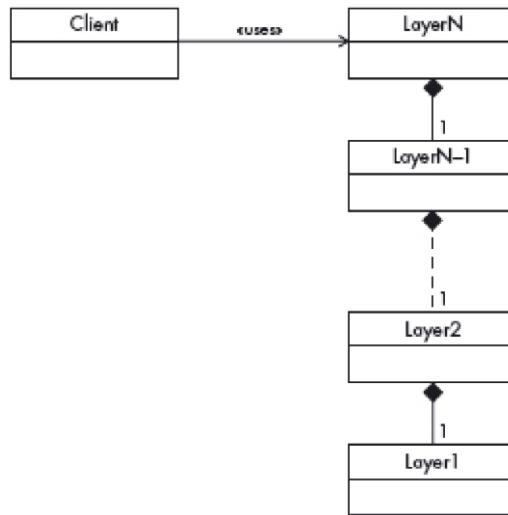


Figure 126: Class diagram for LAYERED OPERATING SYSTEM ARCHITECTURE pattern

## 62.7. Dynamics

Figure 127 shows the sequence diagram for the use case ‘Open and read a disk file’:

- A user sends an openFile() request to the OSInterface.
- The OSInterface interprets the openFile() request.
- The openFile() request is sent from the OSInterface to the FileManager.
- The FileManager sends a readDisk() request to the DiskDriver.

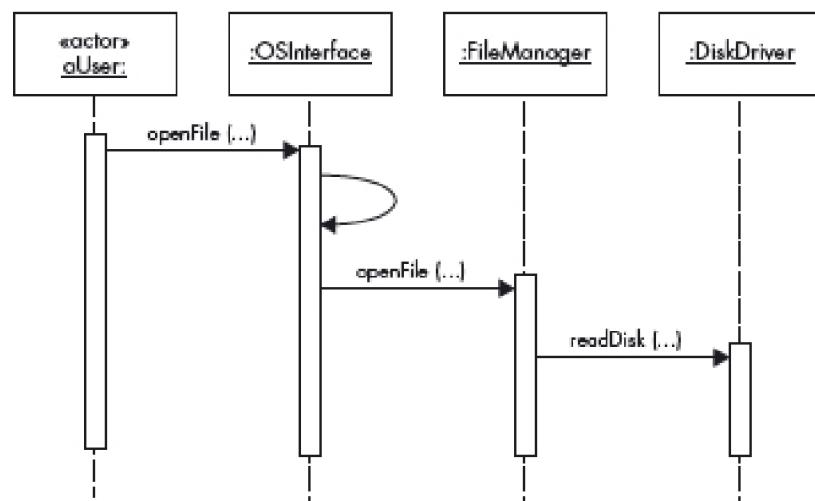


Figure 127: Sequence diagram for the use case ‘Open and read a disk file’

## 62.8. Implementation

1. List all units in the system and define their dependencies.
2. Assign units to levels such that units in higher levels depend only on units of lower levels.
3. Once the modules in a given level are assigned, define a language (set of commands) for the level. This language includes the operations that we want to make visible to the next level above. Add well-defined operation signatures and security checks in these operations to assure the proper use of the level.
4. Hide those modules that control critical security functions in lower levels.

## 62.9. Example Resolved

We structured the functions of our system as shown in Figure 128, and now we have a way to control interactions and enforce abstraction. For example, the file system can use the operations of the disk drives and enforce similar restrictions in the storage of data.

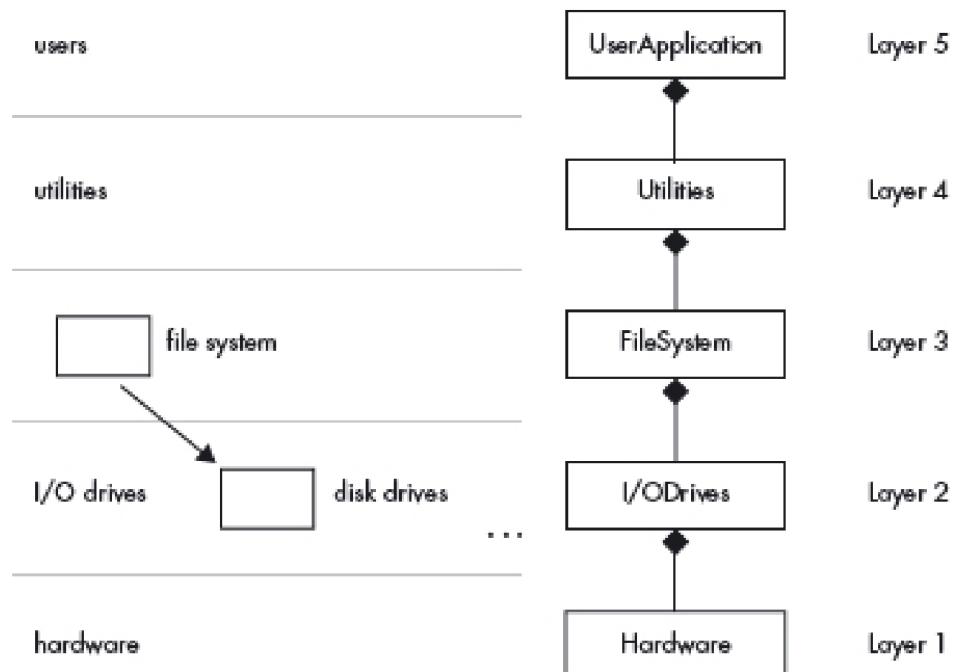


Figure 128: An example of the use of a layered OS architecture

The user of a file cannot take advantage of the implementation details of the disk driver to attack the system.

## 62.10. Consequences

The LAYERED OPERATING SYSTEM ARCHITECTURE pattern offers the following benefits:

- Lower levels can be changed without affecting higher layers. We can add or remove security functions as needed.
- There are clearly defined interfaces between each operating system layer and the user applications, which improves security.

- Control of information is possible using layer hierarchical rules and enforcement of security policies between layers.
- The fact that layers hide implementation aspects is useful for security, in that possible attackers cannot exploit lower-level details.

The pattern also has the following potential liabilities:

- It may not be clear what to put in each layer. In particular, related modules may be hard to allocate. There may be conflicts between functional and security needs when allocating modules.
- Performance may decrease due to the indirection of calls through several layers. If we try to improve performance, we may sacrifice security.

## 62.11. Known Uses

- The Symbian operating system (Figure 129) uses a variation of the layered approach [1].

Connectivity Framework			Connectivity Plug-ins							
Application Services	Application Protocols	Application Engines	Messaging	WAP Browser	Web Browser	JavaPhone				
			Narrowband Protocol	WAP Stack	Web Stack	Infrared	Bluetooth	Networking		
	Multimedia		Comms Infrastructure							
	Graphics		Security	Connectivity Link	Serial Comms	Telephony	Base			

Figure 129: Symbian operating system layered architecture

- The UNIX operating system (Figure 130) is separated into four layers, with clear interfaces between the system calls to the kernel and between the kernel and the hardware.

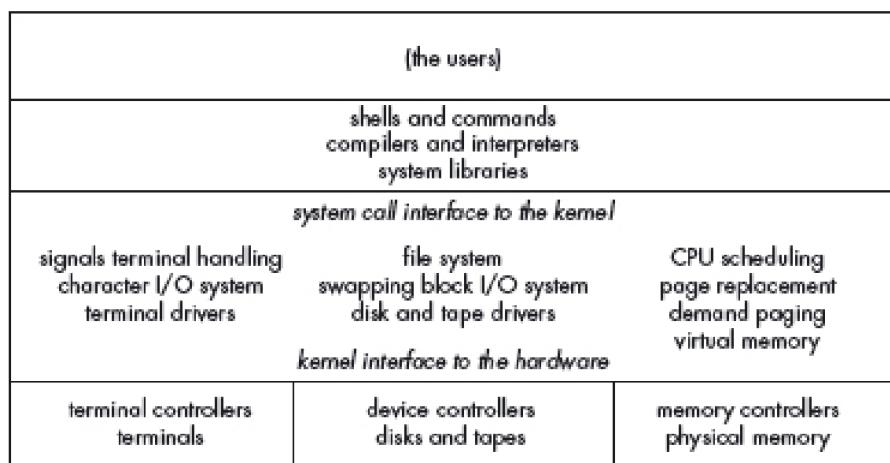


Figure 130: UNIX layered architecture

- IBM's OS/2 also uses this approach [2].

## 62.12. Variants

Layer skipping. In this architecture there are special applications that are able to skip layers for added performance. This structure requires a tradeoff between performance and security. By deviating from the strict hierarchy of the layered system, there may not be enforcement of security policies between layers for such applications.

## 62.13. See Also

This pattern is a specialization of the Layers architectural pattern [3].

## 62.14. References

[1] Symbian. (n.d.). <http://www.symbian.com/developer/>

[2] IBM. (n.d.). <http://www-306.ibm.com/software/os/warp/>

[3] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-Oriented Software Architecture: A System of Patterns, Volume 1. John Wiley & Sons Inc.

## 62.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 63. Microkernel Operating System Architecture

## 63.1. Intent

The MICROKERNEL OPERATING SYSTEM ARCHITECTURE pattern describes how to move as much of the operating system functionality as possible from the kernel into specialized servers, coordinated by a microkernel. The microkernel itself has a very basic set of functions. Operating system components and services are implemented as external and internal servers.

## 63.2. Example

We are building an operating system to support a range of applications with different reliability and security requirements and a variety of plugins. We would like to provide operating system versions with different types of modules: some more secure, some less so.

## 63.3. Context

A variety of applications with diverse requirements that need to execute together sharing hardware resources.

## 63.4. Problem

In general-purpose environments we need to be able to add new functionality with variation in security and other requirements, as well as provide alternative implementations of services to accommodate different application requirements.

The solution to this problem must resolve the following forces:

- The application platform must be able to cope with continuous hardware and software evolution: these additions may have very different security or reliability requirements.
- Strong security or reliability requirements indicate the need for modules with well-defined interfaces.
- We may want to perform different types of security checks in different modules, depending on their security criticality.
- We would like a minimum of functionality in the kernel, so that we have a minimum of processes running in supervisor mode. A simple kernel can be checked for possible vulnerabilities, which is good for security.

## 63.5. Solution

Separate all functionality into specialized services with well-defined interfaces and provide an efficient way to route requests to the appropriate servers. Each server can be built with different security constraints. The kernel mainly routes requests to servers and has minimal functionality.

## 63.6. Structure

The Microkernel is the central communication for the operating system. There is one Microkernel and several InternalServers and ExternalServers, each providing a set of specialized services (Figure 131). In addition to the servers, an Adapter is used between the Client and the Microkernel or an external server. The Microkernel controls the internal servers.

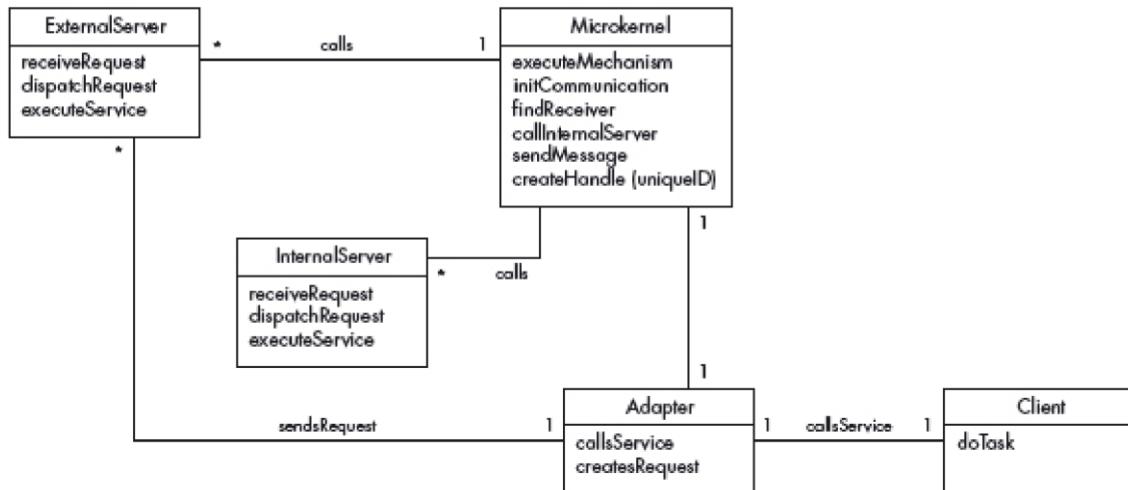


Figure 131: Class diagram for MICROKERNEL OPERATING SYSTEM ARCHITECTURE pattern

## 63.7. Dynamics

A client requests a service from an external server using the following sequence (Figure 132):

1. The Adapter receives the request from the Client and asks the Microkernel for a communication link with the ExternalServer.
2. The Microkernel checks for authorization to use the server, determines the physical address of the ExternalServer and returns it to the Adapter.
3. The Adapter establishes a direct communication link with the ExternalServer.
4. The Adapter sends the request to the ExternalServer using a procedure call or a remote procedure call (RPC). The RPC can be checked for well-formed commands, correct size, and type of parameters (that is, we can check signatures).
5. The ExternalServer receives the request, unpacks the message, and delegates the task to one of its own methods. All results are sent back to the Adapter.
6. The Adapter returns to the Client, which in turn continues with its control flow.

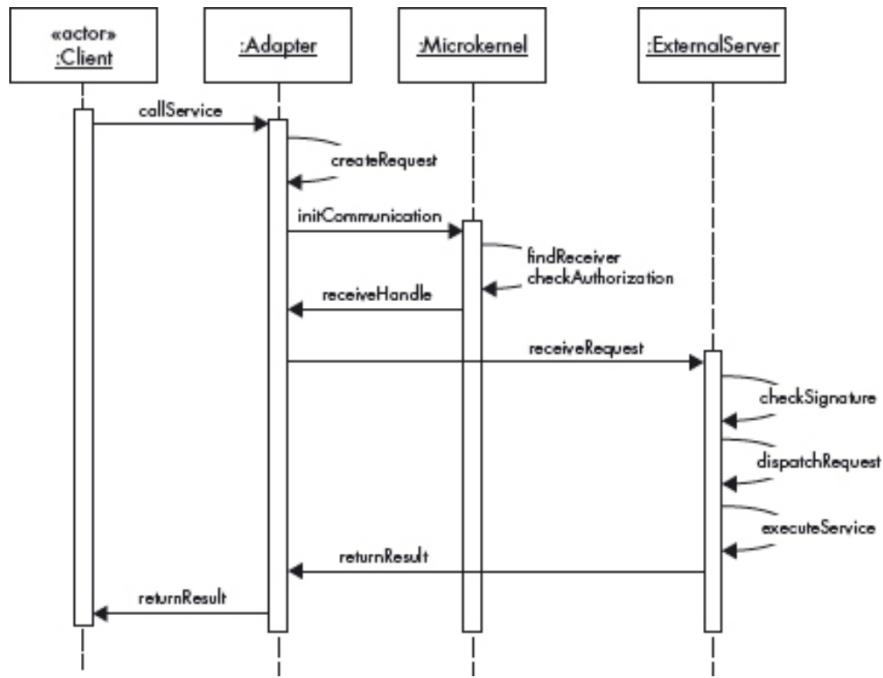


Figure 132: Sequence diagram for an operating system call through the microkernel

## 63.8. Implementation

1. Identify the core functionality necessary for implementing external servers and their security constraints. Typically, basic functions of the operating system should be internal servers; utilities, or user-defined services should go into external servers.
2. Define policies to restrict access to external and internal servers. Clients may be allowed to call only specific servers.
3. Find a complete set of operations and abstractions for every category of server identified.
4. Determine strategies for request transmission and retrieval.
5. Structure the microkernel component. The microkernel should be simple enough to ensure its security properties; for example, it should be impossible to infect it with malware.
6. Design and implement the internal servers as separate processes or shared libraries. Add security checks in each server using the PROTECTED ENTRY POINTS pattern.
7. Implement the external servers. Add security checks in each service provided by the servers using AUTHORIZATION and AUTHENTICATOR.

## 63.9. Example Resolved

By implementing our system using a microkernel, we can have several versions of each service, each with different degrees of security and reliability. We can replace servers dynamically if needed. We can also control access to specific servers and ensure that they are called in the proper way.

## 63.10. Consequences

The MICROKERNEL OPERATING SYSTEM ARCHITECTURE pattern offers the following benefits:

- Flexibility and extensibility: if you need an additional function or an existing function with different security requirements, you only need to add an external server. Extending the system capabilities or requirements also only requires addition or extension of internal servers.
- The microkernel mediates all calls for services and can apply authorization checks. In fact, the microkernel is in effect a concrete realization of a reference monitor.
- The well-defined interfaces between servers allow each server to check every request for their services.
- It is possible to add even more security by putting fundamental functions in internal servers.
- Servers usually run in user mode, which further increases security.
- The microkernel is very small and can be verified or checked for security.

The pattern also has the following potential liability:

- Communication overhead since all messages must go through the Microkernel.

## 63.11. Known Uses

- The PalmOS Cobalt (Figure 133) operating system has a preemptive multitasking kernel that provides basic task management. Many applications in PalmOS do not use the microkernel services; they are handled automatically by the system. The microkernel functionality is provided for internal use by system software or for certain specialpurpose applications [1].

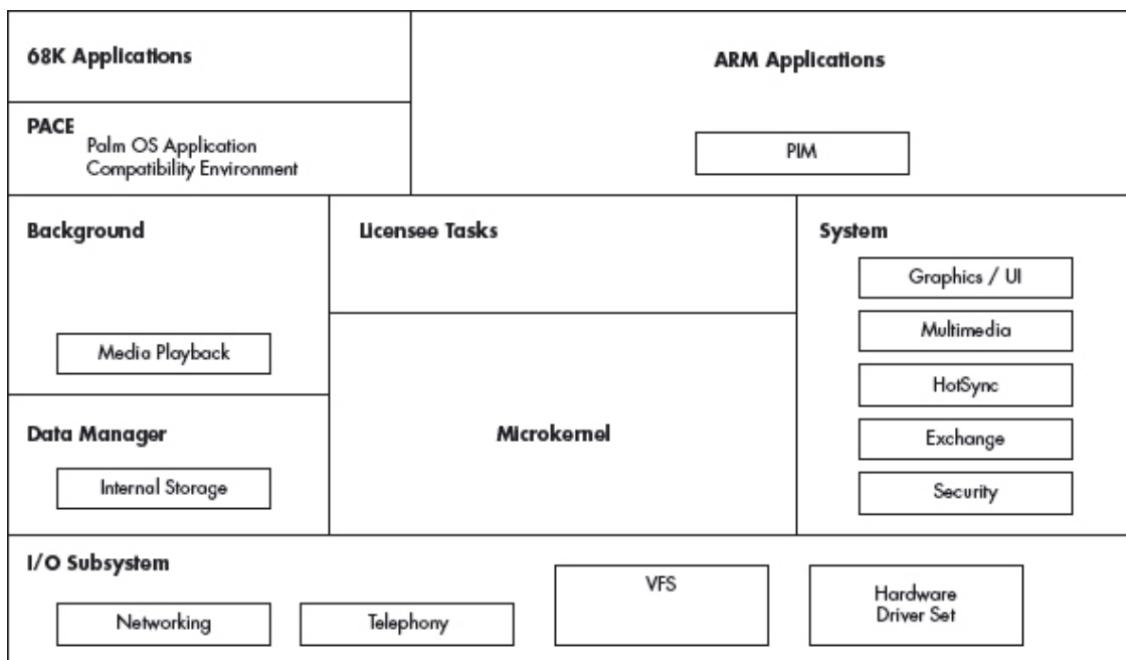


Figure 133: PalmOS Microkernel combined with layered OS architecture [1]

- The QNX Microkernel (Figure 134) is intended mostly for communication and process scheduling in real-time systems [2]. It will be used in the new RIM systems, adopting a layered architecture [3].

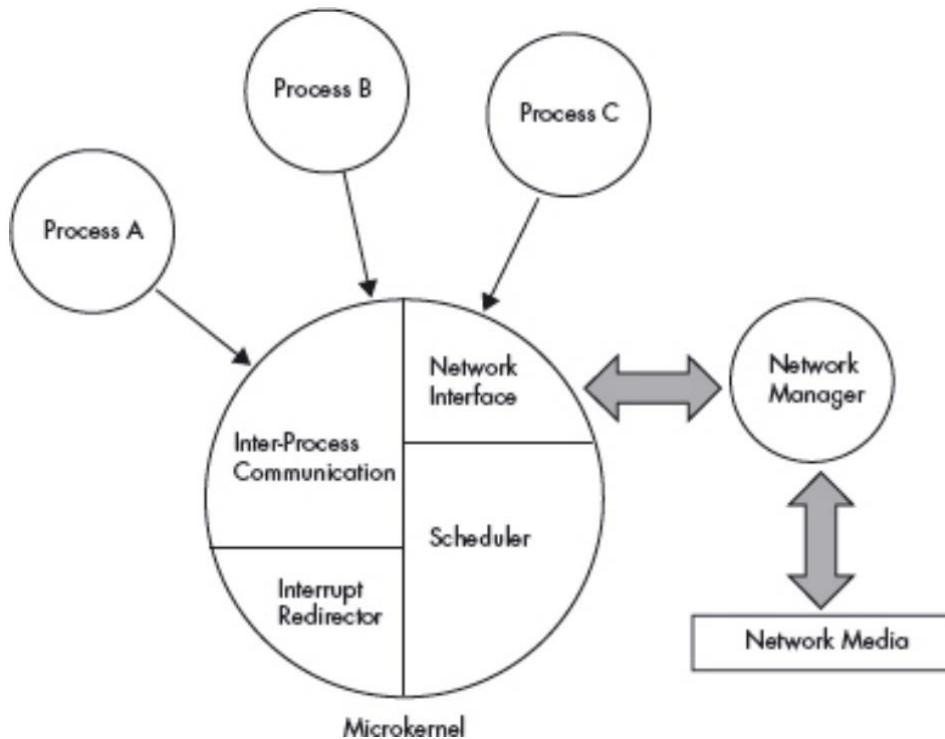


Figure 134: QNX microkernel architecture [2]

- Mach and Windows NT also use some form of microkernels [4].

### 63.12. Variants

Layered Microkernel. The MICROKERNEL OPERATING SYSTEM ARCHITECTURE pattern can be combined with the LAYERED OPERATING SYSTEM ARCHITECTURE pattern. In this case, servers can be assigned to levels and a call is accepted only if it comes from a level above the server level.

### 63.13. See Also

This pattern is a specialization of the Microkernel pattern [5]. As indicated, the microkernel itself can be considered as a concrete version of the REFERENCE MONITOR pattern.

### 63.14. References

- [1] Palmos. (n.d.). <http://www.palmos.com/dev/tech/overview.html>
- [2] QNX Software Systems. (n.d.). <http://www.qnx.com>
- [3] Wikipedia contributors. (n.d.). QNX. Wikipedia. <https://en.wikipedia.org/wiki/QNX>
- [4] Silberschatz, A., Galvin, P., & Gagne, G. (2008). Operating System Concepts. 8th edition. John Wiley & Sons Inc.

[5] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-Oriented System Architecture: a System of Patterns. Volume 1.

### 63.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 64. Modular Operating System Architecture

## 64.1. Intent

The MODULAR OPERATING SYSTEM ARCHITECTURE pattern describes how to separate operating system services into modules, each representing a basic function or component. The basic core kernel only has the required components to start itself and the ability to load modules. The core is the one module always in memory. Whenever the services of any additional modules are required, the module loader loads the appropriate module. Each module performs a function and may take parameters.

## 64.2. Example

Our group is building a new operating system that should support various types of devices requiring dynamic services with a wide variety of security requirements. We want to dynamically add operating system components, functions, and services, as well as tailor their security aspects according to the type of application. For example, a media player may require support to prevent copying of its contents, or a module for which a vulnerability alert has been issued could be removed.

## 64.3. Context

A variety of applications with diverse requirements that need to execute together, sharing hardware resources.

## 64.4. Problem

We need to be able to add or remove functions easily so that we can accommodate applications with a number of security requirements. How can we structure the operating system functions for this purpose?

The solution to this problem must resolve the following forces:

- Operating systems for PCs and other types of uses require a large variety of plugins. New plugins appear frequently, and we need the ability to add and remove them without disrupting normal operation.
- Some of the plugins may contain malware; we need to isolate their execution, so they do not affect other processes.
- We would like to hide security-critical modules from other modules to avoid possible attacks.
- Modules can call each other, which is a possible source of attacks.

## 64.5. Solution

Define a core module that can load, and link modules dynamically as needed.

## 64.6. Structure

Figure 135 shows a class diagram for this pattern. The KernelCore is the core of the modular operating system. A set of LoadableModules is associated with the KernelCore, indicating the modules that can be loaded according to their applications and the functions required. Any LoadableModule can call any other LoadableModule.

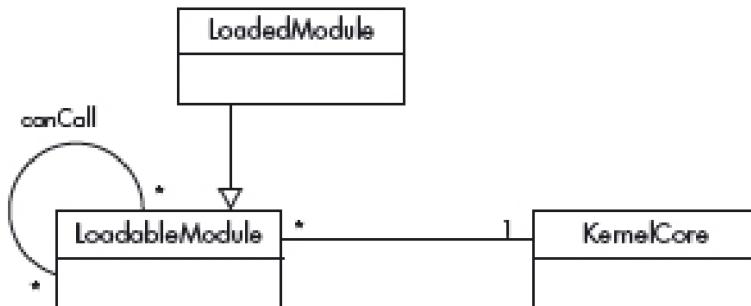


Figure 135: Class diagram for the MODULAR OPERATING SYSTEM ARCHITECTURE pattern

## 64.7. Dynamics

## 64.8. Implementation

1. Separate the functions of the operating system into independent modules according to whether:
  - a. They are complete functional units.
  - b. They are critical with respect to security.
  - c. They should execute in their own process for security reasons, or their own thread for performance reasons.
  - d. They should be isolated during execution because they may contain malware.
2. Define a set of loadable modules. New modules can be added later, according to the needs of specific applications.
3. Define a communication structure for the resultant modules. Operations should have well-defined call signatures and all calls should be checked.
4. Define a preferred order for loading some basic modules. Modules that are critical for security should be loaded only when needed to reduce exposure to attacks.

## 64.9. Example Resolved

We structured the functions of our system following the MODULAR OPERATING SYSTEM ARCHITECTURE pattern. Because each module could have its own address space, we can isolate its execution. Because each module can be designed independently, they can have different security constraints in their structure. This structure gives us flexibility with a good degree of security.

## **64.10. Consequences**

The MODULAR OPERATING SYSTEM ARCHITECTURE pattern offers the following benefits:

- Flexibility to add and remove functions contributes to security, in that we can add new versions of modules with better security.
- Each module is separate and communicates with other modules over known interfaces. We can introduce controls in these interfaces.
- It is possible to partially hide critical modules by loading them only when needed and removing them after use.
- By giving each executing module its own address space, we can isolate the effects of a rogue module.

This pattern also has the following potential liabilities:

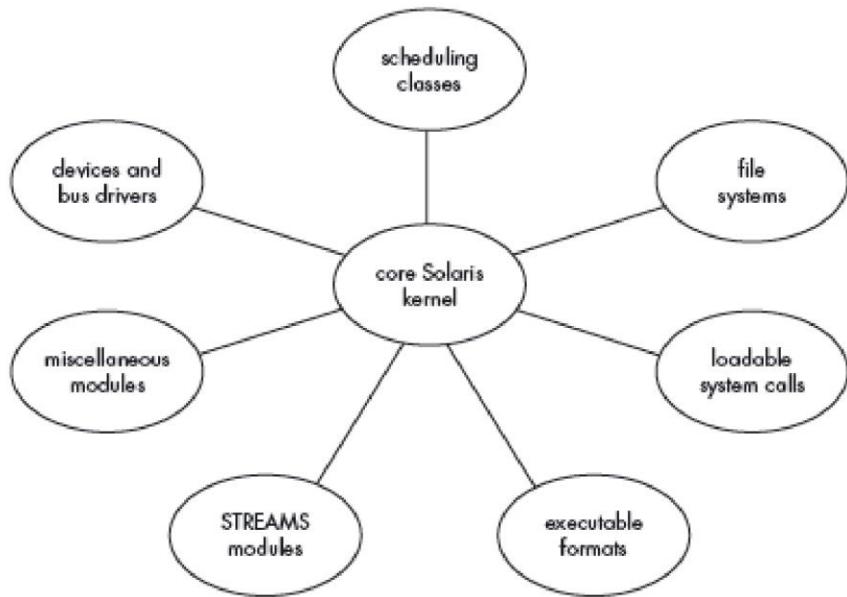
- Any module can ‘see’ all the others and potentially interfere with their execution.
- Uniformity of call interfaces between modules makes it difficult to apply stronger security restrictions to critical modules.

## **64.11. Variants**

Monolithic Kernel. The operating system is a collection of procedures. Each procedure has a well-defined interface in terms of parameters and results and each one is free to call any other [1]. There is no organization relating the operating system, components, services, and user applications: all the modules are at the same level. The difference between monolithic and modular operating system architectures is that in the monolithic approach, all the modules are loaded together at installation time, instead of on demand. This approach is not very attractive for secure systems.

## **64.12. Known Uses**

The Solaris 10 operating system (Figure 136) is designed following this pattern. Its kernel is dynamic and composed of a core system that is always resident in memory [2]. The various types of Solaris 10 loadable modules are shown in Figure 136 as loaded by the kernel core: the diagram does not represent the communication links between individual modules.



*Figure 136: The modular design of the Solaris 10 operating system*

- Extreme Ware from Extreme Networks [3].
- Some versions of Linux use a combination of modular and monolithic architectures.

### 64.13. See Also

The CONTROLLED EXECUTION DOMAIN pattern can be used to isolate executing modules.

### 64.14. References

- [1] Tanenbaum, A. (2009). *Modern operating systems*. Pearson Education, Inc.
- [2] Trusted Solaris Operating System. (n.d.).  
<http://www.sun.com/software/solaris/trustedsolaris/>
- [3] Extreme Networks. (n.d.). <http://www.extremenetworks.com/products/OS/>

### 64.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 65. Protected Entry Points

## 65.1. Intent

The PROTECTED ENTRY POINTS pattern describes how to force a call from one process to another to go through only prespecified entry points where the correctness of the call is checked, and other access restrictions may be applied.

## 65.2. Example

ChronOS is a company building a new operating system, including a variety of plug-in services such as media players, browsers, and others. In their design, processes can call each other in unrestricted ways. This makes process calls fast, which results in generally good performance, and everybody is satisfied. However, when they test the system, an error anywhere produces problems, because it propagates to other processes, corrupting their execution. Also, many security attacks are shown to be possible. It is clear that when their systems are in use, they will acquire a bad reputation and ChronOS will have problems selling it. They need to have a system that provides resilient service in the presence of errors, and which is resistant to attacks.

## 65.3. Context

Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm, and usually share data and other resources. The environment can be centralized or distributed. Some processes may be malicious or contain errors.

## 65.4. Problem

Process communication has an effect on security, because if a process calls another using entry points without appropriate checks, the calling process may read or modify data illegally, alter the code of the executing process, or take over its privilege level. If the checks are applied at specific entry points, some languages, such as C or C++, let the user manipulate pointers to bypass those entry points. Process communication also has a major effect on reliability because an error in a process may propagate to others and disrupt their execution.

The solution to this problem must resolve the following forces:

- Executing processes need to call each other to perform their functions. For example, in operating systems user processes need to call kernel processes to perform I/O, communications and other system functions. In all environments, process may collaborate to solve a common problem, and this collaboration requires communication. All this means that we cannot use process isolation to solve this problem.
- A call must go to a specified entry point or checks could be bypassed. Some languages let users alter entry point addresses, allowing input checks to be bypassed.

- A process typically provides services to other processes, but not all services are available to all processes. A call to a service not authorized to a process can be a security threat or allow error propagation.
- In a computing environment we have a variety of processes with different levels of trust. Some are processes that we normally trust, such as kernel processes; others may include operating system utilities, user processes and processes of uncertain origin. Some of these processes may have errors or be malicious. All calls need to be checked.
- The number, type, and size of the passed parameters in a call can be used to attack a process, for example by producing a buffer overflow. Incorrect parameters may produce or propagate an error

## 65.5. Solution

Systems that use explicit message passing have the possibility of checking each message to see if it complies with system policies. For example, one security feature that can be applied when calling another process is protected entry points. A process calling another process can only enter the called process at predefined entry points, and only if the signature used is correct (name, number of parameters, type, and size of parameters). This prevents bypassing entry checks and avoids attacks such as buffer overflows.

## 65.6. Structure

Figure 137 shows the class diagram of the solution. CallingProcess and CalledProcess are roles of processes in general. When a CallingProcess makes a request for a service to another process, the request is handled by an EntryPoint. This EntryPoint has a name and a list of parameters with predefined numbers, types, and size limits that can be used to check the correctness of the call signature. It can optionally add access control checks by using a Reference Monitor pattern or other input data tests.

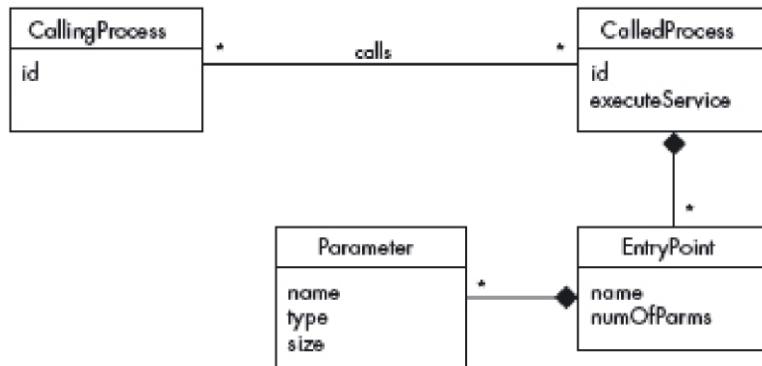


Figure 137: Class diagram for the PROTECTED ENTRY POINTS pattern

## 65.7. Dynamics

Figure 138 shows a CallingProcess performing a service call. The call must use a proper signature: that is, if the name of the service (opName) or the names of the parameters are incorrect, and the type or length of the parameters is not correct, it is rejected (this is checked by operation checkParmList).

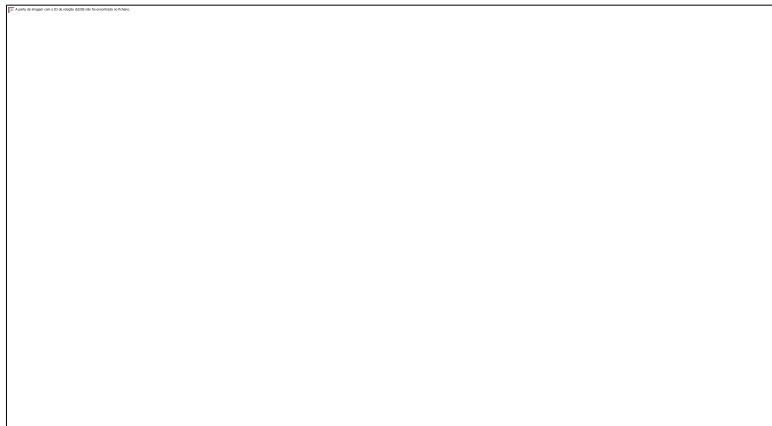


Figure 138: Sequence diagram for a process making a service call

## 65.8. Implementation

Kernels support calls as direct calls or through mailboxes. In the first case, the called process must check that the call is correct; in the second case, the mailbox must do the checking.

Entry points must be expressed as references as in Java, and not as pointers, as in C or C++ (as pointers allow arithmetic operations). In languages that use pointers, it is necessary to restrict their use in procedure calls, for example by disallowing pointer arithmetic.

## 65.9. Example Resolved

If the parameters of all calls are validated through protected entry points, many security and reliability problems can be avoided. Additional checks, such as access control and data value checks, can also be applied.

## 65.10. Consequences

The PROTECTED ENTRY POINTS pattern offers the following benefits:

- If we can check all the calls of one process to another, we can check that the calls are for appropriate services and apply checks for security or reliability purposes.
- Checking the number, type and length of the parameters passed in a call can prevent a variety of attacks and stop the propagation of some errors.
- If we know the level of trust of processes, we can adjust the number of checks; for example, we can apply more checks to suspicious processes.

## 65.11. Known Uses

- Multics.
- Systems that use ring architectures, for example the Intel Series 86 and Pentium.
- Systems that use capabilities, such as IBM S/6000.
- A specific use can be found in a patent for PC BIOS [1].

## **65.12. See Also**

- This pattern can be seen as a specific realization of the abstract principle validate input parameters.
- The protection rings pattern.
- Multilevel Secure Partitions pattern.
- The capability pattern.
- Access control and distributed access control. These checks can be applied in specific entry points to control access to resources.

## **65.13. References**

[1] Dayan, R. A., Geisler, D. R., Kinnear, S. G., Macon Jr, J. F., & Schwartz, W. H. (1991). *U.S. Patent No. 5,063,496*. Washington, DC: U.S. Patent and Trademark Office.

## **65.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 66. Protection Rings

## 66.1. Intent

The PROTECTION RINGS pattern allows control of how processes call other processes and how they access data. Crossing of rings is done through gates that check the rights of the crossing process. A process calling another process or accessing data in a higher ring must go through a gate.

## 66.2. Example

The ChronOS designers found that for applications that use programs with a variety of origins, there is a high overhead in applying elaborate checks to all of them. It would be more efficient to apply the checks selectively, depending on how much they trust the programs making the calls, but this is not usually known at execution time. If they could find a way to classify processes according to trust, they could improve the application of checks. It is not enough to rely on program features to enforce entering the right entry points, because applications may come in a variety of languages, some of which may allow skipping entry points.

## 66.3. Context

Executing processes in a computing system. Processes need to call other processes to ask for services or to collaborate in the computation of an algorithm, and usually share data and other resources. Some processes may be malicious or contain errors that may affect process execution. This pattern applies only to centralized environments, as opposed to distributed systems.

## 66.4. Problem

Defining a set of protected entry points is not enough if we cannot enforce their use. How can we prevent a process from calling another on an entry point that has no checks? We cannot rely on language features unless we only use a restricted set of languages, which is not practical in general. If all processes are alike, we also need to apply the same checks to all of them, which may be overkill.

The solution to this problem must resolve the following forces:

- We want to be able to enforce the application of protected entry points, at least for some processes. In this way, requests from suspicious processes can always be controlled.
- We would like to separate processes according to their level of trust and check only calls from a low-level to a higher-level process. This can reduce execution-time overhead considerably.
- In each higher level we can check signature validity, as well as control access or apply reliability tests. These actions should result in a more secure execution environment.

## 66.5. Solution

Define a set of hierarchical protection domains, called protection rings (typically 4 to 32) with different levels of trust. Assign processes to rings based on their level of trust. Ring crossing is performed through gates that enforce protected entry points: a process calling a higher-level process or accessing data at a higher level can only call this process or access data at predefined entry points with controlled parameters. Additional checks for security or reliability can be applied at the entry points.

## 66.6. Structure

Figure 139 shows a class diagram for this pattern. The CallingProcess requests services from a CalledProcess. To do so, it must enter a CallGate, which applies protected entry points that check the correct use of signatures. CallRules define the requirements for inter-level calls. The CallingProcess can access Data according to a set of DataAccessRules.

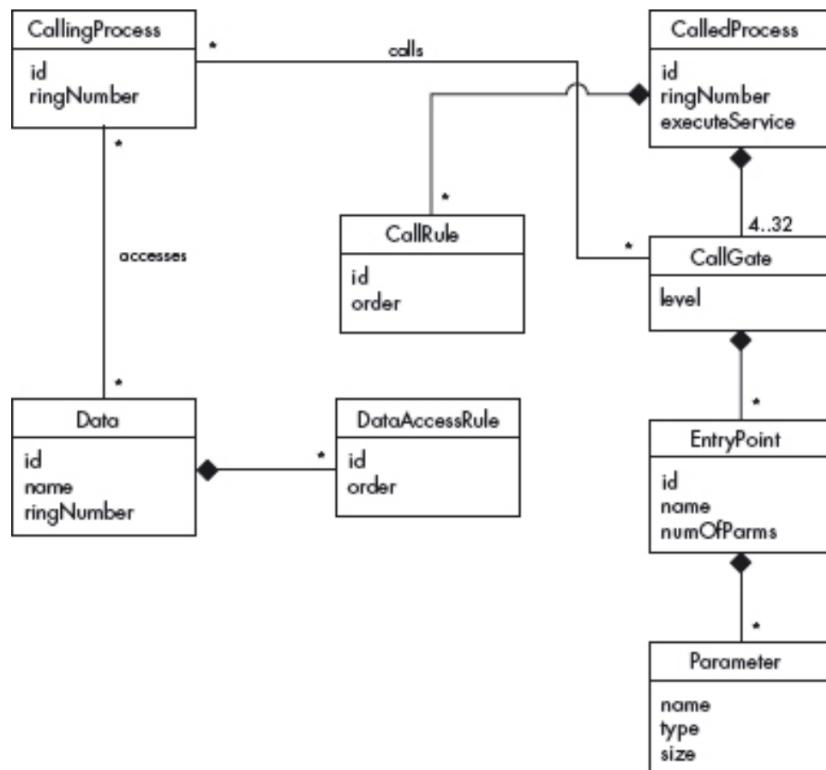


Figure 139: Class diagram for the PROTECTION RINGS pattern

## 66.7. Dynamics

Figure 140 shows a sequence diagram for a call to a higher-privilege ring. If the call fails an exception may be raised.

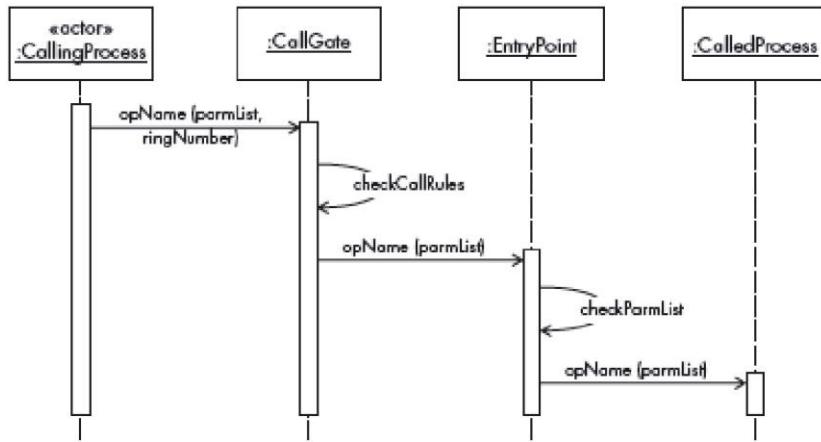


Figure 140: Sequence diagram for a successful call to a higher-privilege ring

## 66.8. Implementation

The call rules and the data access rules are usually implemented in the call instruction microcode [1]. Figure 141 shows a typical use of rings. Processes are assigned to rings based on their level of trust; for example, we could assign four rings in decreasing order of privilege and trust, to supervisor, utilities, trusted user programs, untrusted user programs.

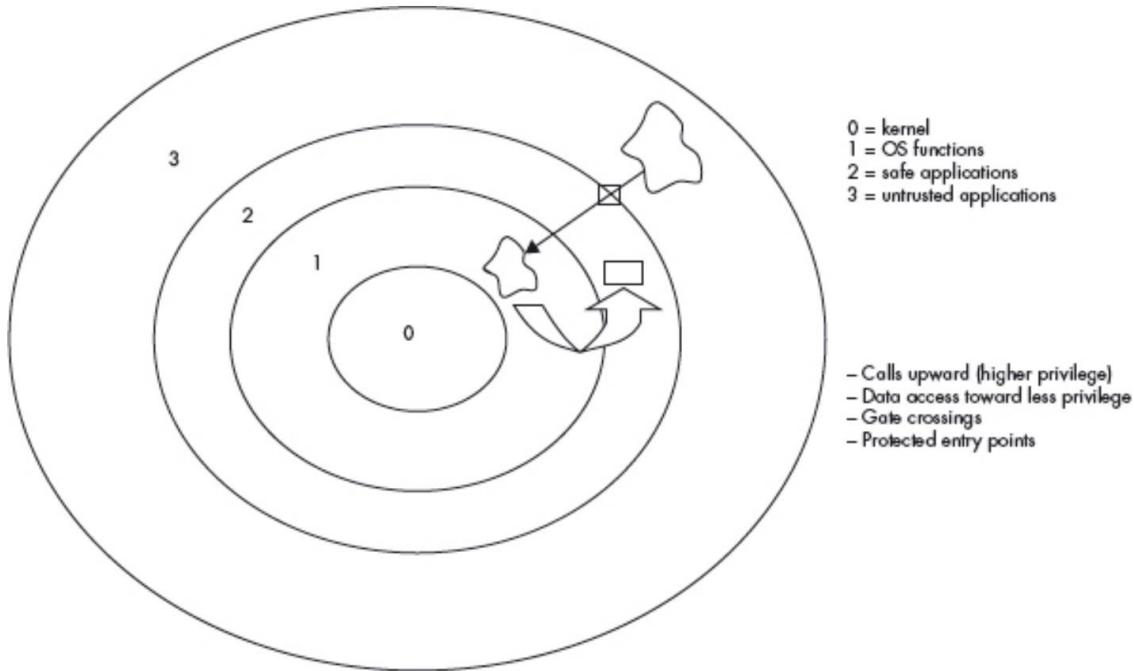


Figure 141: Assignment of protection rings

The program status word of the process indicates its current ring, and data descriptors also indicate their assigned rings. The values of the calling and called processes are compared to apply the transfer rules.

The Intel X86 architecture [1] applies two rules:

- Calls are allowed only in a more privileged direction, with possible restriction of a minimum calling level.

- Data at level p can be accessed only by a program executing at a more privileged level ( $\leq p$ ).

Another possibility for improving security is to allow calls only within a range of rings: in other words, jumping many rings is considered suspicious. Multics defined a call bracket, where calls are allowed only within rings in the bracket. More precisely, for a call from procedure i to a procedure with bracket (n1, n2, n3) the following rules apply:

- If  $n_2 < i \leq n_3$ , the call is allowed to specific entry points.
- If  $i < n_3$ , the call is not allowed.
- If  $i < n_1$ , any entry point is valid.

This extension only makes sense for systems that have many rings.

## 66.9. Example Resolved

Now we can preassign processes to levels according to their trust. All calls to processes of higher privilege are checked. Processes of low trust get more checks.

## 66.10. Consequences

The PROTECTION RINGS pattern offers the following benefits:

- We can separate processes according to their level of trust.
- Level transfers happen only through gates where we can apply the PROTECTED ENTRY POINTS pattern; that is, we have enforced protected entry points for upward calls.
- We can control procedure calls as well as data access across levels.

The pattern also has some potential liabilities:

- Crossing rings take time. Because of this delay, some operating systems use fewer rings. For example, Windows uses two rings, IBM's OS/2 uses three rings [2]. Using fewer rings improves performance at the expense of security.
- Without hardware support the crossing ring overhead is unacceptable, which means that this approach is only practical for operating systems and for centralized environments.

## 66.11. Variants

- Rings don't need to be strictly hierarchical; partial orders are possible and convenient for some applications. For example, a system that includes a secure database could assign a level to the database equal to but separated from system utilities; the highest level is for the kernel, and the lowest level is for user programs. This was done in a design involving an IBM 370 [3].
- In some systems, such as the MV8000, rings are associated with memory locations.
- Multics used the concept of the call bracket, where a call can be made within a range of rings.

## 66.12. Known Uses

- Multics introduced this concept and used 32 rings, as well as call brackets [4].
- The Intel Series X86 and Pentium [1].
- MV8000 [5,6].
- Hitachi HITAC.
- ICL 2900, VAX 11 and MARA, described in [7], which also describes Multics and the Intel series.
- [8] shows a use of rings to protect against malicious mobile code.
- An IBM S/370 was modified to have non-hierarchical rings [3].
- Rings have been used for fault-tolerant applications [9].

## 66.13. See Also

- A combination (process, domain) corresponds to a row of the Access Matrix pattern.
- Multilevel Secure Partitions pattern is an alternative for distributed environments, in which processes are assigned levels based on multilevel security models.
- PROTECTED ENTRY POINTS.

## 66.14. References

- [1] Intel Corporation. (n.d.). Intel Architecture Software Developer's Manual. Volume 3: System Programming-
- [2] Wikipedia contributors. (n.d.). Protection ring. Wikipedia.  
[https://en.wikipedia.org/wiki/Protection\\_ring#Supervisor\\_mode](https://en.wikipedia.org/wiki/Protection_ring#Supervisor_mode)
- [3] Fernandez, E. B., Summers, R. C., Lang, T., & Coleman, C. D. (1978). Architectural support for system protection and database security. *IEEE Transactions on Computers*, 27(08), 767-771.
- [4] Graham, R. M. (1968). Protection in an information processing utility. *Communications of the ACM*, 11(5), 365-369.
- [5] (n.d.). MV8000 principles of operation.  
[http://cide17ca7e5bcaa1096.skydrive.live.com/self.aspx/P%bablico/01400648\\_MV8000\\_PrincOps\\_Apr80.pdf](http://cide17ca7e5bcaa1096.skydrive.live.com/self.aspx/P%bablico/01400648_MV8000_PrincOps_Apr80.pdf)
- [6] Wallach, S. (1981). 32-bit Minicomputer Achieves Full 16-bit compatibility.
- [7] Frosini, G., & Lazzerini, B. (1985). Ring-protection mechanisms: general properties and significant implementations. *IEE Proceedings E-Computers and Digital Techniques*, 132(4), 203-210.
- [8] Shinagawa, T., Kono, K., & Masuda, T. (2000). *Exploiting segmentation mechanism for protecting against malicious mobile code*. University of Tokyo. Department of Information Science.
- [9] Ozaki, B. M., Fernandez, E. B., & Gudes, E. (1988). Software fault tolerance in architectures with hierarchical protection levels. *IEEE Micro*, 8(4), 30-43.

## **66.15. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **67. Role-Based Access Control**

## **67.1. Intent**

The ROLE-BASED ACCESS CONTROL pattern describes how to assign rights based on the functions or tasks of users in an environment in which control of access to computing resources is required.

## **67.2. Example**

The hospital has many patients, doctors, nurses, and other personnel. The specific individuals also change frequently. Defining individual access rights has become a time-consuming activity, prone to errors.

## **67.3. Context**

Any environment in which we need to control access to computing resources and in which there is a large number of users and information types, or a large variety of resources.

## **67.4. Problem**

For convenient administration of authorization rights, we need to have a means of factoring out rights. Otherwise, the number of individual rights is just too large; granting rights to individual users would require storing many authorization rules, and it would be hard for administrators to keep track of these rules. It is also hard to associate semantic meanings to the rules. How can we reduce the number of rules and make their semantics clearer?

The solution to this problem must resolve the following forces:

- Complexity. We would like to make the work of the security administrator as simple as possible.
- Semantics. In most organizations people are assigned specific functions or tasks. Their rights should correspond to those tasks.
- Policy. We need to define rights according to organizational policies.
- Commonality. People performing the same tasks should have the same rights.
- Policy enforcement. We want to help the organization to define precise access rights for its members according to a need-to-know policy.
- Flexibility. People joining, leaving, and changing functions should not require complex rights manipulation.

## **67.5. Solution**

Most organizations have a variety of job functions that require different skills and responsibilities. Users should be assigned rights based on their job functions or their designated tasks. This corresponds to the application of the need-to-know principle, a fundamental security policy [1]. Job functions can be interpreted as roles that people play in

performing their duties. In particular, web-based systems have a variety of users: company employees, customers, partners, search engines and so on.

## 67.6. Structure

Figure 142 shows a class diagram for ROLE-BASED ACCESS CONTROL. The User and Role classes describe registered users and their predefined roles respectively. Users are assigned to roles; roles are given rights according to their functions. The association class Right defines the access types that a user within a role is authorized to apply to the protection object. The combination Role, ProtectionObject and Right is an instance of the AUTHORIZATION pattern.

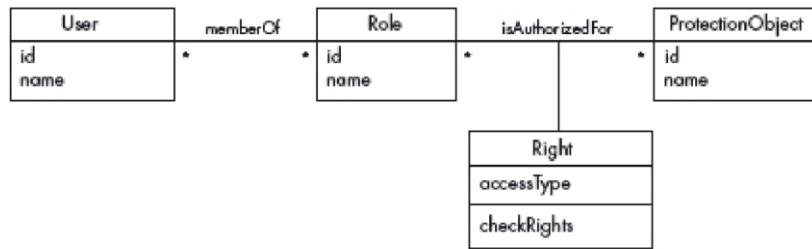


Figure 142: Class diagram for the ROLE-BASED ACCESS CONTROL pattern

## 67.7. Dynamics

Use cases include ‘Add a rule’, ‘Delete a rule’, ‘Modify a rule’, ‘Assign user to role’, ‘Assign rights to role’ and so on. We do not show their sequence diagrams here because they are very simple.

## 67.8. Implementation

Roles may correspond to job titles, for example ‘manager’, ‘secretary’. A finer-grained approach is to make them correspond to tasks. For example, a professor has the roles of ‘thesis advisor’, ‘teacher’, ‘committee member’, ‘researcher’ and so on. An approach to defining role rights is described in [2].

There are many possible ways to implement roles in a software system. [3] considers the implementation of the data structures needed to apply an RBAC model. Concrete implementations can be found in operating systems, database systems and web application servers.

## 67.9. Example Resolved

The hospital now assigns rights to the roles of doctors, nurses and so on. The number of authorization rules has decreased dramatically as a result.

## 67.10. Consequences

The ROLE-BASED ACCESS CONTROL pattern offers the following benefits:

- Semantics. We can make the rights given to a role correspond to tasks.

- Complexity. It allows administrators to reduce the complexity of security. Because there are many more users than roles, the number of roles becomes much smaller.
- Policy. Organization policies about job functions can be reflected directly in the definition of roles and the assignment of users to roles.
- Commonality. People performing the same tasks can be given the same rights.
- Flexibility. It is very simple to accommodate users joining, leaving, or being reassigned. All these use cases require only manipulation of the associations between users and roles.
- Structure. Roles can be structured into hierarchies for further flexibility and reduction of rules.
- Role separation. Users can activate more than one session at a time for functional flexibility: some tasks may require multiple views or different types of actions. Role separation is also important to avoid conflicts of interest: we can add UML constraints to indicate that some roles cannot be used in the same session or given to the same user (separation of duties).

The following potential liability may arise from applying this pattern:

- Some institutions may not have clearly defined roles in their organization, and some work must therefore be done to define these roles.

## 67.11. Variants

The model shown in Figure 143 additionally considers composite roles and objects: it is an application of the Composite pattern [4]. The figure also includes the concept of a session, which defines a context for the use of a role, may restrict the number of roles used together at execution time, and can be used to enforce role exclusion at execution time.

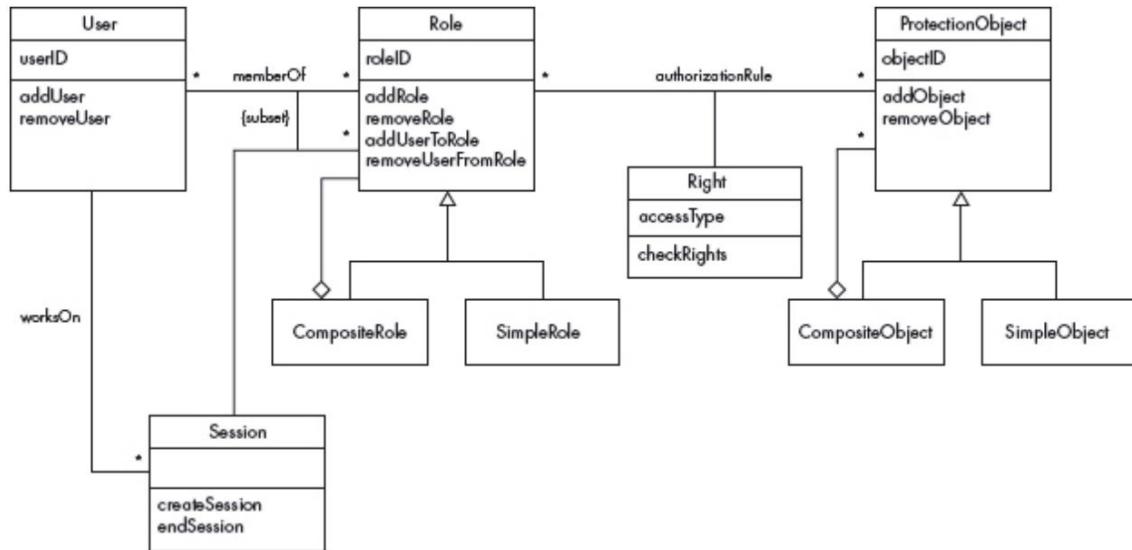


Figure 143: Class diagram for the Extended RBAC model

## 67.12. Known Uses

The RBAC pattern represents in object-oriented form a model described in terms of sets in [5]. That model has been the basis of most research papers and implementations of this concept.

RBAC is implemented in a variety of commercial systems, including Sun's J2EE, Microsoft's Windows 2000 and later, Microsoft's .NET [6], IBM's WebSphere, and Oracle, among others. The basic security facilities of Java's JDK 1.2 have been shown to be able to support a rich variety of RBAC policies. The NIST has developed a standard for RBAC [7]. We have used RBAC to describe access to physical structures [8].

### 67.13. See Also

The pattern whose class diagram is shown in Figure 143 includes AUTHORIZATION and Composite. A session object is used to provide execution context (see Variants).

### 67.14. References

- [1] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc..
- [2] Fullerton, M., & Fernandez, E. B. (2007, May). An analysis pattern for customer relationship management (CRM). In *Proc. of the 6th Latin American Conference on Pattern Languages of Programming* (pp. 80-90).
- [3] Kodituwakku, S. R., Bertok, P., & Zhao, L. (2001). APLRAC: A Pattern Language for Designing and Implementing Role-Based Access Control. In *EuroPLoP* (Vol. 1, p. 2001).
- [4] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [5] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2), 38-47.
- [6] Fenster, L. (2006). *Effective use of Microsoft Enterprise Library: building blocks for creating enterprise applications and services*. Pearson Education.
- [7] Ferraiolo, D. F., Sandhu, R., Gavrila, S., Kuhn, D. R., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3), 224-274.
- [8] Fernandez, E. B., Ballesteros, J., Desouza-Doucet, A. C., & Larrondo-Petrie, M. M. (2007, July). Security patterns for physical access control systems. In *IFIP Annual Conference on Data and Applications Security and Privacy* (pp. 259-274). Springer, Berlin, Heidelberg.

### 67.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 68. Virtual Address Space Structure Selection

## 68.1. Intent

The VIRTUAL ADDRESS SPACE STRUCTURE SELECTION pattern describes how to select the virtual address space for operating systems that have special security needs. Some systems emphasize isolation, others information sharing, others good performance. The organization of each process' virtual address space (VAS) is defined by the hardware architecture and has an effect on performance and security. The pattern enables all the hardware possibilities to be considered and selected according to need.

## 68.2. Example

We have a system running applications using images that require large graphic files. The application also has stringent security requirements because some of the images are sensitive and should be only accessed by authorized users. We need to decide on an appropriate VAS structure.

## 68.3. Context

Virtual memory allows the total size of the memory used by processes to exceed the size of physical memory. Upon use, the virtual address is translated by the address translation unit (usually the memory management unit (MMU) in microprocessors) to obtain a physical address that is used to access physical memory. To execute a process, the kernel creates a per-process virtual address space. We have a multiprogramming system with a variety of users and applications. Processes execute on behalf of users and at times must be able to share memory areas, at other times must be isolated, and in all cases, we need access control. Performance may also be an issue.

## 68.4. Problem

We need to select the virtual address space for processes depending on the majority of the applications we intend to execute, otherwise we can have mismatches that may result in poor security or performance.

The solution to this problem must resolve the following forces:

- Each process needs to be assigned a relatively large VAS to hold its data, stack, space for temporary variables, variables to keep the status of its devices, and other information.
- In multiprogramming environments processes have diverse requirements; some require isolation, others information sharing, others good performance.
- Data typing is useful to prevent errors and improve security. Several attacks occur by executing data and modifying code [1].
- Sharing between address spaces should be convenient, otherwise performance may suffer.

## 68.5. Solution

Select from four basic approaches that differ in their security features:

- One address space per process (Figure 144). The supervisor (kernel plus utilities) and each user process get their own address spaces. Using one VAS per process has the following trade-offs:
  - Good process isolation.
  - Some protection against possible illegal actions by a compromised operating system.
  - Simplicity.
  - Sharing is complex (special instructions to cross spaces are needed).

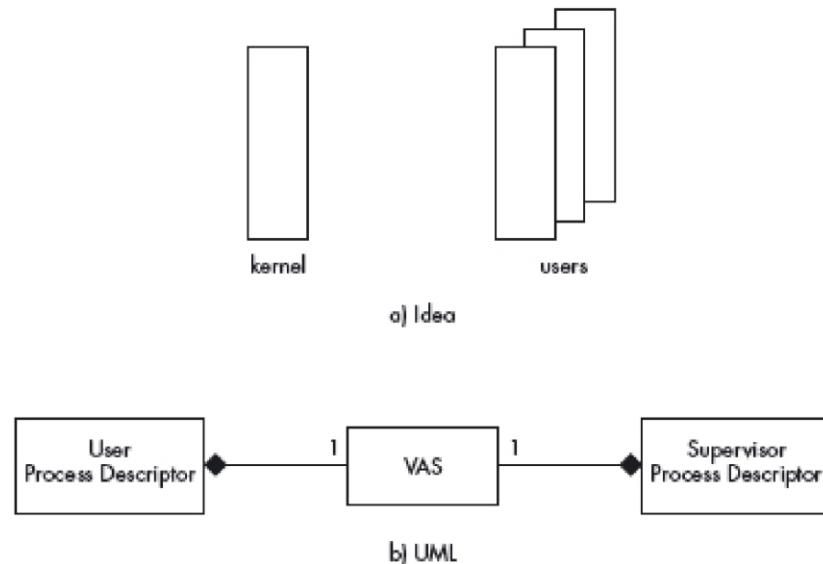
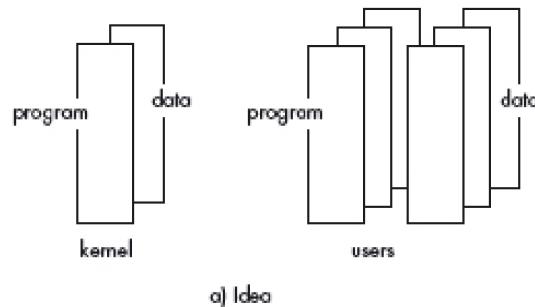
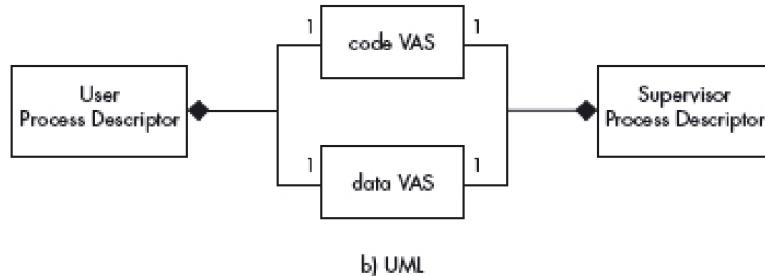


Figure 144: One address space per process

- Two address spaces per process (Figure 145). Each process gets a data and a code (program) virtual address space. Use of two VASs per process has the following trade-offs:
  - Good process isolation.
  - Some protection against possible illegal actions by a compromised operating system.
  - Data and instructions can be separated for better protection (some attacks take advantage of execution of data or modification of code). Data typing is also good for reliability.
  - A disadvantage is complex sharing, plus poor address space utilization.



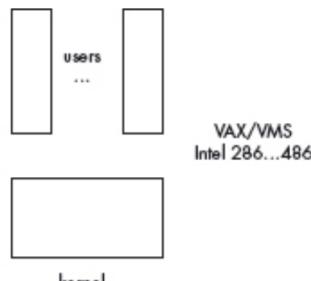
a) Idea



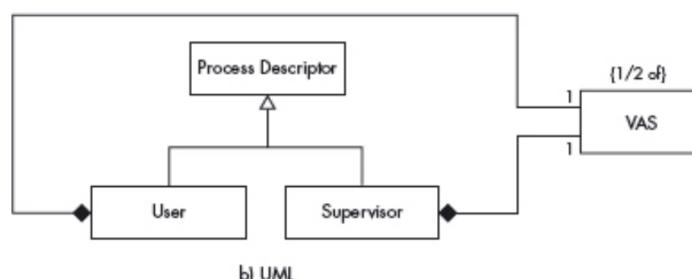
b) UML

Figure 145: Two address spaces per process

- One address space per user process, all of them shared with one address space for the operating system (Figure 146). The operating system (supervisor) can be shared between all processes. This scheme has the following trade-offs:
  - Good process isolation.
  - Good sharing of resources and services.
  - Suboptimal with respect to security (the supervisor has complete access to the user processes, and it must be trusted).
  - The address space available to each user process has been halved.



a) Idea



b) UML

Figure 146: One address space per user process, all of them shared with one address space for the operating system

- A single-level address space (Figure 147). Everything, including files, is mapped to one memory space. Use of a single-level address space has the following trade-offs:

- Good process isolation.
- Logical simplicity.
- Uniform protection (all I/O is mapped to memory).
- Offers the most elegant solution (only one mechanism to protect memory and files) and is potentially the most secure if capabilities are also used.
- Hard to implement in hardware due to the large address space required.

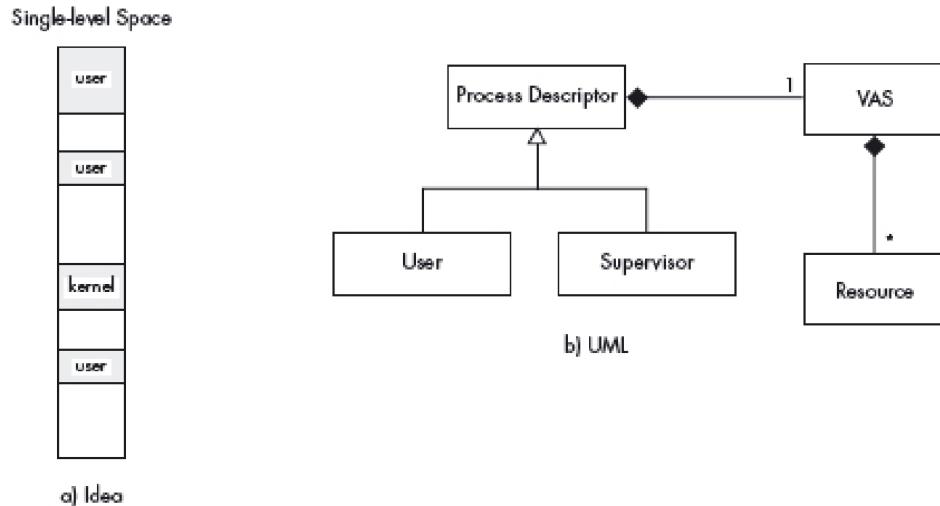


Figure 147: A single-level address space

## 68.6. Structure

## 68.7. Dynamics

## 68.8. Implementation

The VAS is implemented by the hardware architecture. The operating system designer can choose one of the architectures based on the requirements of the applications, according to the trade-offs discussed above. In a particular case, the choice may be influenced by company policies, cost, performance, and other factors, as well as security.

## 68.9. Example Resolved

## 68.10. Consequences

In addition to the specific benefits described as part of the solution (tradeoffs), the VIRTUAL ADDRESS SPACE STRUCTURE SELECTION pattern has the following general liabilities:

- Without hardware support it is not feasible to separate the virtual address spaces of the processes. Most processors use register pairs or descriptors that indicate the base (start) of a memory unit (segment) and its length or limit [2].
- If the mix of applications is not well-defined, it is hard to select the best solution. Considerations other than security then become more important.

## 68.11. Known Uses

- One address space per process. The NS32000, WE32100 and Clipper microprocessors [3]. Several versions of UNIX were implemented in these processors.
- Two address spaces per process. Used in the Motorola 68000 series. The Minix 2 operating system uses this approach [4].
- One address space per user process, all of them shared with one address space for the operating system. Used in the VAX series and in Intel processors. Windows runs in this type of address space.
- Single-level address space. Multics, IBM S/38, IBM S/6000 and HP PA-RISC use this approach. Multics had its own operating system. IBM AIX ran in S/6000 [5]. The PA-RISC architectures ran a version of UNIX.

## 68.12. See Also

- SECURE PROCESS/THREAD. The interaction between processes depends strongly on the virtual address space configuration, which can affect security, performance and sharing properties of the processes.
- VIRTUAL ADDRESS SPACE ACCESS CONTROL. A VAS is assigned to each process that can be accessed according to the rights of the process. The VIRTUAL ADDRESS SPACE STRUCTURE SELECTION pattern is applied first to select the appropriate structure. Once selected, the VAS is secured using the Controlled Virtual Address Space pattern.
- The Secure Storage pattern for rebooting..

## 68.13. References

- [1] Gollmann, D. (2010). Computer security. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(5), 544-554.
- [2] Silberschatz, A., Galvin, P., & Gagne, G. (2008). Operating System Concepts. 8th edition. John Wiley & Sons Inc.
- [3] Fernandez, E. B. (1985). Microprocessor architecture: The 32-bit generation. *VLSI Systems Design*, 34-44.
- [4] Tanenbaum, A. S., Herder, J. N., & Bos, H. (2006). Can we make operating systems reliable and secure?. *Computer*, 39(5), 44-51.
- [5] Camillone, N. A., Steves, D. H., & Witte, K. C. (1990). AIX operating system: A trustworthy computing system. *IBM RISC System/6000 Technology*, 168-172.

## **68.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 69. Access Control List

## 69.1. Intent

The ACCESS CONTROL LIST (ACL) pattern allows controlled access to objects by indicating which subjects can access an object and in what way. There is usually an ACL associated with each object.

## 69.2. Example

We are designing a system in which documents should be accessible only to specific registered users, who can either retrieve them for reading or submit modified versions. We need to verify that a specific user can access the document requested in a rapid manner.

## 69.3. Context

Centralized or distributed systems in which access to resources must be controlled. The systems comprise a policy decision point and policy enforcement points that enforce the access policy. A system has subjects that need to access resources to perform tasks. In the system, not every subject can access any object: access rights are defined and can be modeled as an access matrix, in which each row represents a subject, and each column represents an object. An entry in the matrix is indexed by a specific subject and a specific object and lists the types of actions that this subject can execute on this object.

## 69.4. Problem

In some systems the number of subjects and/or objects can be large. In this case, the direct implementation of an access matrix can use significant amounts of storage, and the time used for searching this large matrix can be significant.

In practice, the matrix is sparse. Subjects have rights on few objects and thus most of the entries are empty.

How can we implement the access matrix in a space- and time-efficient way?

The solution to this problem must resolve the following forces:

- The matrix may have many subjects and objects. Finding the rule that authorizes a specific request to an object may take a lot of time, as entries are unordered.
- The matrix can be very sparse: storing it as a matrix would require storing many empty entries, thus wasting space.
- Subjects and objects may be frequently added or removed. Making changes in a matrix representation is inefficient.
- The time spent for accessing a centralized access matrix may result in an additional overhead.
- A request received by a policy enforcement point indicates the requester identity, the requested object and the type of access requested. The requester identity, in particular, is controlled by the requester, and so may be forged by a malicious user.

## 69.5. Solution

Implement the access matrix by associating each object with an access control list (ACL) that specifies which actions are allowed on the object, by which authenticated users. Each entry in the list comprises a subject's identifier and a set of rights. Policy enforcement points enforce the access policy by requesting the policy decision point to search the object's ACL for the requesting subject identifier and access type. For the system to be secure, the subject's identity must be authenticated prior to its access to any objects. Since the ACLs may be distributed, like the objects they are associated with, several policy administration points may be responsible for creating and modifying the ACLs.

## 69.6. Structure

Figure 148 illustrates the solution. To be protected, an Object must have an associated ACL. This ACL is made up of ACLEntries, each of which contains a set of Rights permitted for a specific authenticated Subject. An authenticated Subject accesses an Object only if a corresponding Right exists in the Object's ACL. For security reasons, only the PDP can create and modify ACLs. At execution time, the PDP is responsible for searching an Object's ACL for a Right in order to make an access decision.

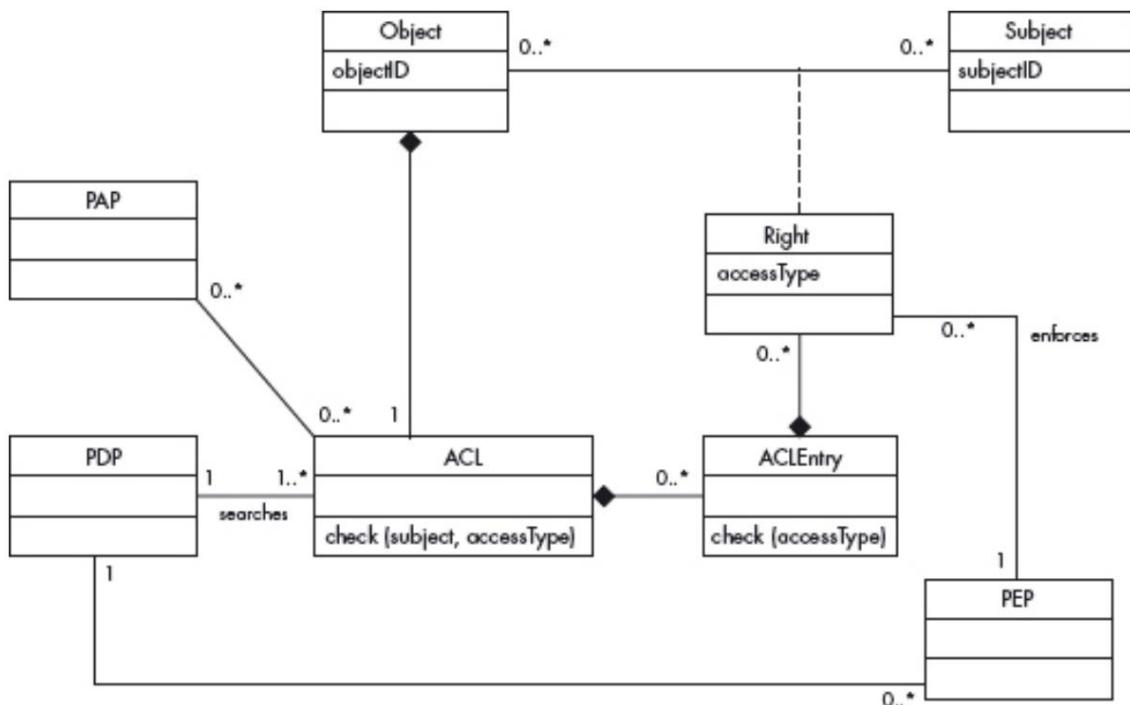


Figure 148: Class diagram for the ACCESS CONTROL LIST pattern

## 69.7. Dynamics

Figure 149 shows a sequence diagram describing the typical use case for 'Request object access'. The authenticated Subject's request for access to an Object is intercepted by a PEP, which forwards the request to the PDP. It can then check that the ACL corresponding to the Object contains an ACLEntry which corresponds to the Subject, and which holds the accessType requested by the Subject.

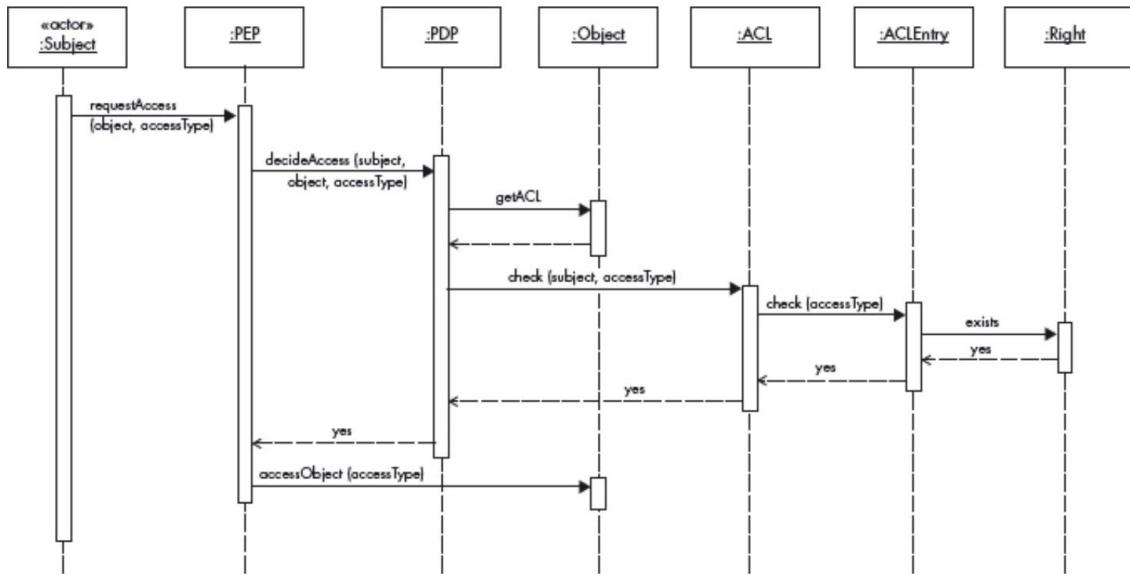


Figure 149: Sequence diagram for use case ‘Request object access’

## 69.8. Implementation

A decision must be made regarding the granularity of the ACLs. For example, it is possible to regroup users, such as the minimal access control lists in UNIX. It is also possible to have a finer-grained access control system. For example, the extended access control lists in UNIX allow specified access not only for the file’s owner and owner’s group, but also for additional users or groups.

The choice of access types can also contribute to a finer-grained access control system. For example, Windows defines over ten different permissions, whereas UNIX-like systems usually define three.

A creation/inheritance policy must also be defined: what should the ACL look like at the creation of an object? From what objects should it inherit its permissions?

ACLs are pieces of information of variable length. A strategy for storing ACLs must be chosen. For example, in the Solaris UFS file system, each inode has a field called `i_shadow`. If an inode has an ACL, this field points to a shadow inode. On the file system, shadow inodes are used like regular files. Each shadow inode stores an ACL in its data blocks. Linux and most other UNIX-like operating systems implement a more general mechanism called extended attributes (EAs). Extended attributes are name/value pairs associated permanently with file system objects, similar to the environment variables for a process [1].

## 69.9. Example Resolved

To enforce access control, we create a policy decision point and its corresponding policy enforcement points, which are responsible for intercepting and controlling accesses to the documents. For each document, we provide the policy decision point with a list of the users authorized to access the document and how (read or write). At access time, the policy decision point is able to search the list for the user. If the user is on the list with the proper access type, it can grant access to the document, otherwise it will refuse access.

In our distributed system, we make sure that only authenticated users — that is, users who provided a valid credential — can make requests.

## 69.10. Consequences

The ACCESS CONTROL LIST pattern offers the following benefits:

- Because all authorizations for a given object are kept together, we can go to the requested object and find out if a subject is there. This is much quicker than searching the whole matrix.
- The time spent accessing an ACL is less than the time that would have been spent accessing a centralized matrix.
- Access to unauthorized objects using forged requests on behalf of legitimate subjects is not possible, because we make sure that the requests are from only authenticated subjects.

The pattern also has the following potential liabilities:

- The administration of the subjects is rendered more difficult: the deletion of a subject may imply a scan of all ACLs, although this can be done automatically.
- When the environment is heterogeneous, it needs to be adapted to each type of PEPs. PDPs and PAPs must be implemented in a different way, adding an additional development cost.

## 69.11. Known Uses

- Operating systems such as Microsoft Windows (from NT/2000 on), Novell's NetWare, Digital's OpenVMS and UNIX-based systems use ACLs to control access to their resources.
- In Solaris 2.5, file ACLs allow a finer control over access to files and directories than the control that was possible with the standard UNIX file permissions. It is possible to specify specific users in an ACLEntry. It is also possible to modify ACLs for a file testfile by using the setfacl command in a similar way to the chmod command used for changing standard UNIX permissions:  
*setfacl -s u::rwx,g::-,o::-,m:rwx,u:user1:rwx,u:user2:rwx testfile*
- IBM Tivoli Access Manager for e-businesses uses ACLs to control access to the web and application resources [2].
- Cisco IOS, Cisco's network infrastructure software, provides basic traffic filtering capabilities with ACLs [3].

## 69.12. See Also

- The PEP and PDP come from the Policy-Based Access Control pattern. The CAPABILITY pattern is another way to implement the Access Matrix.
- Access Matrix [4] and role-based access control are models that can be implemented using ACLs.
- PEP is just a Reference Monitor.

- A variant exists oriented to centralized systems, Policy Enforcement pattern, which leverages ad hoc data structures to enhance efficiency.
- Acegi is a security framework for Java, used to build ACLs [5].

## 69.13. References

- [1] Grünbacher, A. (2003). {POSIX} Access Control Lists on Linux. In *2003 USENIX Annual Technical Conference (USENIX ATC 03)*.
- [2] IBM. (n.d.). Tivoli Federated Identity Manager.  
<http://www306.ibm.com/software/tivoli/products/federated-identity-mgr/>
- [3] Cisco. (n.d.). Cisco IOS Software. Retrieved June 26, 2007, from  
[http://www.cisco.com/en/US/products/sw/iosswrel/products\\_ios\\_cisco\\_ios\\_software\\_category\\_home.html](http://www.cisco.com/en/US/products/sw/iosswrel/products_ios_cisco_ios_software_category_home.html)
- [4] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc.
- [5] Siddiqui, B. (n.d.). Securing Java Applications with Acegi, Part 1: Architectural overview and Security Filters. Retrieved October 29, 2012, from  
<http://www.ibm.com/developerworks/java/library/j-acegi1/index.html>

## 69.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 70. Administrator Hierarchy

## 70.1. Intent

Many attacks come from the unlimited power of administrators. The ADMINISTRATOR HIERARCHY pattern allows the power of administrators to be limited, by defining a hierarchy of system administrators with rights controlled using a ROLE-BASED ACCESS CONTROL (RBAC) model and assigns them rights according to their functions.

## 70.2. Example

UNIX defines a superuser who has all possible rights. This is expedient: for example, when somebody forgets a password, but allows hackers to totally control the system through a variety of implementation flaws. Through gaining access to the administrator rights, an individual can create new administrator and user accounts, restrict their privileges and quotas, access their protected areas, or remove their accounts.

## 70.3. Context

An operating system with a variety of users, connected to the Internet. There are some commands and data that are used for system administration, and access to them needs to be protected. This control is usually applied through special interfaces. There are at least two roles required to properly manage privileges, Administrator and User.

## 70.4. Problem

Usually, the administrator has rights such as creating accounts and passwords, installing programs and so on. This creates a series of security problems. For example, a rogue administrator can perform all the usual functions, and even erase the log to hide their tracks. A hacker that takes over administrative power can do similar things. How can we curtail the excessive power of administrators to control rogue administrators or hackers?

The solution to this problem must resolve the following forces:

- Administrators need to use commands that permit management of the system, for example define passwords for files, define quotas for files, and create user accounts. We cannot eliminate these functions.
- Administrators need to be able to delegate some responsibilities and privileges to manage large domains. They also need the right to take back these delegations, otherwise the system is too rigid.
- Administrators should have no control of system logs, or no valid auditing would be possible.
- Administrators should have no access to the operational data in the users' applications. Or, if they do, their accesses should be logged.

## 70.5. Solution

Separate the different administrative rights into several hierarchical roles. The rights for these roles allow the administrators to perform their administrative functions and no more. Critical functions may require more than one administrative role to participate. Use the principle of separation of duty, where a user cannot perform critical functions unless in conjunction with other users.

## 70.6. Structure

Figure 150 shows a hierarchy for administration roles. This follows the Composite pattern [1]; that is, a role can be simple or composed of other roles, defining a tree hierarchy. The top-level Administrator can add or remove administrators of any type and initialize the system but should have no other functions. Administrators in the second level control different aspects, for example security, or use of resources. Administrators can further delegate their functions to lower-level administrators. Some functions may require two administrators to collaborate.

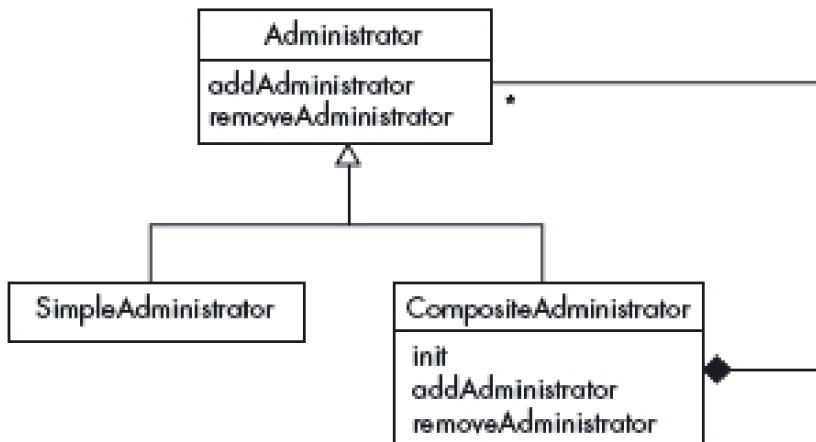


Figure 150: Class diagram for the ADMINISTRATOR HIERARCHY pattern

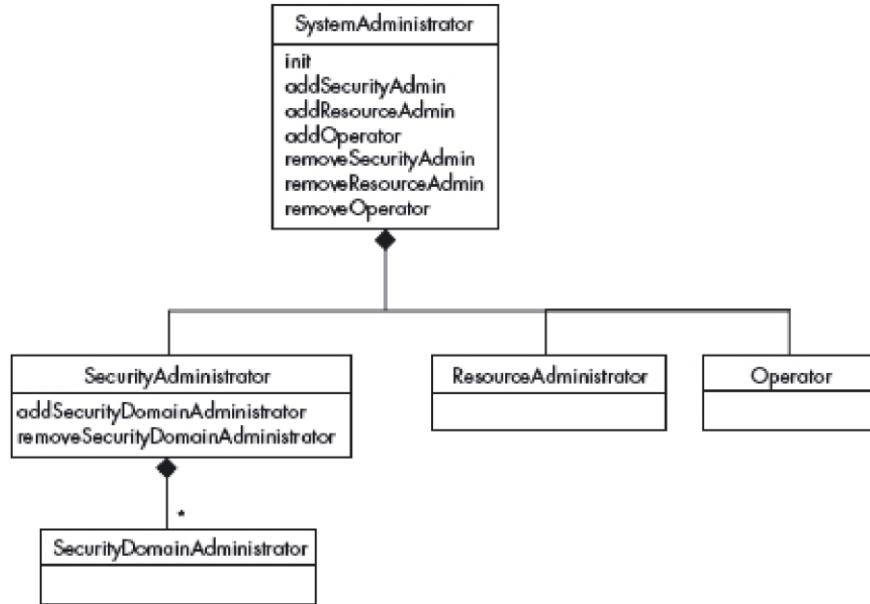
## 70.7. Dynamics

## 70.8. Implementation

1. Define a top-level administrative role with only the functions of setting up and initializing the system. This includes definition of administrative roles, assignment of rights to roles and assignment of users to roles.
2. Separate the main administrative functions of the system and define an administrative role for each one of them. These define the second level of the hierarchy.
3. Define further levels to accommodate administrative units in large systems, or for breaking down rights into functional sets.

Figure 151 shows a class diagram describing a typical administrator hierarchy. Here the SystemAdministrator starts the system and does not perform further actions. The second-level

administrators can perform set up and other functions; the SecurityAdministrator defines security rights. SecurityDomainAdministrators define security in their domains.



*Figure 151: A typical administration hierarchy*

## 70.9. Example Resolved

Now the superuser only starts the system. During normal operation the administrators have restricted powers. If a hacker takes over their functions, they can do only limited damage.

## 70.10. Consequences

The ADMINISTRATOR HIERARCHY pattern offers the following benefits:

- If an administrative role is compromised, the attacker gets only limited privileges, so the potential damage is limited.
- The reduced rights also reduce the possibility of misuse by the administrators.
- The hierarchical structure allows taking back control of a compromised administrative function.
- The advantages of the RBAC model apply simpler and fewer authorization rules, flexibility for changes, and so on.
- This structure is useful not only for operating systems, but also for servers, databases systems or any systems that require administration.

The pattern also has the following potential liabilities:

- Extra complexity for the administrative structure.
- Less expediency: performing some functions may involve more than one administrator.
- Many attacks are still possible: if someone misuses an administrative right, this pattern only limits the damage. Logging can help misuse detection.

## **70.11. Known Uses**

- AIX [2] reduces the privileges of the system administrator by defining five partially ordered roles: superuser, security administrator, auditor, resource administrator and operator.
- Windows NT uses four roles for administrative privileges: standard, administrator, guest, and operator. A user manager has procedures for managing user accounts, groups, and authorization rules.
- Trusted Solaris [3]. This operating system is an extension of Solaris 8, using the concept of trusted roles with limited powers.
- Argus Pitbull [4]. In this operating system, least privilege is applied to all processes, including the superuser. The superuser is implemented using three roles: systems security officer, system administrator and system operator.

## **70.12. See Also**

- This pattern applies the principles of least privilege and separation of duty, which some people consider also to be patterns. Each administrator role is given only the rights it needs to perform its duties and some functions may require collaboration.
- Administrative rights are usually organized according to a ROLE-BASED ACCESS CONTROL model.

## **70.13. References**

- [1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [2] Camillone, N. A., Steves, D. H., & Witte, K. C. (1990). AIX operating system: A trustworthy computing system. *IBM RISC System/6000 Technology*, 168-172.
- [3] (n.d.). Trusted Solaris Operating System.  
<http://www.sun.com/software/solaris/trustedsolaris/>
- [4] Argus Systems Group. (n.d.). Trusted OS security: Principles and Practice. Retrieved October 27, 2012, from <http://www.argus-systems.com/Products/products.html>

## **70.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 71. Capability

## 71.1. Intent

The CAPABILITY pattern allows controlled access to objects by providing a credential or ticket to a subject to allow it to access an object in a specific way. Capabilities are given to the principal.

## 71.2. Example

We are designing a system that allows registered users to read or modify confidential documents. We need to verify that a specific user can access a confidential document in an efficient and secure manner. In particular, we worry that if the parts of our system that deal with access control are too large and/or distributed, they may be compromised by attackers.

## 71.3. Context

Distributed systems in which access to resources must be controlled. The systems have a policy decision point and its corresponding policy enforcement points that enforce the access policy. A system is composed of subjects that need to access resources to perform their tasks. In the system, not every subject can access any object: access rights are defined and can be modeled as an access matrix, in which each row represents a subject, and each column represents an object. An entry of the matrix is indexed by a specific subject and a specific object and lists the types of actions that this subject can execute on this object. The system's implementation is vulnerable to threats from attackers that may compromise its components.

## 71.4. Problem

In some of these systems the number of subjects and/or objects can be large. In this case, the direct implementation of the access matrix can use significant amounts of storage, and the time to search a large matrix can be significant.

In practice, the matrix is sparse. Subjects have rights on few objects and thus most of the entries are empty. How can we implement the access matrix in a space- and time-efficient way?

The solution to this problem must resolve the following forces:

- The matrix may have many subjects and objects. Finding the rule that authorizes a specific request to an object may take a lot of time (unordered entries).
- The matrix can be very sparse and storing it as a matrix would require storing many empty entries, thus wasting space.
- Subjects and objects may be frequently added or removed. Making changes in a matrix representation is inefficient.
- The time spent for accessing a centralized access matrix may result in an additional overhead.

- A request received by a policy enforcement point indicates the requester identity, the requested object and the type of access requested. The requester identity, in particular, is controlled by the requester, and so may be forged by a malicious user.
- The size of the units that can create and/or modify the policies (such as policy administration points) has an impact on the security of the system. Minimizing their size will reduce their chance of being compromised by attackers.

## 71.5. Solution

Implement the access matrix by issuing a set of capabilities to each subject. A capability specifies that the subject possessing the capability has a right on a specific object. Policy enforcement points and the policy decision point of the system enforce the access policy by checking that the capability presented by the subject at the access time is authentic, and by searching the capability for the requested object and access type. Trust a minimum part of the system – create a unique capability issuer that is responsible for issuing the capabilities. The capabilities must be implemented in a way that allows the policy decision point to verify their authenticity, so that a malicious user cannot forge one.

## 71.6. Structure

Figure 152 illustrates the solution. In order to protect the Objects, a CapabilityProvider, the minimum trusted part of our system, issues a set of Capabilities to each Subject by using a secure channel. A Capability contains a set of Rights that the Subject can perform on a specific Object. A Subject accesses an Object only if a corresponding Right exists in one of the Subject's Capabilities. At execution time, the PDP is responsible for checking the Capability's authenticity and searching the Capability for both the requested Object and the requested accessType in order to make an access decision.

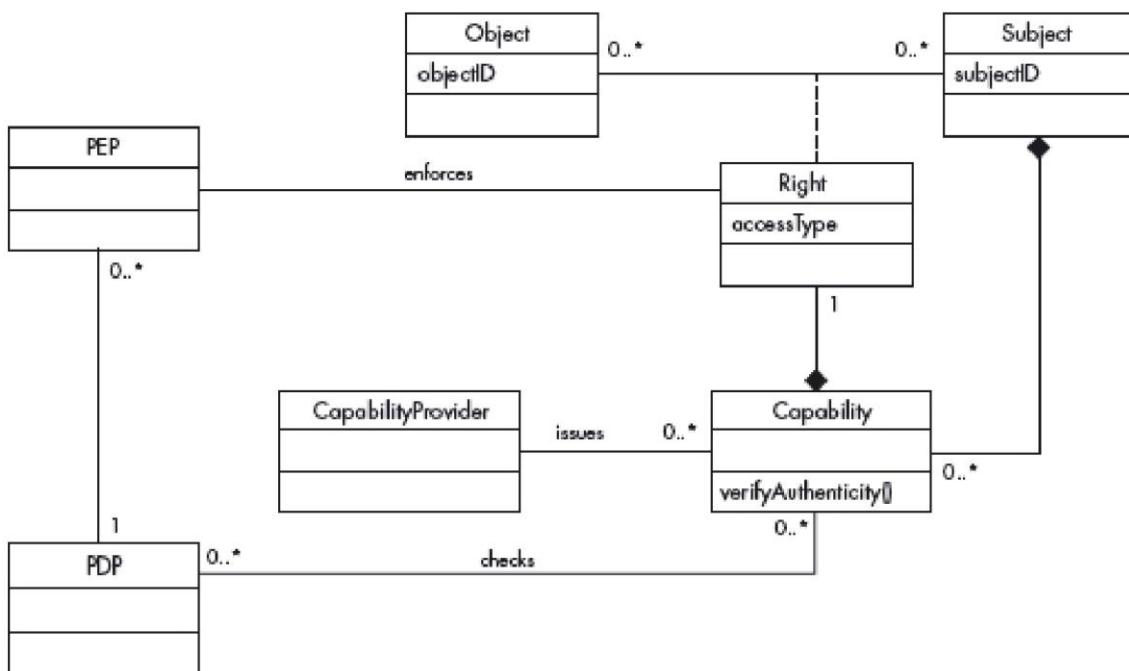


Figure 152: Class diagram for the CAPABILITY pattern

## 71.7. Dynamics

Figure 153 shows a sequence diagram describing the typical use case of ‘Request object access’. The Subject requests access to an Object by including a corresponding Capability. The request is intercepted by a PEP, which forwards the request to the PDP. It can then check that the Capability holds the accessType requested by the Subject.

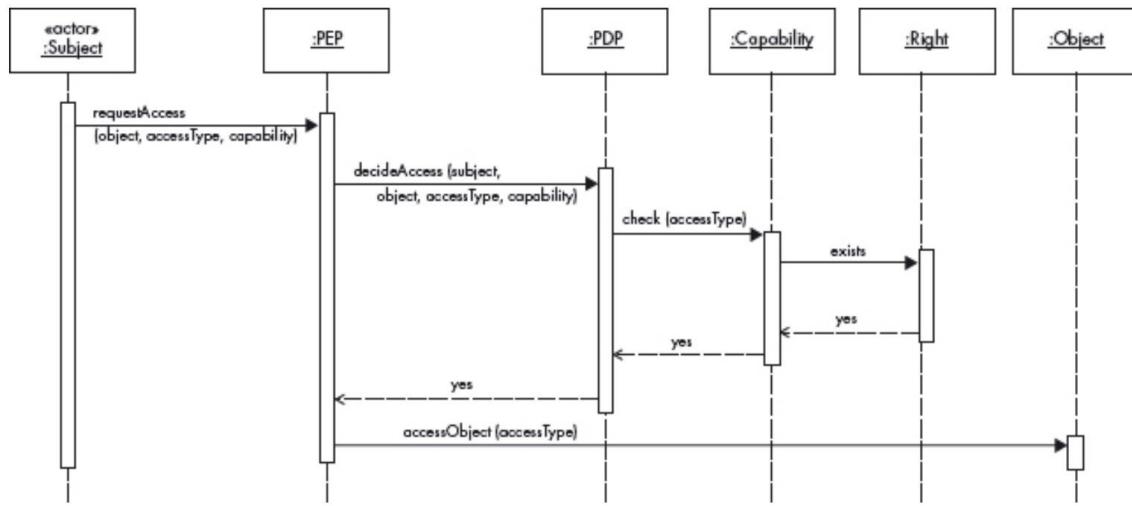


Figure 153: Sequence diagram for the use case ‘Request object access’

## 71.8. Implementation

Since a capability must be unforgeable and unmodifiable, it can be implemented as hardware or software:

- As hardware:
  - Tags. Tagging allows for the categorization of each word as data or a capability. Then no copying should be allowed from capability to data or vice versa, no arithmetic operation should be allowed on capabilities. A disadvantage of this method is the memory waste by using tags.
  - Segmentation. Whole segments of memory are used exclusively for capabilities or for data. No operation should be allowed between partitions of different types. A disadvantage of this is that many processes may need two segments.
- As software:
  - Cryptography. Usually used, the capabilities may be encrypted by the capability issuer’s key.

## 71.9. Example Resolved

To enforce access control, we create a policy decision point and its corresponding policy enforcement points that are responsible for intercepting and controlling accesses to confidential documents. When a user logs on to the system, a robust token issuer provides a set of tokens that indicate which confidential documents are authorized. Tokens are digitally signed so that they can’t be created or modified by users. At request time, a user wishing to access a confidential document presents its token to the policy enforcement point, and then to

the policy decision point, which grants them access to the document. If a user does not present a token corresponding to the document and the access mode, access is refused.

## 71.10. Consequences

The CAPABILITY pattern offers the following benefits:

- Because the capability is sent together with the request, the time spent for accessing an authorization is much less than the time that would have been spent searching a whole matrix or searching an access control list (ACL).
- The time spent accessing a capability at request time is less than the time that would have been spent accessing a centralized matrix.
- The part of the system that we need to trust is minimal. The capability provider is only responsible for issuing capabilities to the right users at an initial time.
- It is harder for malicious users to forge or modify capabilities, since a capability provides a way to verify its authenticity.

The pattern also has some potential liabilities:

- The administration of the objects is more difficult: The addition of an object implies the issuing of capabilities to every authorized user.
- When the environment is heterogeneous, the administration of the rights is more complex. There is no straightforward way to revoke a right, since users are in control of the capabilities they have acquired. A solution could be to add a validity time to each capability, or through indirection, or by using virtual addresses [1].
- The right is transferable: that is, a capability can be stolen and replayed by (or given to) a malicious user. (This is not the case in OSs in which accesses to the capabilities are also controlled by the TCB, but those need the support of special hardware.)

## 71.11. Known Uses

- Most of the capability-based systems are operating systems. Usually hardware assistance is needed; for example, capabilities are placed in special registers and manipulated with special instructions (Plessey P250), or they are stored in tagged areas of memory (IBM 6000).
- Many distributed capability-based systems have been researched and described [2,3,4,5]. Among those, Amoeba [5] is a distributed operating system in which multiple machines can be connected together. It has a microkernel architecture. All objects in the system are protected using a simple scheme. When an object (representing a resource) is created, the server doing the creation constructs a capability in the form of a 128-bit value and returns it to the caller. Subsequent operations on the object require the user to send its capability to the server to both specify the object and to prove the user has permission to manipulate the object. Capabilities are protected cryptographically to prevent tampering. The Symbian operating system uses capabilities [6].

## 71.12. See Also

- The PEP and PDP are from the ACCESS CONTROL LIST pattern. The ACCESS CONTROL LIST pattern is another way to implement the access matrix.
- Capabilities can be implemented into the VAS (virtual address space) using segmentation.
- The PEP is just a Reference Monitor.
- Access Matrix [7], RBAC are models that can be implemented using ACLs. Credential pattern is a type of capability.

## 71.13. References

- [1] Anderson, R. (2008). *Security Engineering*. 2nd edition. John Wiley & Sons, Inc.
- [2] Johnson, H. L., Koegel, J. F., & Koegel, R. M. (1985, October). A secure distributed capability based system. In *Proceedings of the 1985 ACM annual conference on The range of computing: mid-80's perspective: mid-80's perspective* (pp. 392-402).
- [3] Donnelley, J. E. (1976, August). A Distributed Capability Computing System (DCCS). In *ICCC* (pp. 432-440).
- [4] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. *Computer*, 29(2), 38-47.
- [5] (n.d.). Amoeba Operating System. Retrieved October 27, 2012, from [www.cs.vu.nl/pub/amoeba/](http://www.cs.vu.nl/pub/amoeba/)
- [6] Heath, C. (2006). *Symbian OS: platform security*. John Wiley & Sons.
- [7] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc.

## 71.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **72. Policy-Based Access Control**

## **72.1. Intent**

The POLICY-BASED ACCESS CONTROL pattern describes how to decide whether a subject is authorized to access an object according to policies defined in a central policy repository.

## **72.2. Example**

Consider a financial company that provides services to its clients. Their computer systems can be accessed by clients, who send orders to the company for buying or selling commodities (stocks, bonds, real estate, art and so on) via e-mail or through their website. Brokers employed by the company can carry out the clients' orders by sending requests to the systems of various financial markets, or by consulting information from financial news websites. A government auditor visits periodically to check for compliance with laws and regulations.

All of these activities are regulated by policies with various granularities within the company. For example, the billing department can have the rule 'Only registered clients whose account status is in good standing may send orders', the technical department can decide that 'E-mails with attachments bigger than x Mb won't be delivered', the company security policy can state that 'Only employees with a broker role can access the financial market's web services' and that 'Only the broker or custodian of a client can access its transaction information', whereas the legal department can issue the rule that 'Auditors can access all transaction information', and so on.

All of these policies are enforced by different components of the company's computer system (e-mail server, file system, web service access control component, and financial application). This approach has several problems: the policies may be described in different syntaxes, and it is difficult to have a global view of which policies apply to a specific case. Moreover, two policies can be conflicting, and there is no way to combine them in a clear way. In summary, this approach could be error prone and complex to manage.

## **72.3. Context**

Consider centralized or distributed systems with a large number of resources (objects). A large number of subjects may access those objects. Rules are defined to control access to objects. The rules defined by the organization are typically designed by different actors (technical, organizational, legal, and so on), and each set of rules designed by a specific policy designer can concern overlapping sets of objects and/or subjects.

## **72.4. Problem**

Enforcing these rules for a particular access request may be complex, and thus error-prone, because there is no clear view of which rules to apply to a request.

How can we enforce access control according to the predefined rules in a consistent way?

The solution to this problem must resolve the following forces:

- Objects may be frequently added or removed.
- The solution should be able to implement a wide variety of access control models, such as access matrix, RBAC.
- Malicious users can attempt unauthorized access to objects.
- There should be no direct access to objects: every request must be mediated.

## 72.5. Solution

Most access control systems are based on the AUTHORIZATION pattern, in which the access of a subject to an object depends only on the existence of a positive applicable rule. If no such rule exists, the access is denied.

In our case, the situation is more complicated: the existence of a positive applicable rule should not necessarily imply that access should be granted. All the rules must be considered, and a final decision must be made from the set of applicable rules and some meta-information about the way they should be combined. Part of that meta-information is located in a policy object. This policy object aggregates a set of rules and specifies how those rules must be combined. For more flexibility about the combination of rules, a composite object regroups the rules into policies and policy sets. Policy sets aggregate policies and include information about how to combine rules from different policies. To be able to select all applicable rules easily, they should be stored in a unique repository for the organization and administered in a centralized way.

At access time, all requests are intercepted by policy enforcement points (PEPs), a specific type of Reference Monitor. The repository is accessed by a unique policy decision point (PDP), which is responsible for computing the access decision by cooperating with a policy information point (PIP), which may provide information about the subject, or the resource accessed. The rules and policies are administered through a unique policy administration point (PAP).

Finally, because rules and policies are designed by different teams, possibly for the same objects and subjects, this scheme does not guarantee that a conflict between rules in different policy components would never occur. In that case, the PDP may have a dynamic policy conflict resolver to resolve the conflict, which would need to use meta-rules. A complementary static policy conflict resolver may be a part of the PAP and should detect conflicts between rules at the time they are entered into the repository.

## 72.6. Structure

Figure 154 illustrates the solution. A Subject's access requests to particular objects are intercepted by PEPs, which are a part of the security infrastructure that is responsible for enforcing the organization Policy about this access. PEPs query another part of the security infrastructure, the PDP, which is responsible for computing an access decision. To compute the decision, the PDP uses information from a PIP, and retrieves the applicable Policy from the unique PolicyRepository, which stores all of the PolicyRules for the organization.

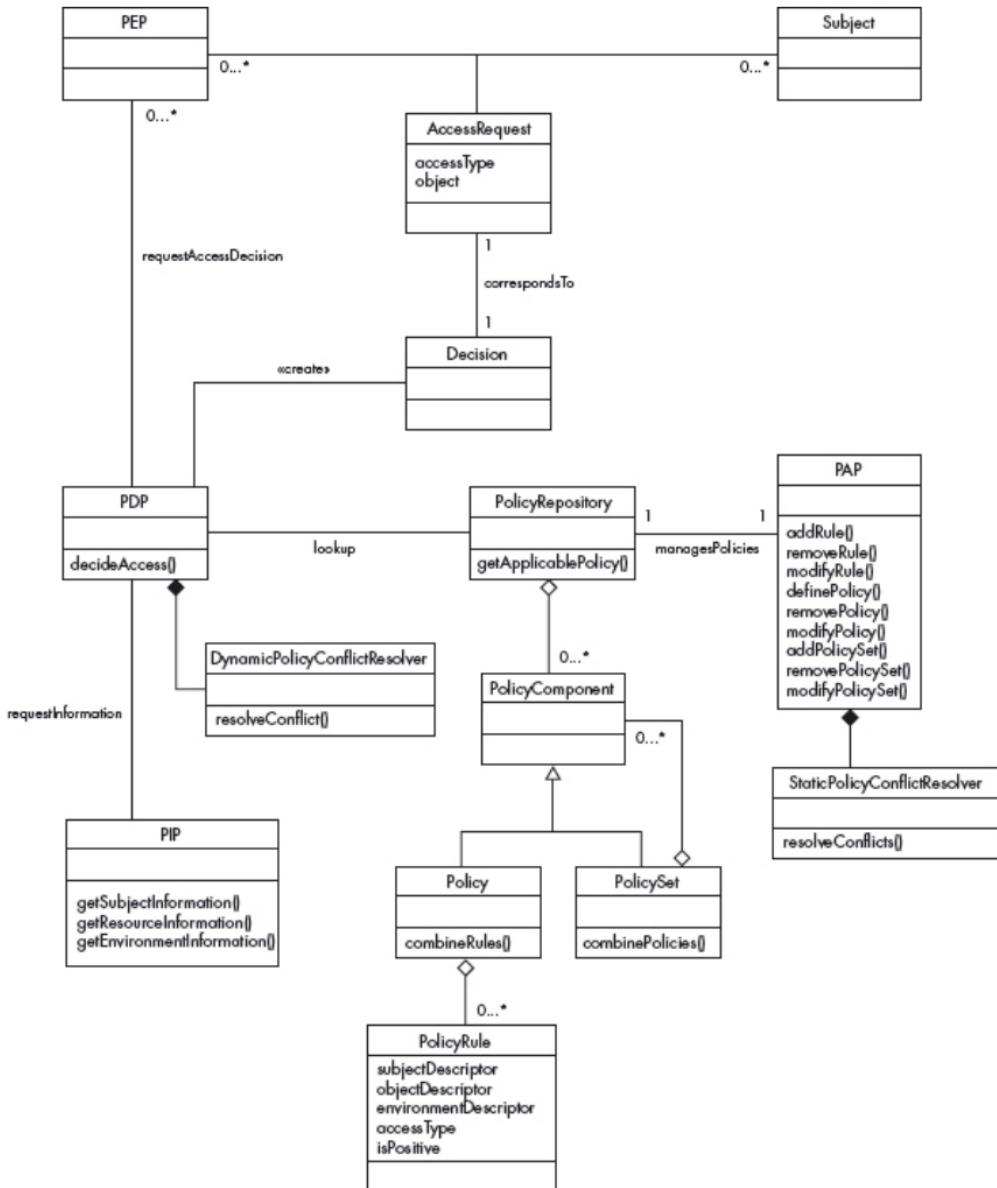


Figure 154: Class diagram for the POLICY-BASED ACCESS CONTROL pattern

The PolicyRepository is also responsible for retrieving the applicable rules by selecting those rules whose subjectDescriptor, resourceDescriptor and environmentDescriptor match the information about the subject, the resource and the environment obtained from the PIP, and whose accessType matches the required accessType from the request. The PAP is a unique point for administering the rules. In case the evaluation of the Policy leads to a conflict between the decisions of the applicable Rules, a part of the PDP, the DynamicPolicyConflictResolver, is responsible for producing a uniquely determined access decision. Similarly, a StaticPolicyConflictResolver is a part of the PAP and is responsible for identifying conflicting rules within the PolicyRepository.

## 72.7. Dynamics

Figure 155 shows a sequence diagram describing the most commonly used case of ‘Request access to an object’. The Subject’s request for accessing an Object is intercepted by a PEP, which forwards the request to the PDP. The PDP can retrieve information about the Subject,

the Object, and the current environment from the PIP. This information is used to retrieve the applicable Rules from the PolicyRepository.

The PDP can then compute the access decision by combining the decisions from the Rules forming the applicable policy and can finally send this decision back to the PEP. If the access has been granted by the PDP, the PEP forwards the request to the Object.

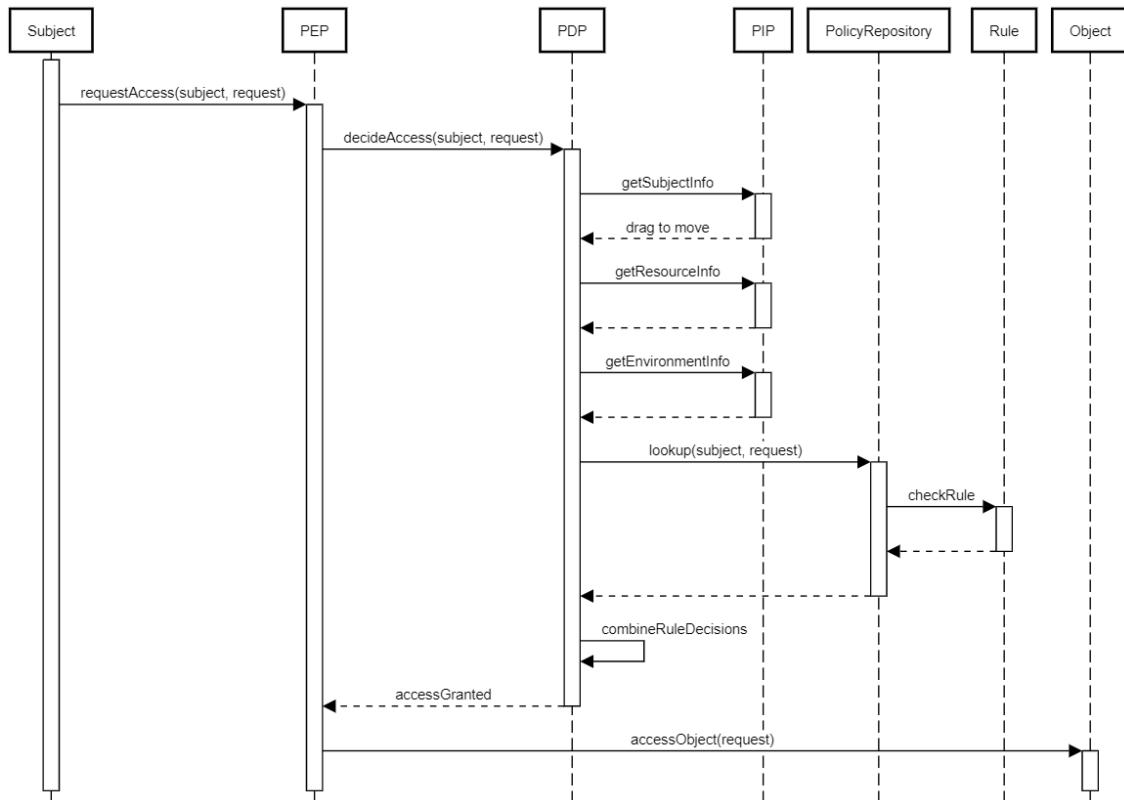


Figure 155: Sequence diagram for the use case 'Request access to an object'

## 72.8. Implementation

### 72.9. Example Resolved

Use of the POLICY-BASED ACCESS CONTROL pattern allows the company to centralize its rules. Now the billing department, as well as the technical department, the legal department and the corporate management department can insert their rules in the same repository, using the same format. The different components of the computer system that used to enforce policies directly (that is, e-mail server, file system, web service access control component and financial application) just need to intercept the requests and redirect them to the central policy decision point. To do that, each of them runs a policy enforcement point, which interfaces with the main policy decision point.

The rules could be grouped in the following way: a unique company policy set might include all other policies and express the fact that all policies coming from the corporate management should dominate all other policies. Each department would have their own policy, composed of rules from that department, and combined according to each department's policy.

Finally, a simple dynamic conflict resolver could be configured to enforce a closed policy in case of conflict. The rules can be managed easily, since they are written to the same repository, conflicts can be resolved, and there is a clearer view of the company's security policy.

## 72.10. Consequences

The POLICY-BASED ACCESS CONTROL pattern offers the following benefits:

- Since the access decisions are requested in a standard format, an access decision becomes independent of its enforcement. A wide variety of enforcement mechanisms can be supported and can evolve separately from the policy decision point.
- The pattern can support the access matrix, RBAC or multilevel models for access control.
- Since every access is mediated, illegal accesses are less likely to be performed.

The pattern also has some potential liabilities:

- It could affect the performance of the protected system since the central PDP/PolicyRepository/PIP subsystem may be a bottleneck in the system.
- Complexity.
- We need to protect the access control information.

## 72.11. Known Uses

- XACML (eXtensible Access Control Markup Language), defined by OASIS, uses XML for expressing authorization rules and for access decision following this pattern.
- Symlabs' Federated Identity Access Manager Federation is an identity management that implements identity federation. Its components include a PDP and PEPs.
- Components Framework for Policy-Based Admission Control, a part of the Internet 2 project, is a framework for the authentication of network components. It is based on five major components: Access Requester (AR), Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Repository (PR) and the Network Detection Point (NDP).
- XML and Application firewalls [1] also use policies.
- SAML (Security Assertion Markup Language) is an XML standard defined by OASIS for exchanging authentication and authorization data between security domains. It can be used to transmit the authorization decision.

## 72.12. See Also

- XACML patterns [1,2] is an implementation of this pattern.
- The ACCESS CONTROL LIST pattern and CAPABILITY patterns are specific implementations of this pattern.
- The PEP is just a Reference Monitor.
- This pattern can implement the Access Matrix [3] and ROLE-BASED ACCESS CONTROL patterns.

A general discussion of security policies, including some IETF models that resemble patterns, is given in [4].

## **72.13. References**

- [1] Delessy, N., & Fernandez, E. B. (2005, September). Patterns for the eXtensible Access Control Markup Language. In Proceedings of the 12th Pattern Languages of Programs Conference (PLoP2005). Monticello, Illinois. Retrieved September 18, 2011, from [http://hillside.net/plop/2005/proceedings/PLoP2005\\_ndelessyandebfernandez0\\_1.pdf](http://hillside.net/plop/2005/proceedings/PLoP2005_ndelessyandebfernandez0_1.pdf)
- [2] Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.
- [3] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc.
- [4] Sloman, M., & Lupu, E. (2002). Security and management policy specification. *IEEE network*, 16(2), 10-19.

## **72.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 73. Transport Layer Security

## 73.1. Intent

The TRANSPORT LAYER SECURITY pattern describes how to provide a secure channel between a client and a server by which application messages are communicated over the transport layer of the Internet. The client and the server are mutually authenticated, and the integrity of their data is preserved.

## 73.2. Example

A bank customer may want to check their account balance online. The bank uses the transport layer to transfer its confidential data. We need to protect this communication, as this confidential data is vulnerable to attack. The customer also has to ensure that the transactions are with the bank and not with an imposter, while the bank may need to verify that access is by a legitimate customer

## 73.3. Context

Users using applications that exchange sensitive information, such as web browsers for e-commerce or similar activities. The transport layer in TCP/IP provides end-to-end communication services for applications within a layered architecture of network components and protocols, and specifically convenient services such as connection-oriented data stream support, flow control and multiplexing.

## 73.4. Problem

The messages communicated between applications and servers on the transport layer are vulnerable to attack by intruders, who may try to read or modify them. Either the server or the client may be imposters.

The solution to this problem must resolve the following forces:

- Confidentiality and integrity. The data transferred in the transport layer between the client and the server could be intercepted and read or modified illegally.
- Authenticity. Either the server or the client could be an imposter, which may allow security breaches. A ‘man-in-the-middle’ attack is also possible, in which an attacker poses both as the client to the server and as the server to the client.
- Flexibility. Security protocol should be flexible and configurable, to be able to handle new attacks.
- Transparency. The security measures of the protocol should be transparent to users.
- Configurability. The protocol should allow users to select different degrees of security.
- Overhead. The overhead should be minimal, or users will not want to use the protocol.

### 73.5. Solution

Establish a cryptographic secure channel between the client and the server using algorithms that can be negotiated between the client and the server. Provide the means for client and server to authenticate each other. Provide a way to preserve the integrity of messages.

### 73.6. Structure

Figure 156 shows a class diagram for the basic architecture of the TRANSPORT LAYER SECURITY pattern. A Client requests some Service from the Server. The TSLProtocol controller conveys this request using an Authenticator to mutually authenticate the Server and the Client and creates a Secure Channel between them. AUTHENTICATOR and Secure Channel are patterns.

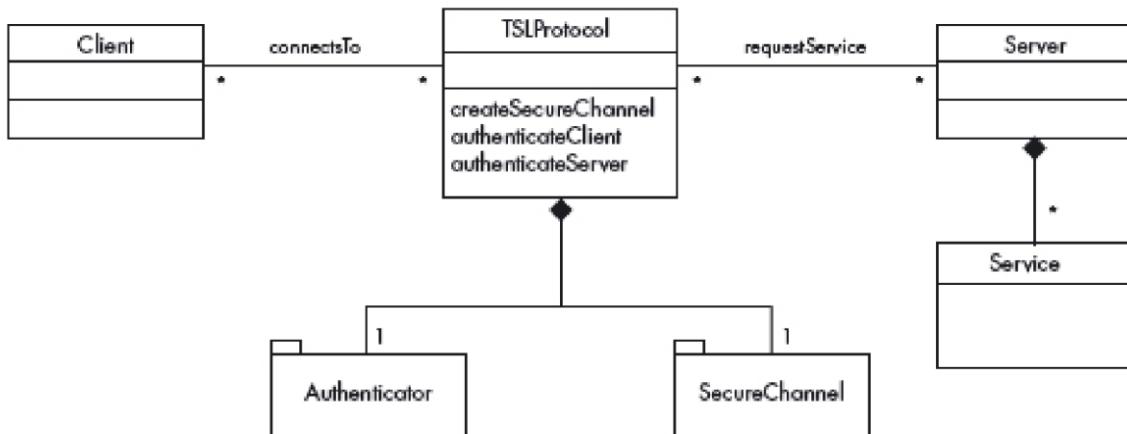


Figure 156: Class diagram for the TRANSPORT LAYER SECURITY pattern

### 73.7. Dynamics

We describe the dynamic aspects of the TRANSPORT LAYER SECURITY pattern using a sequence diagram for the following use case:

Use Case:	Request a Service – Figure 157
Summary	A Client requests a service and the TSLProtocol authenticates the request and creates a secure channel.
Actors	Client, Server.
Precondition	The security parameters of the secure exchange have been predefined.
Description	<ol style="list-style-type: none"><li>1. The Client makes a service request to the Server.</li><li>2. The TSLProtocol authenticates the Server to the Client and the Client to the Server.</li><li>3. The TSLProtocol creates a secure channel between the Server and the Client</li></ol>
Alternate Flow	<ul style="list-style-type: none"><li>• The authentication can fail.</li><li>• The creation of a secure channel can fail.</li></ul>
Postcondition	The Server accepts the request and grants the service.

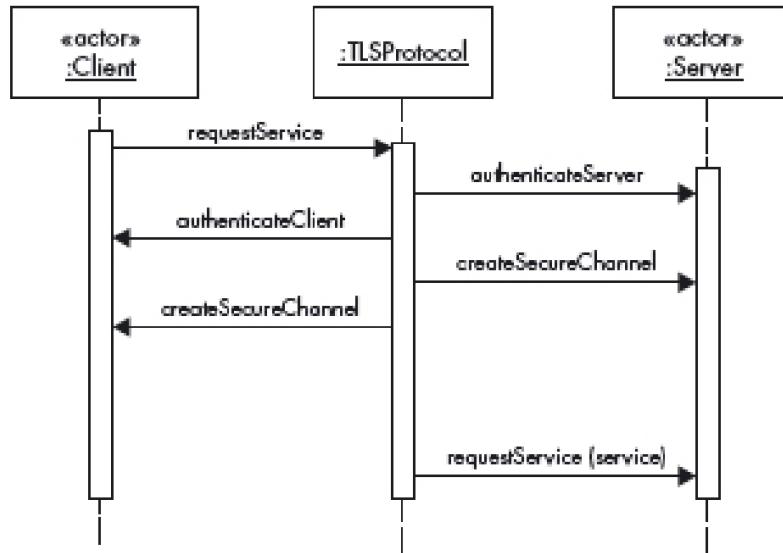


Figure 157: Sequence diagram for the use case ‘Request a service’

### 73.8. Implementation

One of the protocols that is dominant today for providing security at the transport layer is Secure Sockets Layer (SSL). The SSL protocol is a transport layer security protocol that was proposed and developed Netscape Communications in the 1990s. Transport Layer Security (TLS) is an IETF version of the SSL protocol, which has become a standard [1]. Much implementation advice can be found in [2].

The TLS protocol is partitioned into two main protocol layers, the TLS Record Protocol, and the TLS Handshake Protocol, executing above the TCP transport layer protocol, as shown in Figure 158 [3, 4]. There are other minor protocols at the handshake protocol layer, such as the Cipher Change Protocol, Alert Protocol and Application Protocol.

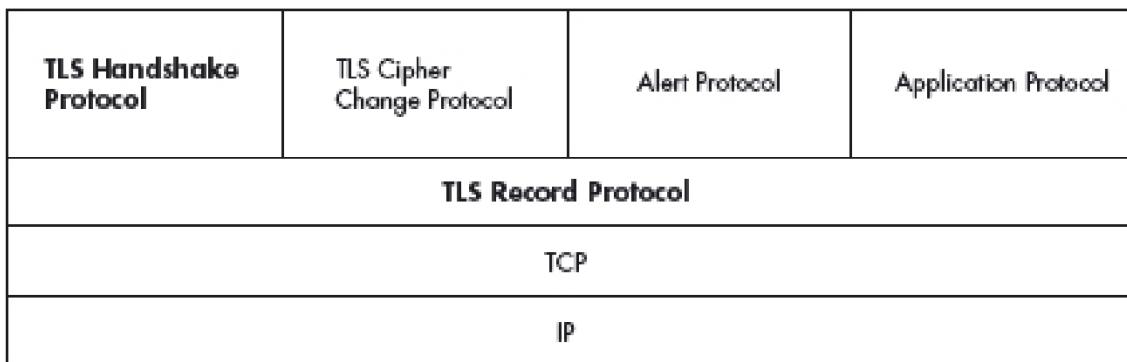


Figure 158: TLS layers

- Record Protocol. The TLS Record Protocol provides encryption and message authentication for each message. A connection is created using symmetric cryptography data encryption. The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). Messages include a message integrity check using a keyed message authentication code (MAC), computed using hash functions [5].

- Handshake Protocol. A TLS handshake supplies the authentication and key exchange operations for the TLS protocol. The security state agreed upon in the handshake is used by the TLS Record Protocol to provide session security. This protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives any data. The TLS Handshake Protocol provides connection security where the peers' identities can be authenticated using asymmetric cryptography. This authentication can be made optional but is generally required for at least one of the peers.

A TLS session is an association between a client and a server, created by the handshake protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

A session state is defined by the following parameters:

- Session identifier. This is generated by the server to identify a session with a chosen client.
- Peer certificate. The X.509 certificate of the peer.
- Compression method. A method used to compress data prior to encryption.
- Algorithm specification or CipherSpec. Specifies the encryption algorithm that encrypts the data and the hash algorithm used during the session.
- Master secret. 48-byte data, being a secret shared between the client and server.
- 'resumable'. A flag indicating whether the session can be used to initiate new connections.

The handshake protocol consists of the following four phases:

1. In the first phase, an initial connection is established to start the negotiation. The client and server exchange 'hello' messages that are used to establish security parameters used in the TLS session, and settings used during the handshake, such as the key exchange algorithm.
2. During the second phase, authentication, the server sends a certificate message to the client: this may include a server certificate when an RSA key exchange is used, or Diffie-Hellman parameters when a Diffie-Hellman key exchange is used. The server may also request a certificate from the client, using the certificateRequest message.
3. During the third phase, the client, if asked, may send its certificate to the server in a certificate message, along with a certificate Verify message, so that the server can verify certificate ownership (if the server requested a client certificate during the second phase). This phase includes the establishment of the security parameters such as the encryption key. The client must send either a pre-master secret encrypted using the server's public key, or public Diffie-Hellman parameters in the clientKeyExchange message, so that the client and server can compute a shared master secret.
4. In the fourth phase, the client and server finish the handshake, which implies that the client and server are mutually authenticated and have completed the required key exchange operations.

The other minor protocol layers from Figure 158 are discussed below:

- Cipher Change Protocol. This protocol signals transitions in cipherSpec, which is a session parameter explained above.

- Alert Protocol. This protocol raises alerts for the communication. This record should normally not be sent during normal handshaking or application exchanges. However, this message can be sent at any time during the handshake and up to the closure of a TLS session. If this record is used to signal a fatal error, the session will be closed immediately after sending the record. If the alert level is flagged as a warning, the remote partner can decide whether or not to close the session.
- Application Protocol. Now the handshake is completed, and the application protocol is enabled. This marks the start of data exchange between the server and the client.

### 73.8.1. Structure of the Handshake Protocol

The structure of the handshake protocol is shown in the class diagram in Figure 159. The Client requests a service from the Server at the Transport layer. The TSLHandshakeProtocolController uses client and server certificates to mutually authenticate the Client and the Server, then performs the clientKeyExchange.

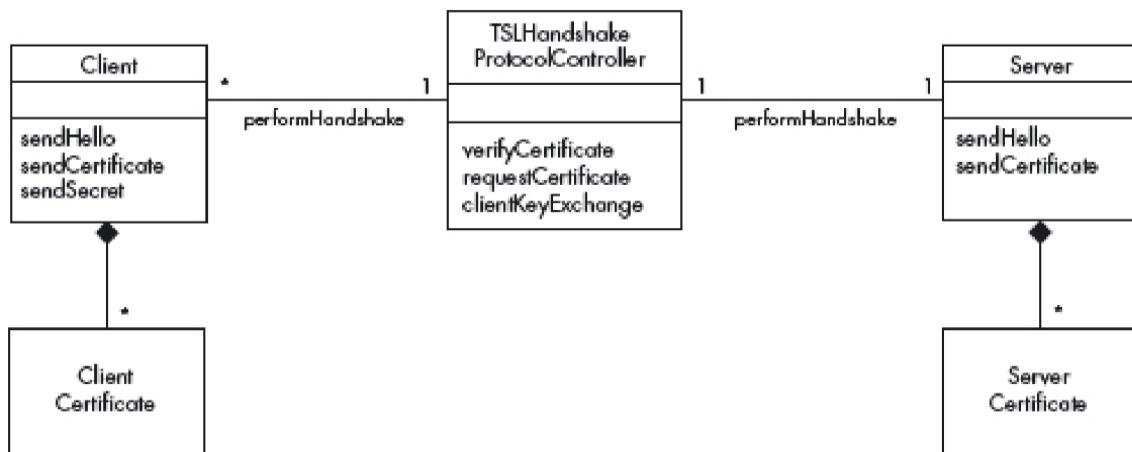


Figure 159: Class diagram for the TLS handshake protocol.

### 73.8.2. Dynamics of the Handshake Protocol

We describe the dynamic aspects of the TLS handshake using the sequence diagram shown in Figure 160.

Summary	A TLS handshake supplies the authentication and key exchange operations for the TLS protocol.
Actors	Client, Server.
Precondition	The Client has made a request for a service from the host Server and an initial connection has already been established. The Client and Server need to have a digital certificate, issued by some Certificate Authority.
Description	<ol style="list-style-type: none"> <li>1. The Client and Server exchange initial Hello messages.</li> <li>2. The ProtocolController requests the certificate from the Server and the Server sends the certificate.</li> <li>3. The server certificate is verified.</li> <li>4. The Server requests the certificate from the Client (optional).</li> <li>5. If asked, the Client sends the certificate to the Server.</li> <li>6. The client certificate is verified.</li> </ol>

7. The Client sends the predefined secret encrypted using the Server's public key which is the client key exchange.
8. The Client and the Server complete mutual handshake and the initial encryption parameters.
- Authentication of the Server or Client can fail. A certificate can be expired or outdated.
  - The Client could lose the encryption key while exchanging with the Server
- Alternate Flow
- Postcondition Client and Server can start exchanging data at the transport layer.

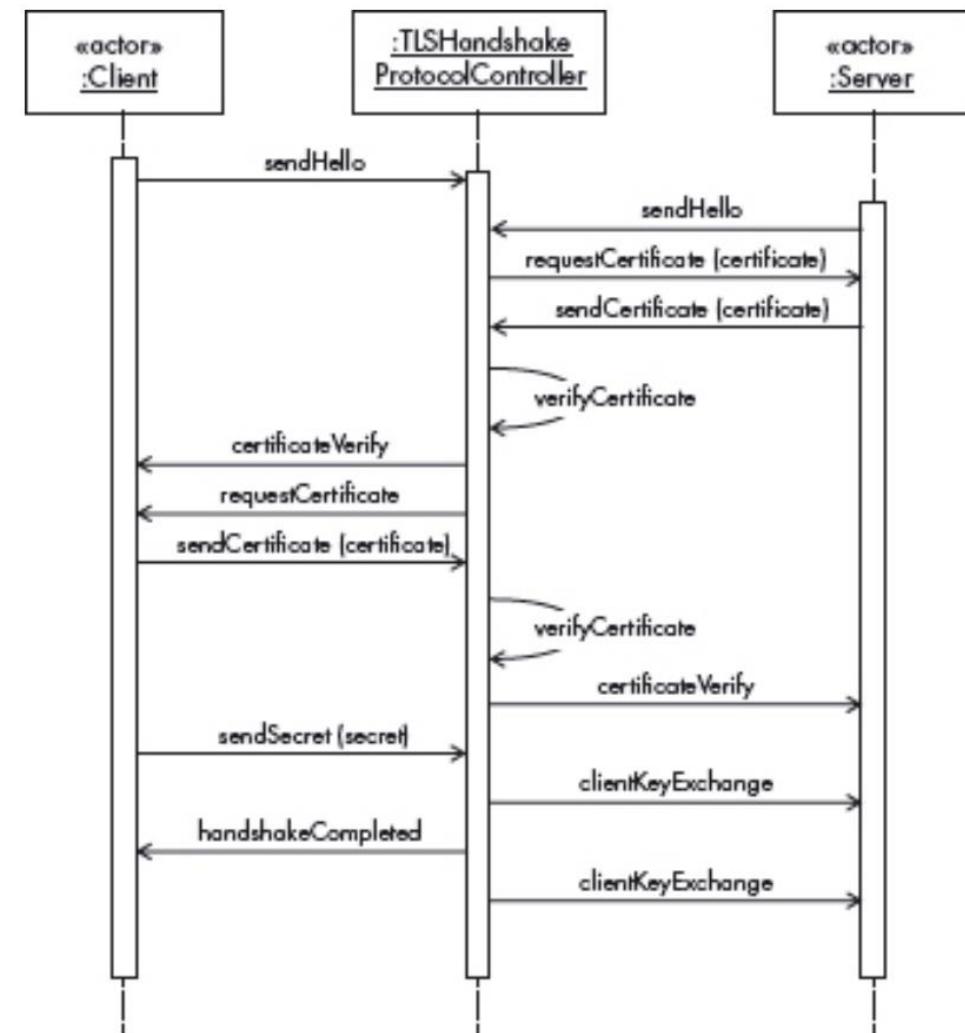


Figure 160: Sequence diagram for the TLS handshake use case

### 73.9. Example Resolved

When a request is made to the bank's server by an online client at the transport layer, the bank's server is authenticated to the customer, the customer is authenticated to the server and a secure channel is created between them. Now the client knows that online bank transactions are secure.

## 73.10. Consequences

The TRANSPORT LAYER SECURITY pattern offers the following benefits:

- Confidentiality and integrity. A secure channel is established between the server and the client, which can provide data confidentiality and integrity for the messages sent. We could add a logging system for the client at its endpoint for future audits.
- Authenticity. Both client and server can be mutually authenticated. Man-in-the-middle attacks can be prevented by mutual authentication.
- Flexibility. We can easily change the algorithms for encryption and authentication protocols.
- Transparency. The users don't need to perform any operation to establish a secure channel.
- Configurability. Users can select algorithms to obtain different degrees of security.

The pattern also has the following potential liabilities:

- Overhead. As seen from Figure 160, the overhead is significant for short sessions: many messages are needed.
- SSL/TLS is a two-party protocol; it is not designed to handle multiple parties. However, the MTLS variant can handle multiple parties

## 73.11. Variants

- WTLS. A modified version of TLS, called WTLS (Wireless TLS protocol) has been used in mobile systems. WTLS is based on TLS and is similar in some respects [6]. WTLS has been superseded in the WAP (Wireless Application Protocol) 2.0 standard by the End-toEnd Transport Layer Security specification.
- MultipleTLS (MTLS). This is an application-level protocol running over the TLS Record protocol. The MTLS provides application multiplexing over a single TLS session. Therefore, instead of associating a TLS session with each application, this protocol allows several applications to protect their communication over a single TLS session [7]. Some different versions of TLS are given below.
- TLS 1.0. TLS 1.0 is an upgrade of SSL Version 3.0 and is an IETF version of SSL. The differences between this protocol and SSL 3.0 are not large, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate. TLS 1.0 does include a means by which a TLS implementation can downgrade the connection to SSL 3.0, but this weakens security.
- TLS 1.1. TLS 1.1 is an update of TLS version 1.0. Significant differences include:
  - Added protection against cipher block chaining (CBC) attacks. In CBC mode, each block of plaintext is XORed with the previous cipher text block before being encrypted.
  - The implicit Initialization Vector (IV) was replaced with an explicit IV.
  - Change in handling of padding errors.
  - Support for registration of parameters.
- TLS 1.2. This is a revision of the TLS 1.1 protocol, which contains improved flexibility, particularly for negotiation of cryptographic algorithms. The major changes are:

- The MD5/SHA-1 combination in the pseudorandom function (PRF) has been replaced with cipher-suite-specified PRFs. All cipher suites in this document use P\_SHA256.
- The MD5/SHA-1 combination in the digitally signed element has been replaced with a single hash. Signed elements now include a field that explicitly specifies the hash algorithm used.
- Substantial cleanup to the client's and server's ability to specify which hash and signature algorithms they will accept.
- Addition of support for authenticated encryption with additional data modes.
- Tightening up of a number of requirements.
- Verification of data length now depends on the cipher suite (default is still 12) [8].
- EAP-TLS is a wireless authentication protocol for TLS [9].

## 73.12. Known Uses

- Mozilla Firefox versions 2 and above support TLS 1.0 [10].
- Internet Explorer (IE) 8 in Windows 7 and Windows Server 2008 support TLS 1.2 [11].
- Presto 2.2, used in Opera 10, supports TLS [12].

## 73.13. See Also

- The Authenticator pattern describes how to mutually authenticate a client and a server.
- The Secure Channel pattern describes a cryptographic channel used to communicate secure data.

## 73.14. References

- [1] Yasinsac, A., & Childs, J. (2005). Formal analysis of modern security protocols. *Information Sciences*, 171(1-3), 189-211.
- [2] Seltzer, L. (2012). Best Practices and Applications of TLS/SSL. white paper. Symantec Corporation.
- [3] Elgohary, A., Sobh, T. S., & Zaki, M. (2006). Design of an enhancement for SSL/TLS protocols. *computers & security*, 25(4), 297-306.
- [4] Stallings, W., Brown, L., Bauer, M. D., & Howard, M. (2012). *Computer security: principles and practice* (Vol. 2). Upper Saddle River: Pearson.
- [5] Stallings, W. (2003). *Cryptography and Network Security: Principles and Practice* (3rd edition), Prentice-Hall
- [6] Badra, M., Serhrouchni, A., & Urien, P. (2004). A lightweight identity authentication protocol for wireless networks. *Computer Communications*, 27(17), 1738-1745.
- [7] Badra, M., & Hajjeh, I. (2009, April). (D)TLS Multiplexing. Internet-Draft.  
<http://tools.ietf.org/html/draft-badra-hajjeh-mtls-05>
- [8] Dierks, T., & Rescorla, E. (2008). The transport layer security (TLS) protocol version 1.2.

[9] Wikipedia contributors. (n.d.). Extensible Authentication Protocol. Wikipedia. Retrieved November 6, 2012, from [https://en.wikipedia.org/wiki/Extensible\\_Authentication\\_Protocol](https://en.wikipedia.org/wiki/Extensible_Authentication_Protocol)

[10] Mozilla Newsgroup. (n.d.). mozilla.dev.tech.crypto.  
<http://www.mozilla.org/projects/security/pki/nss/ssl/>

[11] Microsoft. (n.d.). What is TLS/SSL. [http://technet.microsoft.com/en-us/library/cc784450\(v=WS.10\).aspx](http://technet.microsoft.com/en-us/library/cc784450(v=WS.10).aspx)

[12] Wikipedia. (2009). GNU Free Documentation License. <http://mjrutherford.org/files/2009-Spring-COMP-4704-TLS-Wikipedia.pdf>

### 73.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 74. Multilevel Security

## 74.1. Intent

In some environments data and documents may have critical value and their disclosure could bring serious problems. The MULTILEVEL SECURITY pattern describes how to categorize sensitive information and prevent its disclosure. It describes how to assign classifications (clearances) to users and classifications (sensitivity levels) to data, and how to separate different organizational units into categories. Access of users to data is based on policies, while changes to the classifications are performed by trusted processes that are allowed to violate the policies.

## 74.2. Example

The general command of an army has decided on a plan of attack in a war. It is extremely important that this information is not known outside a small group of people, or the attack may be a failure.

## 74.3. Context

In some environments data and documents may have critical value and their disclosure could bring serious problems.

## 74.4. Problem

How can you control access in an environment with sensitive documents so as to prevent leakage of information?

The solution to this problem must resolve the following forces:

- We need to protect the confidentiality and integrity of data based on its sensitivity.
- Users have to be allowed to read documents based on their position in the organization.
- There should be a way to increase or decrease the ability of users to read documents and the sensitivity of the documents. Otherwise, people promoted to higher positions, for example, will not be able to read sensitive documents, and we will end up with a proliferation of sensitive and obsolete documents.

## 74.5. Solution

Assign classifications (as clearances) to users and classifications (as sensitivity levels) to data. Separate different organizational units into categories. For example, classifications may include levels such as ‘top secret’, ‘secret’ and so on, and categories may include units such as engDept, marketingDept and so on. For confidentiality purposes, access of users to data is based on policies defined by the Bell-LaPadula model [1], while for integrity the policies are defined by Biba’s model [2]. Changes to the classifications are performed by trusted processes that are allowed to violate the policies of these models.

## 74.6. Structure

Figure 161 shows the class diagram of the MULTILEVEL SECURITY pattern. The UserClassification and DataClassification classes define the active entities and the objects of access respectively. Both 402 classifications may include categories and levels. Trusted processes are allowed to assign users and data to classifications, as defined by the Assignment class.

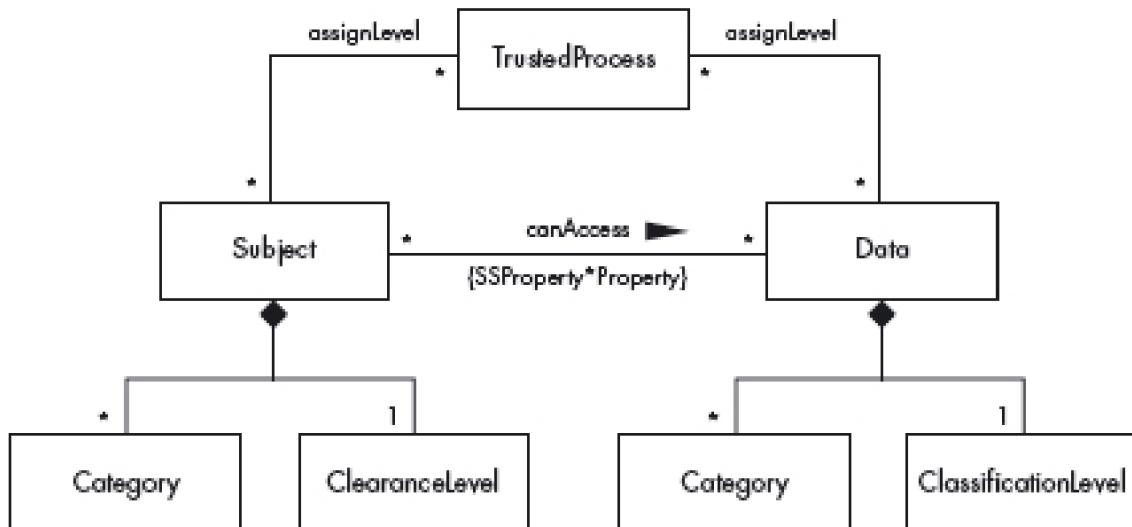


Figure 161: Class diagram for the MULTILEVEL SECURITY pattern

## 74.7. Dynamics

## 74.8. Implementation

Data classification is a tedious task, because every piece of information or document must be examined and assigned a classification tag. New documents may get automatic tags based on their links to other documents. User classifications are based on users' rank and units of work and are only changed when they change jobs. It is hard to classify users in commercial environments in this way: for example, in a medical system it makes no sense to assign a doctor a higher classification than a patient, because a patient has the right to see their record.

## 74.9. Example Resolved

The group involved in planning attacks, as well as all the related documents it produces, are given a classification of 'top secret'. This will prevent leakage towards lower-level army staff.

## 74.10. Consequences

The MULTILEVEL SECURITY pattern offers the following benefits:

- The classification of users and data is relatively simple and can follow organization policies.

- This model of the pattern can be proved to be secure under certain assumptions [2].
- The pattern is useful to isolate processes and execution domains.

The following potential liabilities may arise from applying this pattern:

- Implementations should use labels in data to indicate their classification. This assures security: if not done, the general degree of security is reduced.
- We need trusted programs to assign users and data to classifications.
- Data must be able to be structured into hierarchical sensitivity levels and users should be able to be structured into clearances. This is usually hard, or even impossible, in commercial environments.
- This model can handle only secrecy and prevention of leakage of information. A dual model is needed to also handle integrity.
- Covert channels may break the assumed security

## **74.11. Variants**

It is possible to define a similar pattern to control integrity in multilevel models according to the Biba rules [1].

## **74.12. Known Uses**

This pattern has been used by several military-sponsored projects and in a few commercial products, including DBMSs (Informix, Oracle) and operating systems (Pitbull [3] and HP's Virtual Vault [4]).

## **74.13. See Also**

The concept of roles can also be applied when implementing this pattern, role classifications replacing user classifications.

## **74.14. References**

- [1] Gollmann, D. (2010). Computer security. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(5), 544-554.
- [2] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc..
- [3] Argus Systems Group. (n.d.). Trusted OS security: Principles and Practice. Retrieved October 27, 2012, from <http://www.argus-systems.com/Products/products.html>
- [4] Hewlett Packard Corporation. (n.d.). Virtual Vault. <http://www.hp.com/security/products/virtualvault>

## **74.15. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **75. Security Logger and Auditor**

## **75.1. Intent**

The SECURITY LOGGER AND AUDITOR pattern describes how to keep track of users' actions in order to determine who did what and when. It logs all security-sensitive actions performed by users and provides controlled access to records for audit purposes.

## **75.2. Example**

A hospital uses RBAC to define the rights of its employees. For example, doctors and nurses can read and write medical records and related patient information (lab tests and medicines). When a famous patient came to the hospital, one of the doctors, who was not treating him, read his medical record and leaked this information to the press. When the leak was discovered, there was no way to find out which doctor had accessed the patient's records.

## **75.3. Context**

Any system that handles sensitive data, in which it is necessary to keep a record of access to data.

## **75.4. Problem**

How can we keep track of users' actions in order to determine who did what and when? The solution to this problem must resolve the following forces:

- Accuracy. We should faithfully record what a user or process has done with respect to the use of system resources.
- Security. Any information we use to keep track of what the users have done must be protected. Unauthorized reading may reveal sensitive information. Tampering may erase past actions.
- Forensics. When a misuse of data occurs, it may be necessary to audit the access operations performed by users to determine possible unauthorized actions, and maybe trace the attacker or understand how the attack occurred.
- System improvement. The same misuses may keep occurring; we need to learn from past attacks.
- Compliance. We need a way to verify and to prove to third parties that we have complied with institution policies and external regulations.
- Performance. We need to minimize the overhead of logging.

## **75.5. Solution**

Each time a user accesses some object, we record this access, indicating the user identifier, the type of access, the object accessed and the time when the access happened.

The database of access entries must have authentication and authorization systems, and possibly an encryption capability.

## 75.6. Structure

In Figure 162 User operations are logged by the LoggerAuditor. The LoggerAuditor keeps the Log of user accesses, in which each access is described by a LogEntry. The security administrator (SecAdmin) activates or deactivates the Log. The Auditor can read the Log to detect possible unauthorized actions.

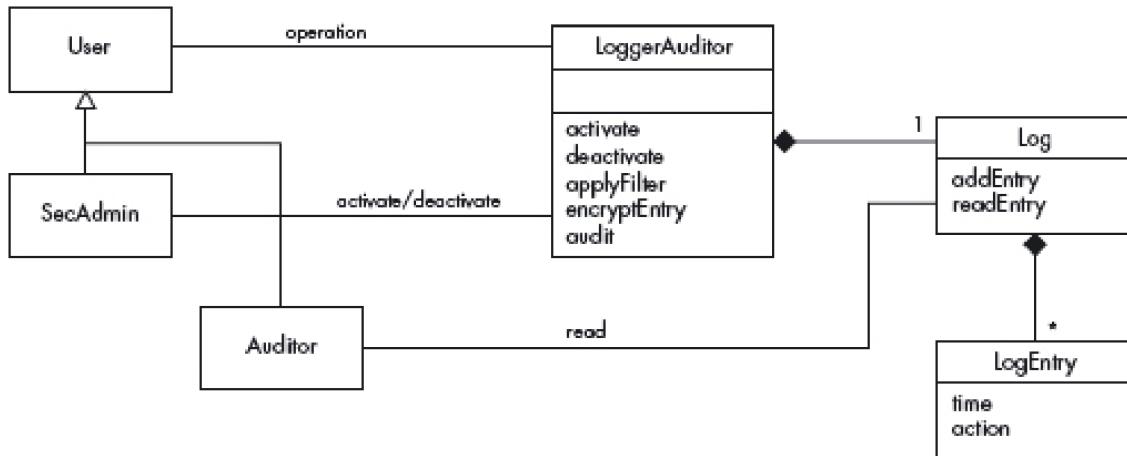


Figure 162: Class diagram of the SECURITY LOGGER AND AUDITOR pattern

## 75.7. Dynamics

Possible use cases include ‘Log user access’, ‘Audit log’, ‘Query log database’.

A sequence diagram for the use case ‘Log user access’ is shown in Figure 163. The User performs an operation to apply an access type on some object: operation (accessType, object). The LoggerAuditor adds an entry with this information, and the name of the user, to the Log. The Log creates a LogEntry, adding the time of the operation.

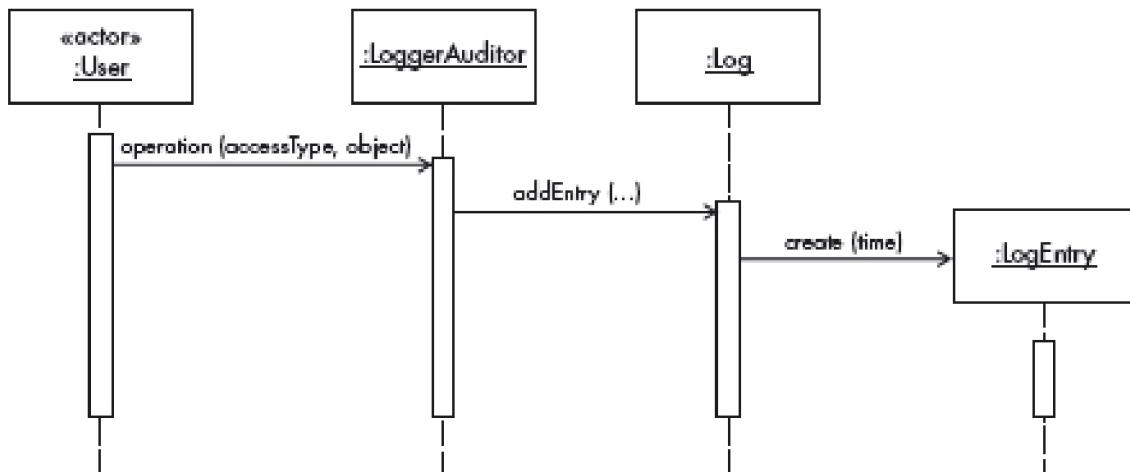


Figure 163: Sequence diagram for the use case ‘Log user access’

## 75.8. Implementation

The class diagram shown in Figure 162 provides a clear guideline for implementation, since its classes can be directly implemented in any object-oriented language. We need to define

commands to activate or deactivate logging, apply filters, indicate devices to be used, allocate amount of storage, and select security mechanisms. One can filter some logging by selecting users, events, importance of events, times, and objects in the filters. Administrative security actions, for example account creation/deletion, assignment of rights and others, must also be logged.

Logging is performed by calling methods on the LoggerAuditor class. Every non-filtered user operation should be logged. Logged messages can have levels of importance associated with them.

Audit is performed by an auditor reading the log. This can be complemented with manual assessments that include interviewing staff, performing security vulnerability scans, reviewing application and operating system access controls, and analyzing physical access to the systems [1]. The Model-View-Controller pattern can be used to visualize the data using different views during complex statistical analysis of the log data.

## 75.9. Example Resolved

After the incident, the hospital installed a SECURITY LOGGER AND AUDITOR, so in the future such violations can be discovered.

## 75.10. Consequences

The SECURITY LOGGER AND AUDITOR pattern offers the following benefits:

- Security. It is possible to add appropriate security mechanisms to protect recorded data, for example access control and/or encryption.
- Forensics. The pattern enables forensic auditing of misused data objects. Records of access can be used to determine whether someone has maliciously gained access to data. This pattern can also be used to log access to data objects by system processes. For example, malicious code planted in the system can be tracked by finding processes that have misused objects.
- System improvement. By studying how past attacks happened, we can improve the system security. Compliance. Auditing a log can be used to verify and prove that institutional and regulatory policies have been followed.
- Performance. We can reduce overhead by parallel or background logging. We can also not log some events not considered significant. Finally, we can merge this log with the recovery log, needed for possible rollback.

The pattern has the following potential liabilities:

- It can incur significant overhead since each object access has to be logged.
- A decision must be made by software designers as to the granularity at which objects are logged. There is a trade-off between security and performance.
- It is not easy to perform forensic analysis, and specialists are required.
- Protecting the log adds some overhead and cost.

## **75.11. Variants**

Most systems have a system logger, used to undo/rollback actions after a system crash. That type of logger has different requirements, but sometimes is merged with the security logger [2]. System logs are of interest to system and database administrators, while security logs are used by security administrators, auditors, and system designers.

Another variant could include the automatic raising of alarms by periodic examination of the log, searching records that match a number of rules that characterize known violations. For example, intrusion detection systems use this variant.

We can also add logging for reliability, to detect accidental errors.

## **75.12. Known Uses**

- Most modern operating systems, including Microsoft Windows [3], AIX [4], Solaris and others have security loggers.
- SAP uses both a security audit log and a system log [2].

## **75.13. See Also**

- The Secure Logger is a pattern for J2EE [5]. It defines how to capture application-specific events and exceptions to support security auditing. This pattern is mostly implementation-oriented and does not consider the conceptual aspects discussed in our pattern. It should have been called a ‘security logger’ because it does not include any mechanisms to protect the logged information.
- Martin Fowler has an Audit Log analysis pattern [6] for tracking temporal information. The idea is that whenever something significant happens, you write some record indicating what happened and when it happened.
- Patterns for authentication: how can we make sure that a subject is who they say they are?
- AUTHORIZATION describes how we can control who can access to which resources, and how, in a computer system.

## **75.14. References**

[1] Wikipedia contributors. (n.d.). Information security audit. Wikipedia.  
[https://en.wikipedia.org/wiki/Information\\_security\\_audit](https://en.wikipedia.org/wiki/Information_security_audit)

[2] Software AG. (2009). webMethods Audit Logging Guide.  
[http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8\\_ga/Cross\\_Product/8-OSP1\\_Audit\\_Logging\\_Guide.pdf](http://documentation.softwareag.com/webmethods/wmsuites/wmsuite8_ga/Cross_Product/8-OSP1_Audit_Logging_Guide.pdf)

[3] Smith, F. R. (n.d.). Auditing Users and Groups with the Windows Security Log.  
<http://www.windowsecurity.com/articles/auditing-users-groups-windows-security-log.html>

[4] AIX. (n.d.). System Security Auditing. <http://www.aixmind.com/?p=1019>

[5] Steel, C., Nagappan, R., Lai, R. (2006). Securing the Web Tier: Design Strategies and Best Practices (Chapter 9). Sun.

[6] Fowler, M. (n.d.). Audit Log. <http://martinfowler.com/ap2/auditLog.html>

## 75.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 76. Session-Based Role-Based Access Control

## 76.1. Intent

The SESSION-BASED ROLE-BASED ACCESS CONTROL pattern allows access to resources based on the role of the subject and limits the rights that can be applied at a given time based on the roles defined by the access session.

## 76.2. Example

John is a developer on a project. He is also a project leader for another project. As a project leader he can evaluate the performance of the members of his project. He combines his two roles and adds several flattering evaluations about himself in the project where he is a developer. Later, his manager, thinking that the comments came from the project leader of the project on which John is a developer, gives John a big bonus.

## 76.3. Context

Any environment in which we need to control access to computing resources, in which users can be classified according to their jobs or their tasks, and in which we assign rights to the roles needed to perform those tasks.

We assume the existence of a Session pattern that can be used for the solution.

## 76.4. Problem

In an organization a user may play several roles. However, for each access the user must act only within the authorizations of a single role (that is, within the context of the role) or combinations of roles that do not violate institution policies. How can we force subjects to follow the policies of the institution when using their roles?

In addition to the forces defined for the CONTROLLED ACCESS SESSION pattern, the solution to this problem must resolve the following forces:

- People in institutions have different needs for access to information, according to their functions. They may have several roles associated with specific functions or tasks.
- We want to help the institution to define precise access rights for its members so that the least privilege policy can be applied when they perform specific tasks.
- Users may have more than one role and we may want to enforce policies such as separation of duty, where a user cannot be in two or more specific roles in the same session.

## 76.5. Solution

A subject may have several roles. Each role collects the rights that a user can activate at a given moment (execution context), while a session controls the way in which roles are used and can enforce role exclusion at execution time.

## 76.6. Structure

The structure of the SESSION-BASED ROLE-BASED ACCESS CONTROL pattern is shown in the class diagram in Figure 164. The class Role is an intermediary between Subject and Object, holding all authorizations a user possesses while performing the role, and acts here as an execution context. Within a Session, only a subset of the roles assigned to a Subject may be activated, just those necessary to perform the intended task. Roles may be composed according to a Composite pattern [1], in which higher-level roles acquire (inherit) rights from lower-level roles.

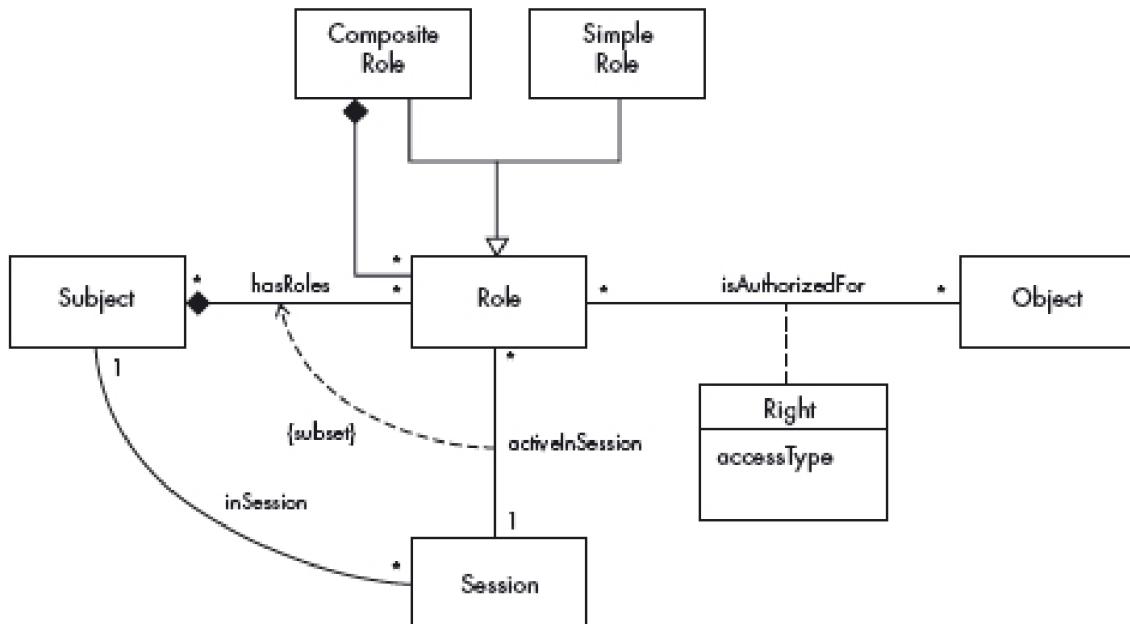


Figure 164: Class diagram for the SESSION-BASED ROLE-BASED ACCESS CONTROL pattern

## 76.7. Dynamics

Figure 165 shows a sequence diagram for the use case ‘Request access to an object’. A Subject has already opened a Session (see Session pattern) and requests access to an object in a specific way (accessType). The session uses the corresponding ReferenceMonitor, which in turn checks whether the rights of the Session roles allow the access. If so, the access is permitted.

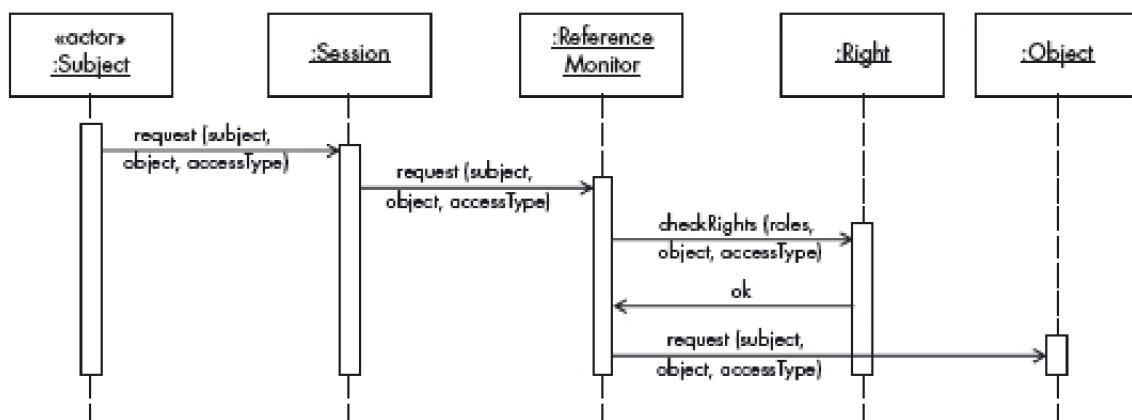


Figure 165: Sequence diagram for the use case ‘Request access to an object’

## **76.8. Implementation**

1. Determine the roles the system should contain (role catalog), according to the user functions or tasks.
2. Collect lists of incompatible roles and use these lists when a session is started (static constraints). These constraints can be defined using OCL or some other formal language as additions to the class diagram of the pattern.
3. Determine the number of roles which may be active within a session (dynamic constraints).
4. When a user opens a session, they must declare what roles they intend to use, and the system will open the corresponding session, or refuse to do so in the case of conflicts.

See [2] for an example of a real implementation.

## **76.9. Example Resolved**

When John logs on in the project where he is a developer, he only gets the rights for a developer and cannot add evaluations. When he logs on in the project where he is a project leader, he can only evaluate the members of his group. He cannot combine the rights of his role in the same session, and now he only gets legitimate evaluations.

## **76.10. Consequences**

In addition to the benefits mentioned for CONTROLLED ACCESS SESSION, additional benefits of the SESSION-BASED ROLEBASED ACCESS CONTROL pattern are:

- Sessions may include all needed roles for those subjects authorized for some task.
- Users can activate more than one session at a time for functional flexibility (some tasks may require multiple roles).
- Fine-grained rights can be assigned to roles to enforce a need-to know policy.
- When a session is open, we can exclude roles that violate institution policies.

The pattern has the following potential liabilities:

- Additional conceptual complexity is required to define which roles can be used together and which should be mutually exclusive.
- User confusion if they have to use several roles to perform their work.

## **76.11. Known Uses**

- The structure and dynamics of a Session-Based RBAC are implemented in the security module CSAP [3] of the Webocrat system. Webocrat is a portal supporting E-Democracy which was developed within the European Webocracy project (FP5IST-1999- 20364) between 2000-2003.
- Views in relational databases can be used to define sets of rights. Controlling the use of views by roles can control the use of rights in sessions. In both Oracle and DB2 SQL can be used to define restricted views based on roles [4].

## **76.12. See Also**

This pattern is a combination of the CONTROLLED ACCESS SESSION pattern and the Role-based access control pattern. As indicated earlier, structuring of roles can be represented by a Composite pattern. A Reference Monitor pattern is needed to enforce the use of rights during execution.

## **76.13. References**

- [1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [2] Fernandez, E. B., & Pernul, G. (2006, October). Patterns for session-based access control. In *Proceedings of the 2006 conference on Pattern languages of programs* (pp. 1-10).
- [3] Dridi, F., Fischer, M., & Pernul, G. (2003, May). CSAP—An Adaptable Security Module for the E-Government System Webocrat. In *IFIP International Information Security Conference* (pp. 301-312). Springer, Boston, MA.
- [4] Elmasri, R., Navathe, S. B. (2003). *Fundamentals of Database Systems (4<sup>th</sup> edition)*. Addison-Wesley.

## **76.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# **77. Signature-Based IDS**

## **77.1. Intent**

The SIGNATURE-BASED IDS pattern describes how to check every request for access to the network against a set of existing attack signatures, to detect possible attacks and trigger an appropriate response.

## **77.2. Example**

Our company has a firewall to control traffic from the Internet. However, we are still plagued by viruses and other attacks that penetrate the firewall. We need to improve our defense against such attacks.

## **77.3. Context**

Distributed systems executing applications that may provide services to remote nodes. Access to the network can be from the Internet or from other external networks.

## **77.4. Problem**

Whenever data is accessed from the distrusted networks, there is always a possibility that this access can be harmful to the local node. We need to detect possible attacks while they are occurring. Security techniques such as authentication and firewalls are usually implemented to provide security, but we need additional defenses to detect whether an access request is a possible attack or not.

The solution to this problem must resolve the following forces:

- Known attacks. It is easier to protect the system against known attacks. Many attacks are new instances of known attacks and have a well-defined attack signature.
- Completeness. If we have a complete collection of known attacks and their signatures, it is easier to detect an attack exhibiting one of these signatures.
- Flexibility. Hard coding the type of attack can be done easily, but it will be hard and time-consuming to adapt to attack patterns that keep changing constantly.

## **77.5. Solution**

Detect the occurrence of attacks by matching the current attack signature against the signature of previously known attacks.

## **77.6. Structure**

Figure 166 shows the class diagram of this pattern. The IDS intercepts an access request for a service. An EventProcessor processes the information and feeds this processed information to a AttackDetector, which tries to match the sequence of requests to the signatures in the

AttackSignatureInformation and decides whether or not the request is an intrusion. If an attack is detected by getting a match of signatures, some appropriate Response is raised.

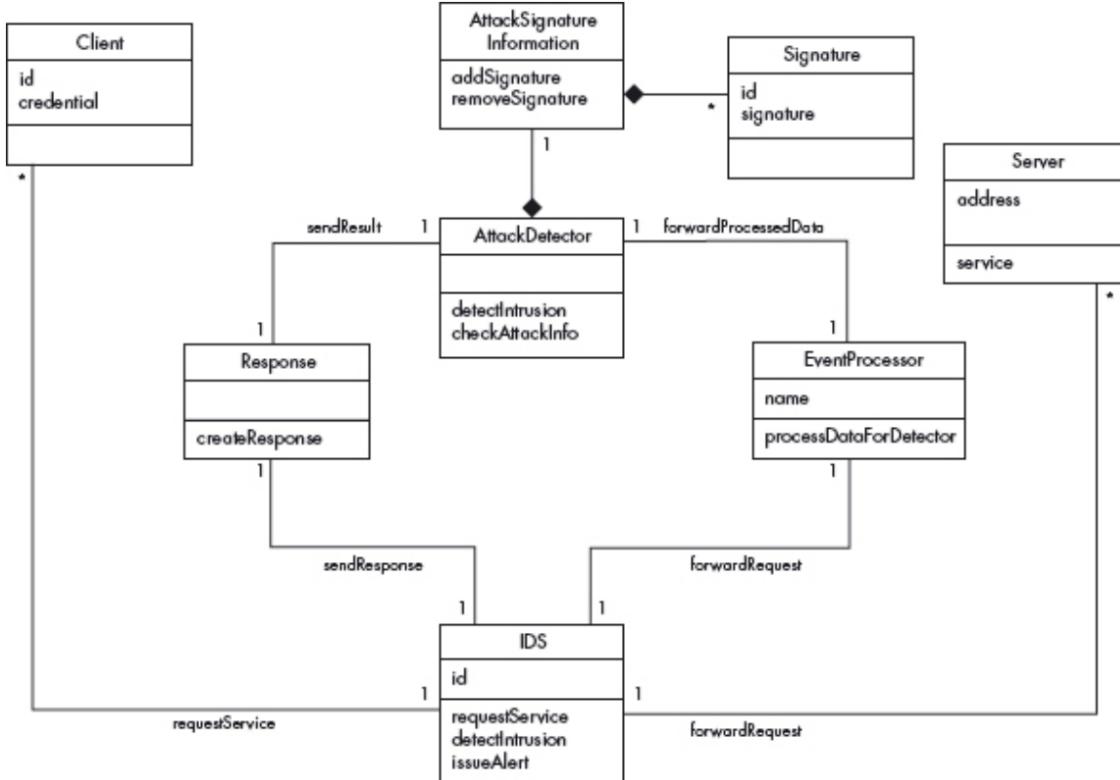


Figure 166: Class diagram for the SIGNATURE-BASED IDS pattern

## 77.7. Dynamics

We describe the dynamic aspects of the SIGNATURE-BASED IDS pattern using a sequence diagram for the following use case.

Use Case:	Detect an Intrusion – Figure 167
Summary	The Client requests a service from the Host. The Signature-Based IDS intercepts the message and determines whether the signature of the event matches an existing attack signature. If the request is an attack, appropriate response is raised.
Actors	Client, Server.
Precondition	Information about attack signatures is available.
Description	<ol style="list-style-type: none"> <li>A Client makes a service request for a service to the Host.</li> <li>The IDS send the request event to an EventProcessor.</li> <li>The EventProcessor processes the event as required by the AttackDetector and passes the processed event data to the AttackDetector.</li> <li>The AttackDetector tries to detect whether this request is an attack or not by comparing the signature of the event with the available signatures in the AttackSignatureInformation.</li> <li>If a match is detected, a Response is created. <ul style="list-style-type: none"> <li>The AttackSignatureInformation may not be able to detect an attack (a false negative).</li> </ul> </li> </ol>
Alternate Flow	

- The AttackSignatureInformation can match and may indicate an attack when no attack is present (a false positive).
- Postcondition** If an attack is detected while it is happening, suitable preventive measures can be adopted.

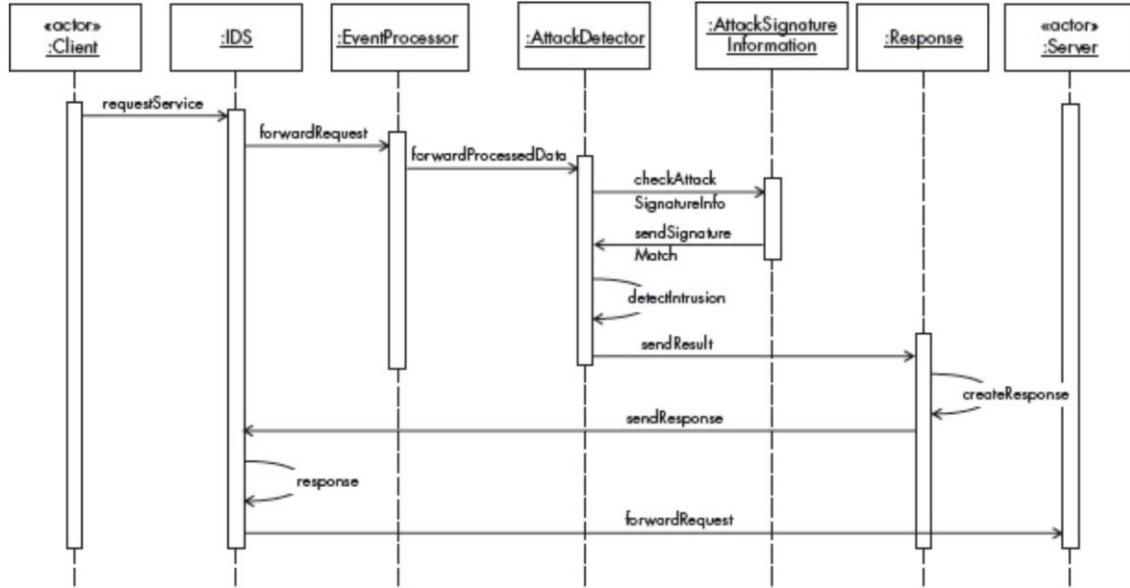


Figure 167: Sequence diagram for the use case 'Detect an intrusion'

## 77.8. Implementation

We first need to create a database with a set of all the known or expected attack patterns. We then select a detection algorithm. Some possible detection algorithms are:

- Expression matching. The simplest form of misuse detection involves searching the event stream for known attack pattern expressions [1].
- State transition analysis. The whole process is a network of states and transitions. Every observed event is applied to finite state machine instances (each representing an attack scenario), possibly causing transitions [1].
- Dedicated languages. Some IDS implementations describe intrusion signatures using specialized languages varying from compiled expressions to programming languages such as Java. A signature takes the form of a specialized program, with raw events as input. Any input triggering a filtering program, or input that matches internal alert conditions, is recognized as an attack [1].
- Genetic algorithms. A genetic algorithm is used to search for the combination of known attacks (expressed as a binary vector, each element indicating the presence of a particular attack) that best matches the observed event stream [1].

## 77.9. Example Resolved

We added an intrusion detection system beside the existing firewall to the system. Now any request authorized by the firewall is checked against known attack signatures to detect whether the access request is a possible attack. If we detect an attack, an alert can be raised, and the firewall can block the request.

## **77.10. Consequences**

The SIGNATURE-BASED IDS pattern offers the following benefits:

- Known attacks. Detection can be effective against known attacks.
- Completeness. If all known attack signatures are available in the database, attacks can be detected in real time.
- Flexibility. It is relatively easy to add new attacks to the detection set.

The pattern also has the following potential liabilities:

- It only works for known attacks: a new attack will not be detected. We have to constantly update the database with new attack signatures.
- Some attacks don't have well-defined signatures, or the attacker may disguise the signatures. This may lead to false positives and false negatives.
- Some attacks may be so fast that it may be hard to recognize them in real time.
- Attack patterns are closely tied to a given environment (operating system, hardware architecture, and so on) and cannot be applied easily to other systems.

## **77.11. Known Uses**

- An IDS can be combined with a firewall, as is done in Nokia's network systems [2].
- Cisco IDS utilizes detection techniques including stateful pattern recognition, protocol parsing, heuristic detection, and anomaly detection [3].
- LIDS is a signature-based intrusion detection/defense system for the Linux kernel [4].
- RealSecure [5] by Internet Security Systems is an IDS adapted by IBM for intrusion detection packages. It can monitor TCP, UDP and ICMP traffic and, if a match is found, countermeasures can be implemented along with read/write server locking, IP blocking and other measures. This product is bundled with CheckPoint Software's Firewall [6].

## **77.12. See Also**

- This pattern is a special (concrete) case of the Reference Monitor pattern.
- Firewall patterns complement this pattern.
- The response class could be implemented as a Strategy pattern [7].

## **77.13. References**

[1] Verwoerd, T., & Hunt, R. (2002). Intrusion detection techniques and approaches. *Computer communications*, 25(15), 1356-1365.

[2] Nokia. (2001, April). Combining Network Intrusion Detection with Firewalls for Maximum Perimeter Protection. White Paper.

[http://www.itu.dk/courses/DSK/F2003/Combining\\_IDS\\_with\\_Firewall.pdf](http://www.itu.dk/courses/DSK/F2003/Combining_IDS_with_Firewall.pdf)

[3] Cisco Systems. (n.d.). Cisco Intrusion Detection.

<http://www.cisco.com/warp/public/cc/pd/sqsw/sqidsz/>

[4] Linux. (n.d.). Intrusion Detection System. <http://www.lids.org/>

- [5] IBM. (n.d.). Real Secure Intrusion Detection Systems.  
<http://publib.boulder.ibm.com/infocenter/sprotect/v2r8m0/index.jsp>
- [6] Checkpoint Software Technologies. (n.d.). Retrieved July 20, 2010, from  
<http://www.checkpoint.com/products/softwareblades/ipsec-virtual-private-network.html>
- [7] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.

## 77.14. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

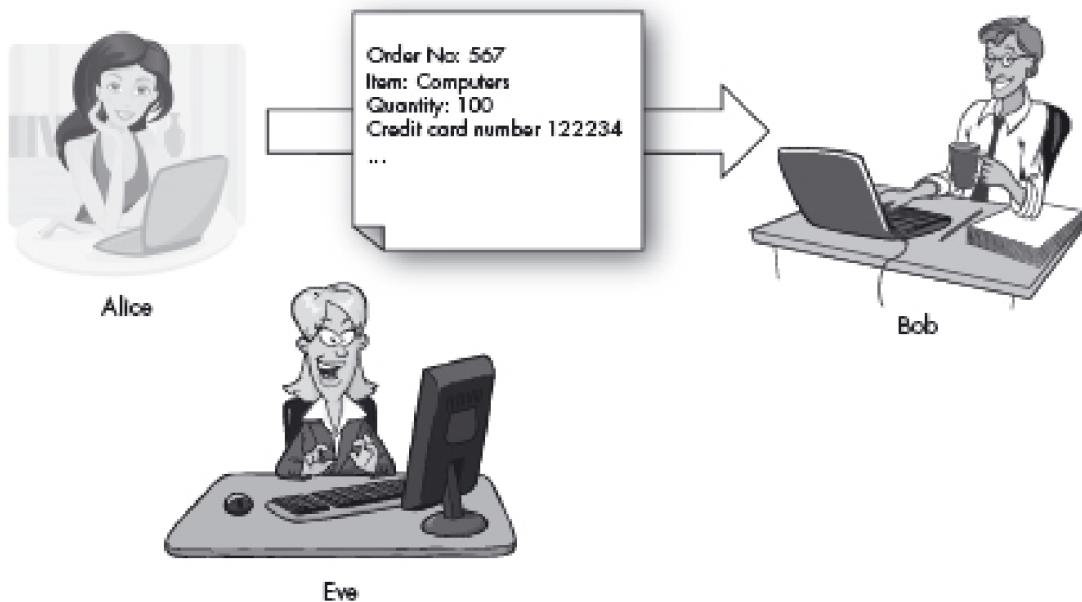
# 78. Symmetric Encryption

## 78.1. Intent

Encryption protects message confidentiality by making a message unreadable to those that do not have access to the key. Symmetric encryption uses the same key for encryption and decryption.

## 78.2. Example

Alice in the purchasing department regularly sends purchase orders to Bob in the distribution office. A purchase order contains sensitive data such as credit card numbers and other company information, so it is important to keep it secret. Eve can intercept her messages and may try to read them to get the confidential information. As part of her work Alice needs to communicate with only a few employees in the company.



## 78.3. Context

Applications that exchange sensitive information over insecure channels and where the number of users and applications is not very large.

## 78.4. Problem

Applications that communicate with external applications interchange sensitive data that may be read by unauthorized users while they are in transit. Clearly, if we send sensitive information, we are exposing confidential information and we may be risking the privacy of many individuals. How can we protect messages from being read by intruders?

The solution to this problem must resolve the following forces:

- Confidentiality. Messages may be captured while they are in transit, so we need to prevent unauthorized users from reading them by hiding the information in the message.
- Convenient reception. The hidden information should be revealed conveniently to the receiver.
- Protocol. We need to apply the solution properly, or it will not be able to withstand attacks (there are several ways to attack a method of hiding information).
- Performance. The time to hide and recover the message should be acceptable.
- Security. In some cases, we need to have a very high level of security.

## 78.5. Solution

We can prevent unauthorized users from reading messages by hiding the information in the message using symmetric cryptographic encryption. Symmetric encryption transforms a message in such a way that it can only be understood by the intended receiver after applying the reverse transformation using a valid key. The transformation process at the sender's end is called encryption, while the reverse transformation process at the receiver's end is called decryption.

The sender applies an encryption function (E) to the message (M) using a key (k); the output is the cipher text (C):

$$C = E_k(M)$$

When the cipher text (C) is delivered, the receiver applies a decryption function (D) to the cipher text using the same key (k) and recovers the message:

$$M = D_k(C)$$

## 78.6. Structure

Figure 168 shows the class diagram for the SYMMETRIC ENCRYPTION pattern. A Principal may be a user or an organization that is responsible for sending or receiving messages. This Principal may have the roles of Sender or Receiver. A Sender may send a Message and/or an EncryptedMessage to a Receiver with which it shares a secret Key.

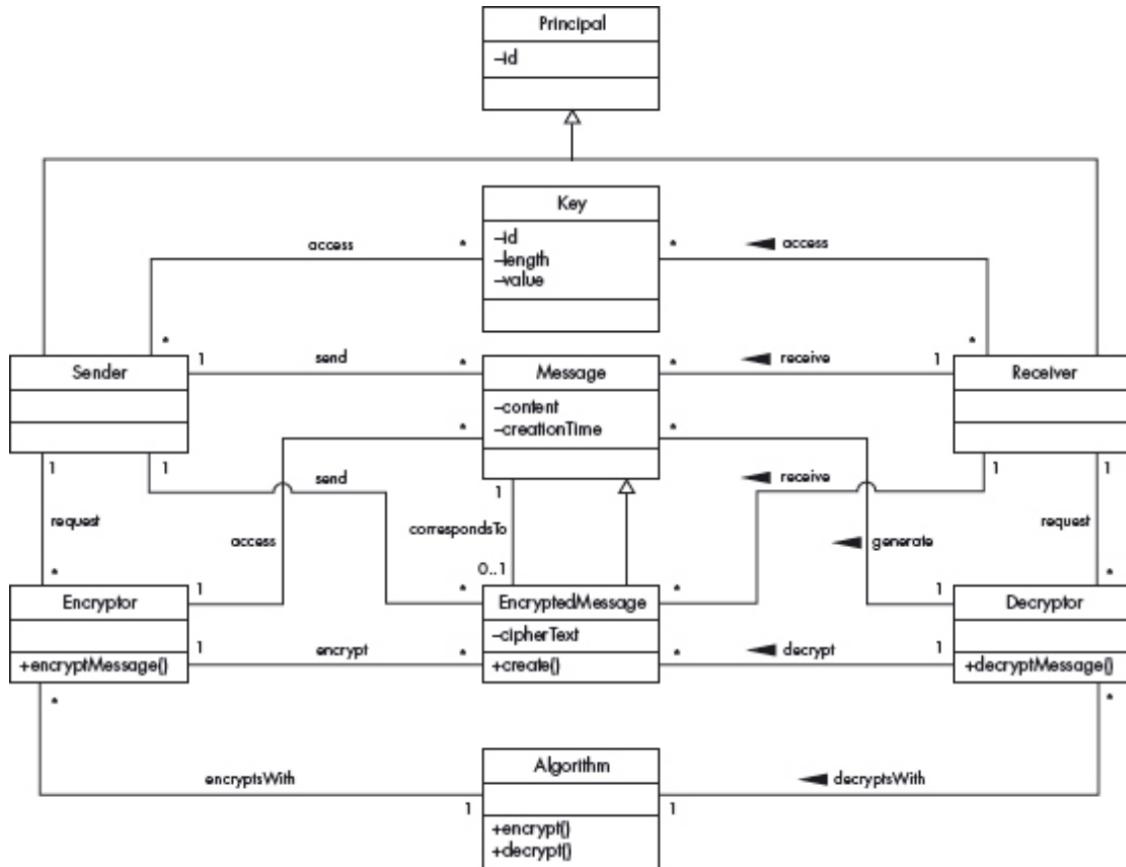


Figure 168: Class diagram for SYMMETRIC ENCRYPTION pattern

The Encryptor creates the EncryptedMessage that contain the cipher text using the shared Key provided by the sender, while the Decryptor deciphers the encrypted data into its original form using the same Key. Both the Encryptor and Decryptor use the same Algorithm to encipher and decipher a message.

## 78.7. Dynamics

We describe the dynamic aspects of the SYMMETRIC ENCRYPTION pattern using sequence diagrams for the use cases ‘Encrypt a message’ and ‘Decrypt a message’.

Use Case:	Encrypt a Message – Figure 169
Summary	A Sender wants to encrypt a message.
Actors	A Sender.
Precondition	Both Sender and Receiver have a shared key and access to a repository of algorithms. The message has already been created by the Sender.
Description	<ol style="list-style-type: none"> <li>1. A Sender sends the message, the shared key, and the algorithm identifier to the Encryptor.</li> <li>2. The Encryptor ciphers the message using the algorithm specified by the Sender.</li> <li>3. The Encryptor creates the EncryptedMessage that includes the cipher text.</li> </ol>
Postcondition	The message has been encrypted and is ready to send.

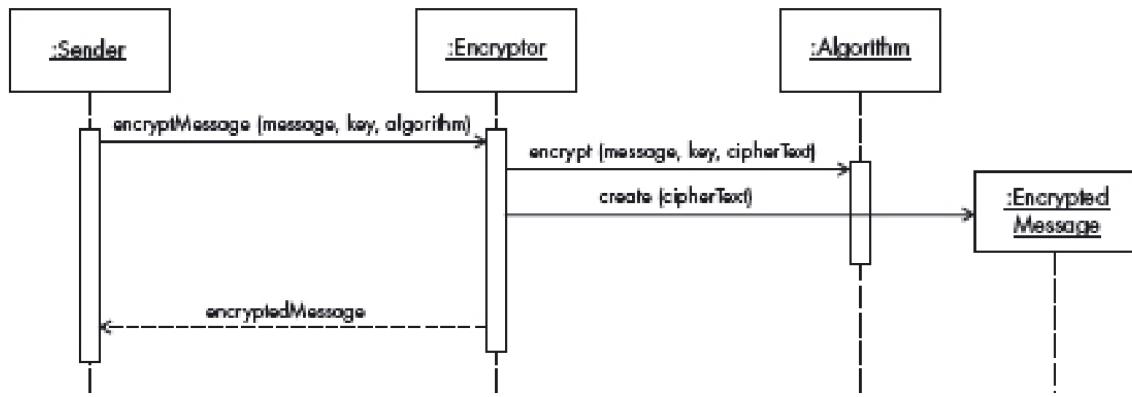


Figure 169: Sequence diagram for the use case 'Encrypt a message'

Use Case:	Decrypt an Encrypted Message – Figure 170
Summary	A Receiver wants to decrypt an encrypted message from a Sender.
Actors	A Receiver.
Precondition	Both the Sender and Receiver have a shared key and access to a repository of algorithms.
Description	<ol style="list-style-type: none"> <li>1. A Receiver sends the encrypted message and the shared key to the Decryptor.</li> <li>2. The Decryptor deciphers the encrypted message using the shared key.</li> <li>3. The Decryptor creates the Message that contains the plain text obtained from the previous step.</li> <li>4. The Decryptor sends the plain text Message to the receiver.</li> </ol> <ul style="list-style-type: none"> <li>• If the key used in step 2 is not the same as the one used for encryption, the decryption process fails.</li> </ul>
Alternate Flow	
Postcondition	The encrypted message has been deciphered and delivered to the Receiver.

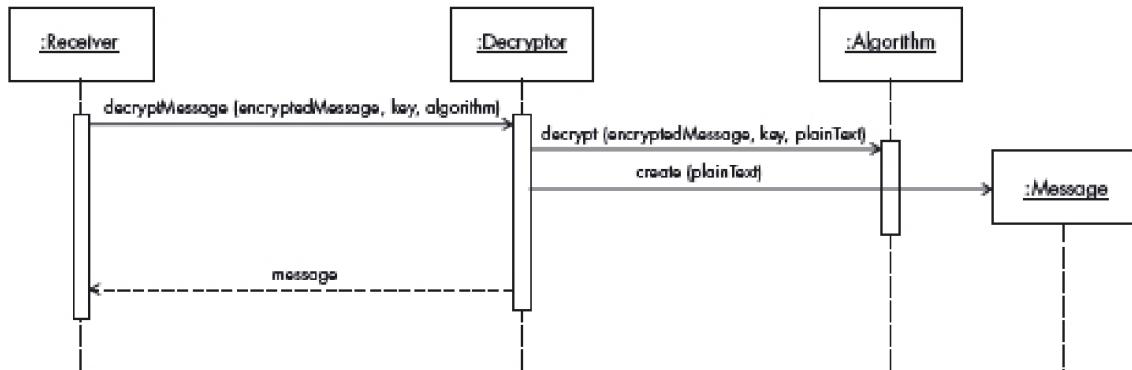


Figure 170: Sequence diagram for the use case 'Decrypt an encrypted message'

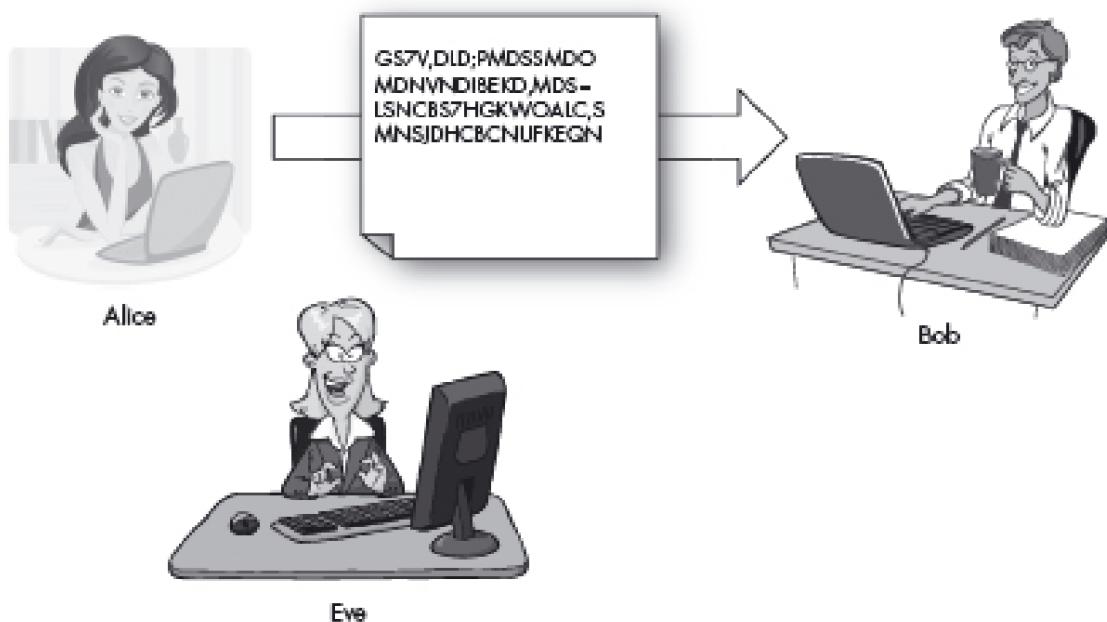
## 78.8. Implementation

- Use the Strategy pattern [1] to select different encryption algorithms. Selection could be based on speed, computational resources, key length, or memory constraints. The selection could happen when instantiating the pattern in an application, or dynamically according to environmental parameters.

- The designer should choose well-known algorithms such as AES (Advanced Encryption Standard) [2] and DES (Data Encryption Standard) [3]. Books such as [4] describe their features and criteria for selection.
- Encryption can be implemented in different applications, such as in e-mail communication, distribution of documents over the Internet, or web services. In these applications we may need to encrypt an entire document or just its body. However, in web services we may want to encrypt specific elements of a message.
- Both the sender and the receiver have to previously agree what cryptographic algorithms they support, and they both must have the same key. This is the key distribution problem, which can be handled in several ways.
- A key management strategy is needed, including key generator, storage, and distribution. This strategy should generate keys that are as random as possible, or an attacker who captures some messages might be able to deduce the key. The key should be properly protected, or an attacker who penetrates the operating system might be able to get it. Timely and secure key distribution is obviously very important.
- A long encryption key should be used (at least 64 bits). Only brute force is known to work against the DES and AES algorithms, for example: using a short key would let an attacker generate all possible keys. Of course, this might change, and the repertoire of algorithms may need to be updated.

## 78.9. Example Resolved

Alice now encrypts the purchase orders she sends to Bob. The purchase order's sensitive data is now unreadable by Eve. Eve can try to apply to it all possible keys, but if the algorithm has been well-chosen and well implemented, Eve cannot read the confidential information. Since Alice only needs to communicate with a few people within the company, key distribution is rather easy.



## 78.10. Consequences

The SYMMETRIC ENCRYPTION pattern offers the following benefits:

- Only receivers who possess the shared key can decrypt a message, transforming it into a readable form. A captured message is unreadable to the attacker. This also makes attacks based on modifying a message very hard.
- The strength of a cryptosystem is based on the secrecy of a long key [4]. The cryptographic algorithms are publicly known, so the key should be kept protected from unauthorized users.
- It is possible to select from several encryption algorithms the one suitable for the application's needs.
- Encryption algorithms that take an acceptable time to encrypt messages exist.

The pattern also has the following potential liabilities:

- The pattern assumes that the shared key is distributed in a secure way. This may not be easy for large groups of nodes exchanging messages. Asymmetric cryptography can be used to solve this problem.
- Cryptographic operations are computationally intensive and may affect the performance of the application. This is particularly important for mobile devices.
- Encryption does not provide data integrity. The encrypted data can be modified by an attacker: other means, such as hashing, are needed to verify that the message was not changed.
- Encryption does not prevent a replay attack, because an encrypted message can be captured and resent without being decrypted. It is better to use another security mechanism, such as time stamps or Nonces, to prevent this attack.

## 78.11. Known Uses

SYMMETRIC ENCRYPTION has been widely used in different products.

- GNuPG [5] is free software that secures data from eavesdroppers.
- OpenSSL [6] is an open-source toolkit that encrypts and decrypts files.
- Java Cryptographic Extension [7] provides a framework and implementations for encryption.
- The .NET framework [8] provides several classes to perform encryption and decryption using symmetric algorithms.
- XML Encryption [9] is one of the foundation web services security standards that defines the structure and process of encryption for XML messages.
- Pretty Good Privacy (PGP), a set of programs used mostly for e-mail security, includes methods for symmetric encryption and decryption [10].

## 78.12. See Also

- The Strategy pattern [1] describes how to separate the implementation of related algorithms from the selection of one of them. This pattern can be used to select an encryption algorithm dynamically.
- ASYMMETRIC ENCRYPTION is commonly used to distribute keys.

## **78.13. References**

- [1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [2] Federal Information Processing Standards Publication. (2001). Advanced Encryption Standard (AES). <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>
- [3] Federal Information Processing Standards Publication. (1999). Data Encryption Data (DES). <http://csrc.nist.gov/publications/fips/fips46- 3/fips46-3.pdf>
- [4] Stallings, W. (2006). Cryptography and Network Security (4th edition). Pearson Prentice Hall
- [5] GnuPG. (n.d.). The GNU Privacy Guard. <http://www.gnupg.org/>
- [6] OpenSSL. (n.d.). The OpenSSL Project. <http://www.openssl.org/>
- [7] Sun Microsystems Inc. (n.d.). Java Cryptography Extension (JCE) <http://java.sun.com/j2se/1.4.2/docs/guide/security/ice/JCERefGuide.html>
- [8] Microsoft Corporation. (2007, November). .NET Framework Class Library. <http://msdn.microsoft.com/enus/library/ms229335.aspx>
- [9] W3C. (2002, December 10). XML Encryption Syntax and Processing. <http://www.w3.org/TR/xmlenc-core/>
- [10] Wikipedia contributors. (n.d.). Pretty Good Privacy. Wikipedia. [https://en.wikipedia.org/wiki/Pretty\\_Good\\_Privacy](https://en.wikipedia.org/wiki/Pretty_Good_Privacy)

## **78.14. Source**

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 79. Virtual Machine Operating System Architecture

## 79.1. Intent

The VIRTUAL MACHINE OPERATING SYSTEM ARCHITECTURE pattern describes how to provide a set of replicas of the hardware architecture (virtual machines) that can be used to execute multiple and possibly different operating systems with strong isolation between them.

## 79.2. Example

A web server is hosting applications for two competing companies. These companies use different operating systems. We want to ensure that neither of them can access the other company's files or launch attacks against the other system.

## 79.3. Context

Mutually suspicious sets of applications that need to execute in the same hardware. Each set requires isolation from the other sets.

## 79.4. Problem

Sometimes we need to execute different operating systems on the same hardware. How can we keep those operating systems isolated in such a way that their executions don't interfere with each other? The solution to this problem must resolve the following forces:

- Each operating system needs to have access to a complete set of hardware features to support its execution.
- Each operating system has its own set of machine-dependent features, such as interrupt handlers. In other words, each operating system uses the hardware in different ways.
- When an operating system crashes or it is penetrated by a hacker, the effects of this situation should not propagate to other operating systems running on the same hardware.
- There should be no way for a malicious user in one virtual machine to get access to the data or functions of another virtual machine.

## 79.5. Solution

Define an architectural layer that is in control of the hardware and supervises and coordinates the execution of each operating system environment. This extra layer, usually called a virtual machine monitor (VMM) or hypervisor, presents to each operating system a replica of the hardware. The VMM intercepts all system calls and interprets them according to the operating system from which they came.

## 79.6. Structure

Figure 171 shows a class diagram for the VIRTUAL MACHINE OPERATING SYSTEM ARCHITECTURE (VMOS) pattern. The VMOS contains one VirtualMachineMonitor (VMM) and multiple virtual machines (VM). Each VM can run a local operating system (LocalOS). The VirtualMachineMonitor supports each LocalOS and is able to interpret its system calls. As a LocalProcess runs on a LocalOS, the VM passes the operating system calls to the VMM, which executes them in the hardware.

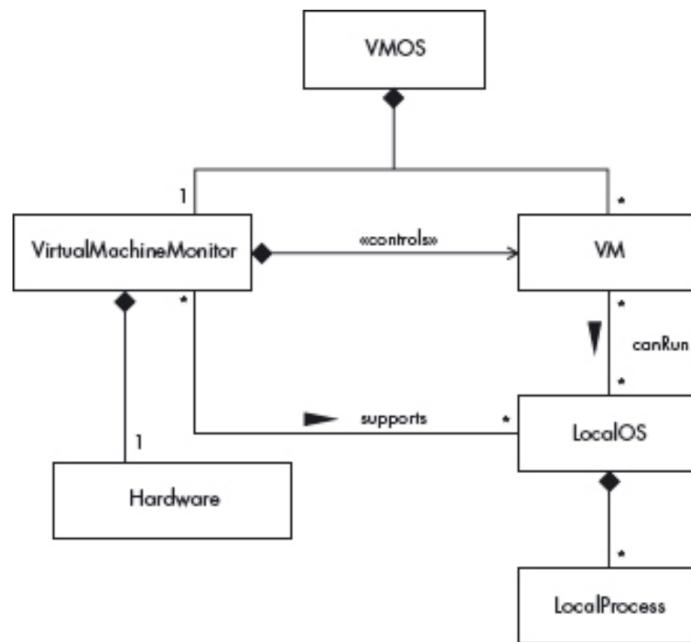


Figure 171: Class diagram for the VIRTUAL MACHINE OPERATING SYSTEM ARCHITECTURE pattern

## 79.7. Dynamics

Figure 172 shows the sequence diagram for the use case ‘Perform an OS call on a virtual machine’. A local process wishing to perform a system operation uses the following sequence:

1. A LocalProcess makes an operating system call to the LocalOS.
2. The LocalOS maps the operating system call to the VMM (by executing a privileged operation).
3. The VMM interprets the call according to the local operating system from which it came, and it executes the operation in hardware.
4. The VMM sends return codes to the LocalOS to indicate successful instruction execution, as well as the results of the instruction execution.
5. The LocalOS sends the return code and data to the LocalProcess.

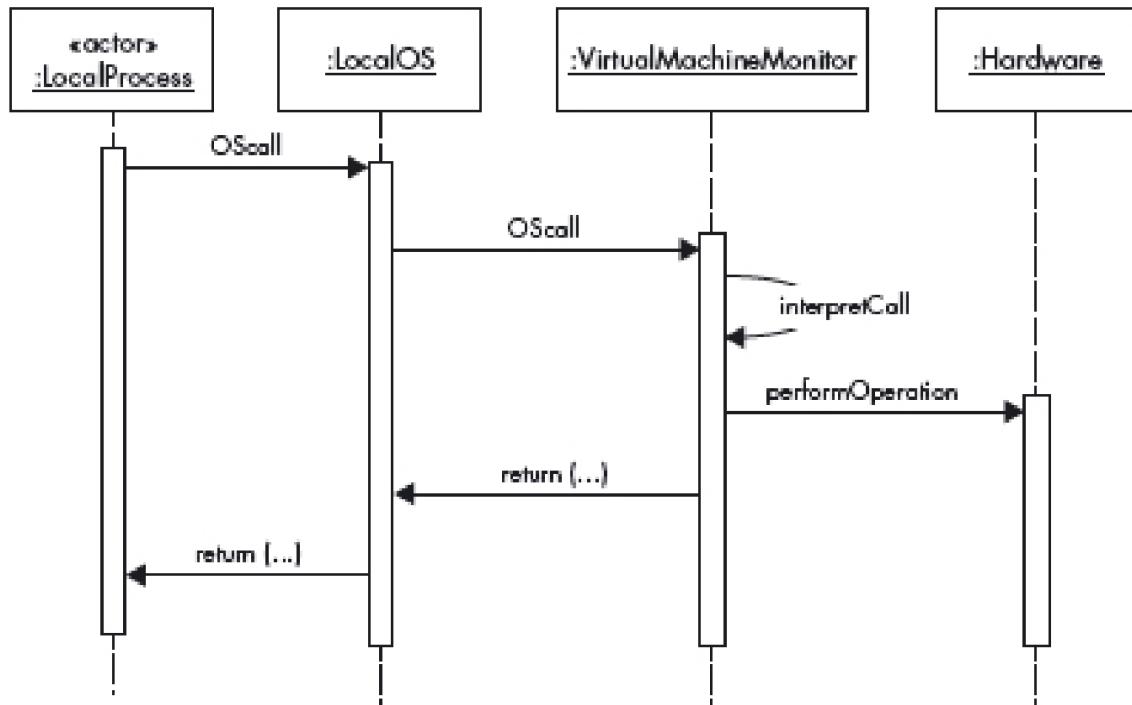


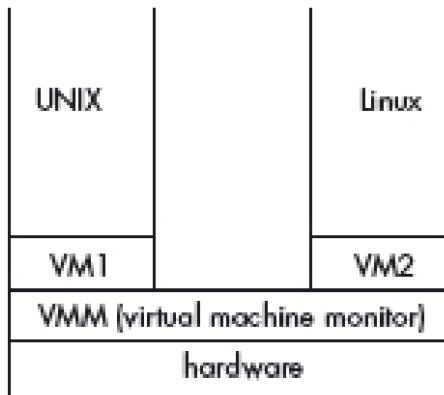
Figure 172: Sequence diagram for the use case ‘Perform an OS call on virtual machine’

## 79.8. Implementation

1. Select the hardware that will be virtualized. All of its privileged instructions must trap when executed in user mode (this is the usual way to intercept system calls).
2. Define a representation (data structure) for describing operating system features that map to hardware aspects, such as meaning of interrupts, disk space distribution, and so on, and build tables for each operating system to be supported.
3. Enumerate the system calls for each supported operating system and associate them with specific hardware instructions.

## 79.9. Example Resolved

In the example shown in Figure 173, two companies using UNIX and Linux can execute their applications in different virtual machines. The VMM provides strong isolation between these two execution environments.



*Figure 173: Virtual Machine operating system example*

## 79.10. Consequences

The VIRTUAL MACHINE OPERATING SYSTEM ARCHITECTURE pattern offers the following benefits:

- The VMM intercepts and checks all system calls. The VMM is in effect a reference monitor and provides total mediation for the use of the hardware. This can provide strong isolation between virtual machines [1].
- Each environment (virtual machine) does not know about the other virtual machine(s), this helps prevent cross-VM attacks.
- There is a well-defined interface between the VMM and the virtual machines.
- The VMM is small and simple and can be checked for security.
- The architecture defined by this pattern is orthogonal to the other three architectures discussed earlier and can execute any of them as local operating systems.

The pattern also has the following potential liabilities:

- All the virtual machines are treated equally. If virtual machines with different security categories are required, it is necessary to build specialized versions. This approach is followed in KVM/370 (see Variants).
- Extra overhead in the use of privileged instructions.
- It is complex to let virtual machines communicate with each other if this is needed.

## 79.11. Variants

- The architecture defined by this pattern is orthogonal to the other three architectures discussed earlier and can execute any of them as local operating systems.
- KVM/370 was a secure extension of VM/370 [2]. This system included a formally verified security kernel, and its virtual machines executed in different security levels, for example top secret, confidential, and so on. In addition to the isolation provided by the VMM, this system also applied the multilevel security model.

## 79.12. Known Uses

- IBM VM/370 [3]. This was the first VMOS and provided virtual machines for an IBM 370 mainframe.
- VMware [4]. This is a current range of products that provide virtual machines for Intel X86 hardware.
- Solaris 10 [5] calls the virtual machines ‘containers’, and one or more applications execute in each container.
- Connectix [6] produces virtual PCs to run Windows and other operating systems.
- Xen is a VMM for the Intel x86 developed as a project at the University of Cambridge, UK [7].
- Some smart phone operating systems use virtual machines to separate users’ private system from their work environment. These include the L4 Microvisor and RIM’s BlackBerry 10 OS [8].

## 79.13. See Also

- REFERENCE MONITOR. The VMM is a concrete version of a reference monitor.
- The operating system patterns described can be used to implement the structure of a VMOS architecture.

## 79.14. References

- [1] Rosenblum, M., & Garfinkel, T. (2005). Virtual machine monitors: Current technology and future trends. *Computer*, 38(5), 39-47.
- [2] Gold, B. D., Linde, R. R., Peeler, R. J., Schaefer, M., Scheid, J. F., & Ward, P. D. (1979, June). A security retrofit of VM/370. In *1979 International Workshop on Managing Requirements Knowledge (MARK)* (pp. 335-344). IEEE.
- [3] Creasy, R. J. (1981). The origin of the VM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5), 483-490.
- [4] Nieh, J., & Leonard, O. C. (2000). Examining vmware. *Dr. Dobb's Journal*, 25(8), 70.
- [5] Trusted Solaris Operating System. (n.d.).  
<http://www.sun.com/software/solaris/trustedsolaris/>
- [6] Connectix Corporation. (n.d.). The Technology of Virtual Machines. white paper. San Mateo. CA. <http://www.connectix.com>
- [7] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., ... & Warfield, A. (2003). Xen and the art of virtualization. *ACM SIGOPS operating systems review*, 37(5), 164-177.
- [8] Wikipedia contributors. (n.d.). QNX. Wikipedia. <https://en.wikipedia.org/wiki/QNX>

## 79.15. Source

Fernandez-Buglioni, E. (2013). *Security patterns in practice: designing secure architectures using software patterns*. John Wiley & Sons.

# 80. Front Door

## 80.1. Intent

Web applications and services often need to identify a user and keep track of a user's session. Integrating several such services allows a single log-in and session context to be provided. A reverse proxy is an ideal point to implement authentication and authorization, by implementing a Web entry server for your back-ends. A sophisticated reverse proxy can even access external back-ends, providing the user's id and password automatically from a 'password wallet.'

## 80.2. Example

Let us continue with the Myshop.com example. Soon after the INTEGRATION REVERSE PROXY was deployed, users complained that they had to re-enter their identity several times on the Web site. Myshop.com's IT personnel recognized that each Web application carried its own user database. Adding an application that required user authentication only meant adding another user data base. Providing support services to their customers and resellers via the Web required more sophisticated authentication, and they wanted to allow access only to those users who paid for the service. See figure 174.

How can Myshop.com provide access control to their Web applications easily, without requiring users to sign on several times, and with support for extensibility?

In addition, the CIO recognizes that new means of user authentication can become popular in the future, so doesn't want the different applications to depend on a single authentication schema. For example, Myshop.com might give security tokens that generate one-time passwords to their resellers, to add a more secure authentication for users who place bulk orders.

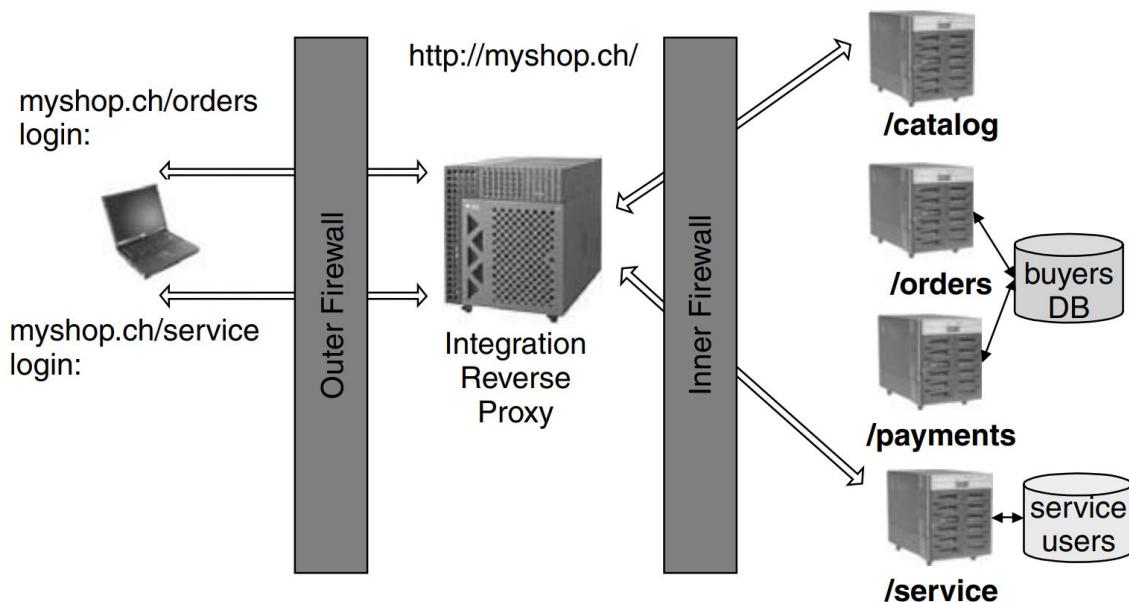


Figure 174: Supporting multiple databases through an INTEGRATION REVERSE PROXY

### **80.3. Context**

A Web site consisting of multiple Web applications that require user authentication.

An INTEGRATION REVERSE PROXY where applications need to authenticate users, and where only authenticated users are authorized to access a defined subset of applications, a PROTECTION REVERSE PROXY where user authentication is required, and where only authenticated users will get access to the underlying Web application, or a combination of both.

### **80.4. Problem**

How do you provide a single sign on for several Web applications or services? The solution to this problem must resolve the following forces:

- You want a single user identity for all applications, even when existing Web applications already carry their own user data base.
- You do not want users to have to provide their password for each application separately, depending on your security policy.
- You might want to force users to identify themselves several times, to avoid misuse of a user's session that is left alone for some time.
- You want your applications to be independent of the authentication schema used. Depending on your security policy, you might even require different schema. For example, strong authentication with a one-time password from a security token for payment service, or weak authentication using a regular id-password combination for service access.
- Different users have different access rights to your systems. You want to be able to handle these differences with a single solution.
- You want new applications to easily integrate into your authentication– authorization schema.
- You want both a single sign-on and a single log-off. That means that a user should keep his session as long as he is active, regardless of the concrete backend he interacts with. On the other hand, when a user logs off, his session should be terminated, so that even when the browser is left open, nobody else can connect to the back-end servers without re-authentication.

### **80.5. Solution**

Implement a FRONT DOOR server as a specialization of the INTEGRATION REVERSE PROXY that identifies users and keeps track of user sessions. This server passes user identity and session identification to all of the back-ends. The FRONT DOOR can log all user activity in a central log. Depending on the nature of the complete solution, some back-end servers might be accessible by everyone, and the FRONT DOOR only protects some back-end servers from unauthenticated users. Nevertheless, remember that it can also act as a PROTECTION REVERSE PROXY for the public part of the Web site.

You need to consolidate user identities held in existing back-end applications. Store the resulting user profiles by combining a user's identities and access rights in a single user

directory. Currently an LDAP directory server is the popular solution for that, but another kind of data base might also be appropriate.

A system for managing user identities and access rights is beyond the scope of this pattern but is often required. In large solutions that use a vendor's solution for access rights, management can be effective, or you might be able to extend an Active Directory when you are using Windows.

## 80.6. Structure

You end up with the following structure if you apply FRONT DOOR to the scenario from our example. The usual place for the machine hosting the FRONT DOOR service is the DEMILITARIZED ZONE. See figure 175.

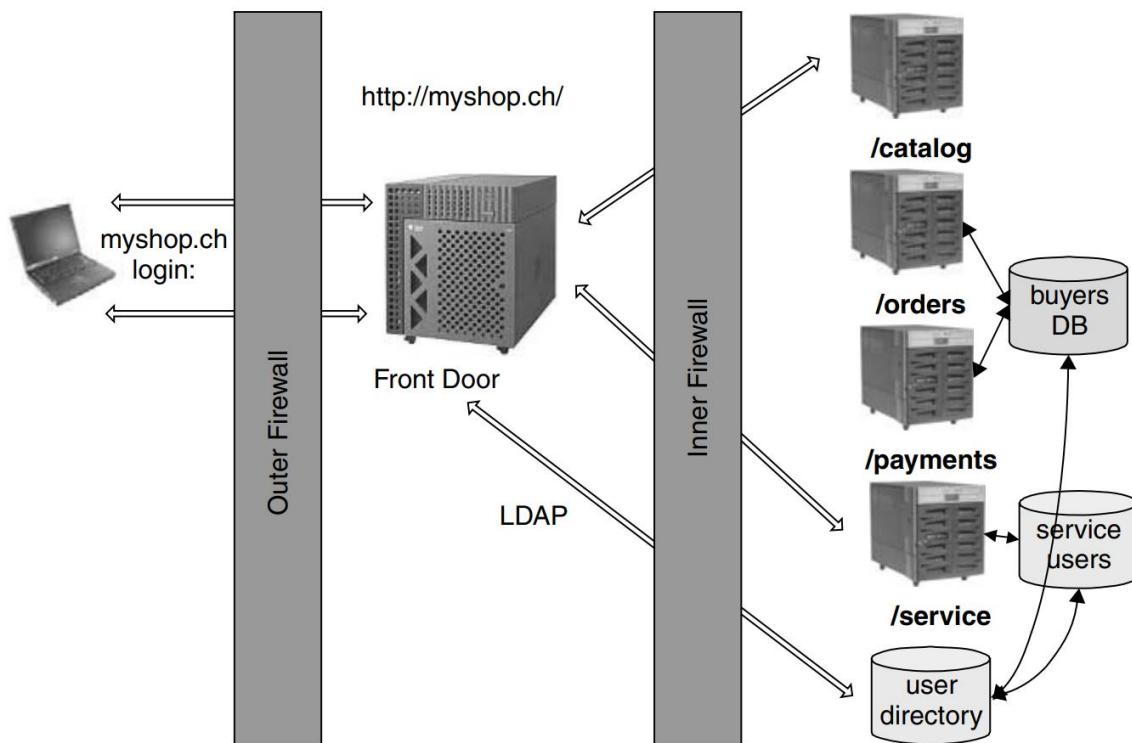


Figure 175: Adding a FRONT DOOR

## 80.7. Dynamics

In addition to INTEGRATION REVERSE PROXY mapping to back-end servers, FRONT DOOR adds another pre-processing step by first checking a user's permissions. Depending on the result of this check, the request is either routed to the desired back-end server, to a log-in page for a user to authenticate themselves, or, in the case of access denial, to an error page. As with PROTECTION REVERSE PROXY, FRONT DOOR might choose to silently drop unauthorized requests and just log the access attempts.

## 80.8. Implementation

To implement a FRONT DOOR reverse proxy the following must be considered in addition to the issues given in the preceding patterns:

1. Unify user representations and data base. This is easiest if you have a clean start, or if only one user database exists. An LDAP directory server is a popular means for storing user identities, passwords, and access rights. If you want to integrate existing back-ends that you cannot change, you might need to add an identity and password wallet to each user object in the directory, to enable automatic replay of id and password when accessing such a back-end.
2. Define authentication mechanism(s). Popular mechanisms are id-password, one-time passwords, one-time token-based password, challenge-response with token, biometrics, certificates, or any useful combination of these. Since now only the FRONT DOOR needs to implement user authentication, it is easy to change or extend authentication mechanisms later without any impact on existing applications.
3. Define access rights schema if needed. Different approaches exist for representing access rights and the mapping of users to the set of allowed services. For the purpose of the FRONT DOOR, a coarse-grained model is sufficient, but individual applications might need fine-grained control to internal functionality. A sophisticated implementation will provide a complete model applicable not only for the FRONT DOOR, but also for all an application's needs.
4. Design user and session representation as passed to back-ends via header fields. This can be a specifically named header field, or you can use HTTP's basic authentication mechanism to pass on the user identity. If there is no single user representation, FRONT DOOR might need to map the user id to the one specific to the back end. This mapping needs to be stored in the user data base as designed in step 1. Optionally define additional header fields for inter-back-end communication. Those header fields are analyzed by FRONT DOOR, kept in its session store, and automatically passed to all interested back-ends.

This and the following three steps correspond roughly to activities described in SECURITY SESSION Implementation section.

5. Design and implement how FRONT DOOR keeps track of user sessions. Some solutions that rely on SSL for browser-FRONT DOOR communication use the SSL session id for that purpose. Using a session cookie is also popular. Rewriting all URLs to add a session id in the content of back-end replies, if cookies are disabled, seems too great an overhead and too complex. Using cookies, it is even possible to be able to keep the session context when switching between HTTP and HTTPS, either for performance or security reasons. FRONT DOOR session cookies should be encrypted and cryptographically signed to ensure that they cannot be manipulated. If FRONT DOOR cookies are secured like this, and they also contain some identification of their source, FRONT DOOR can even accept such a cookie as a valid user identification after a crash without the user being aware of the session's re-authentication.
6. Design and implement FRONT DOOR session context. The session cookie can be the means to store all session context. However, because of a cookie's size limitation and security issues, it might be better to keep the session context on the server side. One solution is to keep a session list with all session contexts in memory. This is the most efficient solution, especially if the access rights of a user are also cached there, but it carries the risk of losing session state on a crash. Another option is to use persistent storage in a data base for session context. However, this tends to be an order of magnitude slower, but allows for several FRONT DOOR instances to share session context. Which solution for keeping session context is best depends on the concrete

requirements. For more explanations for keeping session state, refer to the chapter Session State Patterns in [1].

7. Implement a cookie jar. If back-end servers use their own session cookies, FRONT DOOR can keep those session cookies in its own session context and not pass them to the user's browser. This ensures single log-off. If this is not done, a browser might send an old application session cookie after a new user logged into FRONT DOOR, confusing the back-end server.
8. Design and implement log-in and portal pages. FRONT DOOR can delegate user identification to a special back-end server, or it can implement its own log-in page. As with INTEGRATION REVERSE PROXY, a portal page that consists of a menu of all services available to the logged-in user is a possible poor-man's portal solution. In addition, a special service link (for example, /logoff) should be implemented by FRONT DOOR to allow applications to give the user the ability to consciously terminate their session.  
Apart from its visual nature, this corresponds to CHECK POINT in a FRONT DOOR user's experience.

## 80.9. Example Resolved

## 80.10. Consequences

In addition to the consequences of PROTECTION REVERSE PROXY and INTEGRATION REVERSE PROXY, this pattern implies the following benefits:

- Single sign-on and single log-off because FRONT DOOR keeps track of a user's session, and back-ends automatically obtain the user id from FRONT DOOR instead of asking the user for it again.
- One user profile is possible across back-end applications. This is not necessarily the case, for example if you start with several existing Web applications and integrate them, but FRONT DOOR facilitates the mechanisms that enable you to move implement a solution that uses one user profile and one administration application.
- Applications are relieved from implementing access control and user authentication. This gives you the opportunity to deploy Web applications quickly that readily integrate with FRONT DOOR access control. Experience shows that such an architectural guidance for Web applications can be a great benefit, especially on an intranet.
- Centralized logging allows user tracking and reporting. Marketing departments might die for such logs, which keep a detailed track of user activity.

However, in combination with the previous patterns' liabilities, FRONT DOOR carries the following additional liabilities:

- Applications might enforce their own user database, increasing the risk of inconsistencies. For example, RSA's ACE/Server has its own user database for managing tokens for its strong authentication. If you implement FRONT DOOR using both RSA SecureID and another user authentication schema, you end up with two user databases you need to synchronize.

- A central management application for user identities and access rights is needed. Without a single sign-on, this need can exist already, but might not be recognized. Deploying FRONT DOOR makes this need prominent. Also, a lack of corresponding organizational processes is more easily shown up.
- Password aging policies across back-end applications can conflict. You then need to auto-generate new passwords when they expire, or let the user worry about changing their password on the back-end application and in their FRONT DOOR profile.
- Conflicting session time-outs of FRONT DOOR and applications can confuse users.

## 80.11. Variants

As with INTEGRATION REVERSE PROXY you can deploy two FRONT DOOR that share back-end servers, one for the Internet (effectively making it an extranet) and one for intranet users.

## 80.12. Known Uses

Peter Sommerlad's former company's Frontdoor solution for Telekurs Financial Services Ltd. implements most of the issues given here, in addition to being able to be configured as a PROTECTION REVERSE PROXY.

Bull Evidian PortalXpert [2] implements a Web Entry Service.

IBM Tivoli Access Manager [3] provides FRONT DOOR reverse proxy functionality with its Web Seal product.

## 80.13. See Also

You can view FRONT DOOR as adding CHECK POINT and SECURITY SESSION to an INTEGRATION REVERSE PROXY or PROTECTION REVERSE PROXY, and thus also providing a SINGLE ACCESS POINT to a company's Web applications and services.

## 80.14. References

[1] Fowler, M. (2002). Patterns of Enterprise Application Architecture.

[2] Evidian. (n.d.). <http://www.evidian.com>

[3] IBM. (n.d.). Enterprise Security Architecture using IBM Tivoli Security Solutions.  
<http://www.redbooks.ibm.com/pubs/pdfs/redbooks/sg246014.pdf>

## 80.15. Source

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# 81. Reference Monitor

## 81.1. Intent

In a computational environment in which users or processes make requests for data or resources, this pattern enforces declared access restrictions when an active entity requests resources. It describes how to define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations.

## 81.2. Example

In the hospital example described in ROLE-BASED ACCESS CONTROL we declared the accesses allowed to doctor and other personnel. However, we expected voluntary compliance with the rules. It has not worked; busy personnel bypass the rules and there is no way of enforcing them.

## 81.3. Context

A computational environment in which users or processes make requests for data or resources.

## 81.4. Problem

If we don't enforce the defined authorizations, it is the same as not having them, users and processes can perform all type of illegal actions. Any user could read any file, for example.

The solution to this problem must resolve the following forces:

- Defining authorization rules is not enough, they must be enforced whenever a user or process makes a request for a resource.
- There are many possible ways of enforcement, depending on the specific architectural unit or level involved. We need an abstract model of enforcement that applies to every level of the system.

## 81.5. Solution

Define an abstract process that intercepts all requests for resources and checks them for compliance with authorizations.

## 81.6. Structure

Figure 176 shows a class diagram that describes a reified REFERENCE MONITOR. In this figure Authorization Rules denotes a collection of authorization rules organized as ACLs or in some other way.

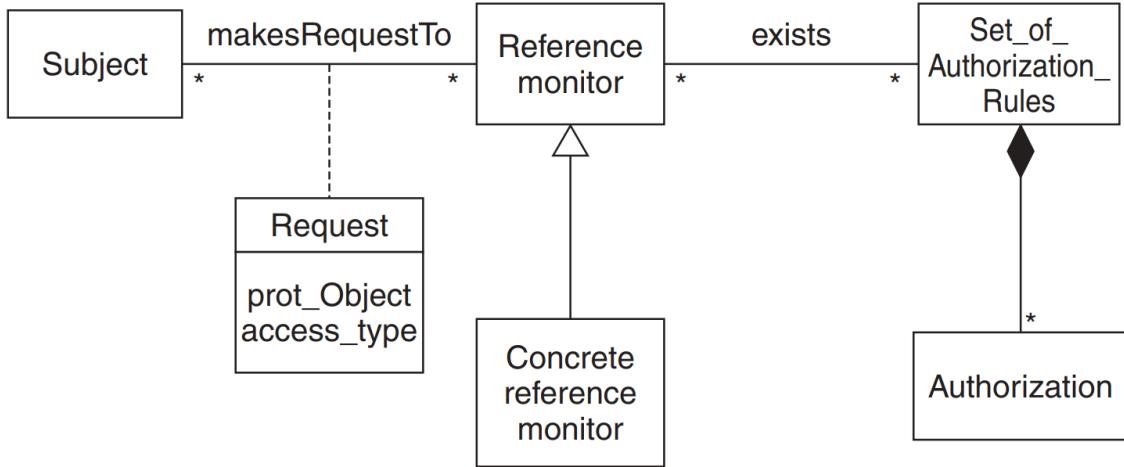


Figure 176: Class diagram for REFERENCE MONITOR

## 81.7. Dynamics

Figure 177 is a sequence diagram showing how a request from a process is checked. The REFERENCE MONITOR looks for the existence of a rule that authorizes the request. If one exists, the request is allowed to proceed.

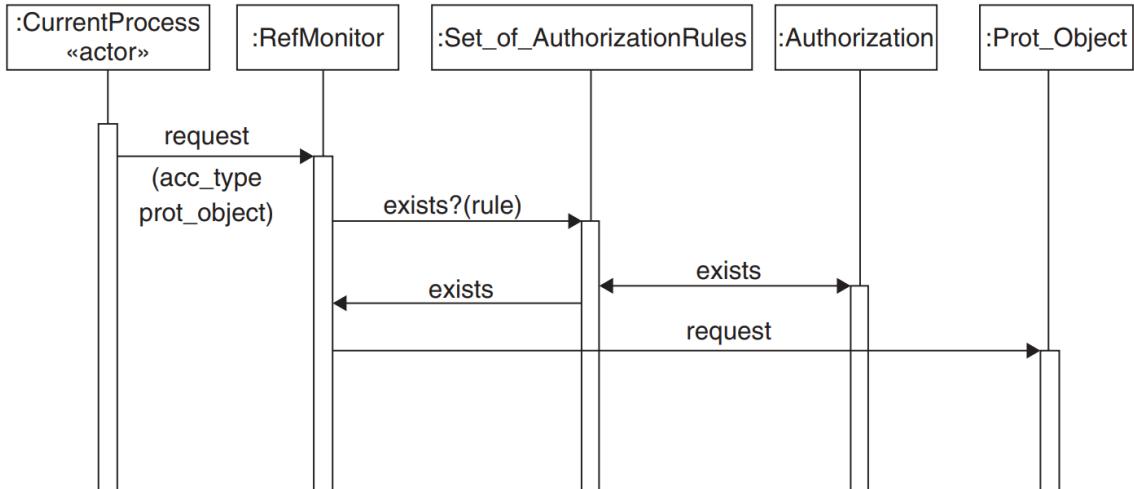


Figure 177: Sequence diagram for enforcing security of requests

## 81.8. Implementation

A concrete reference monitor is required at each section of the system that has resources that can be requested. Examples include a memory manager (to control access to main memory), and a file manager (to control use of files).

## 81.9. Example Resolved

The hospital bought a database system to store patient data. Now, when a user attempts to access patient data, their authorization is checked before giving them access to it. Actions such as read or write are also controlled, for example, only doctors and nurses are allowed to modify patient records.

## **81.10. Consequences**

The following benefits may be expected from applying this pattern:

- If all requests are intercepted, we can make sure that they comply with the rules.
- Implementation has not been constrained by using this abstract process.

The following potential liabilities may arise from applying this pattern:

- Specific implementations (concrete REFERENCE MONITOR) are needed for each type of resource. For example, a file manager is needed to control requests for files.
- Checking each request may result in intolerable performance loss. We may need to perform some checks at compile-time, for example, and not repeat them at execution time.

## **81.11. Known Uses**

Most modern operating systems implement this concept, including Solaris 9, Windows 2000, AIX, and others. The Java Security Manager is another example. Database management systems also have an authorization system that controls access to data requested by queries.

## **81.12. See Also**

This pattern is a special case of CHECK POINT. INTERCEPTOR [1] can act as a REFERENCE MONITOR in some situations. Concrete versions of REFERENCE MONITOR include file control systems and firewall patterns.

## **81.13. References**

[1] Schmidt, D. C., Stal, M., Rohnert, H., & Buschmann, F. (2000). *Pattern-oriented software architecture, patterns for concurrent and networked objects*. John Wiley & Sons.

## **81.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# 82. Controlled Virtual Address Space

## 82.1. Intent

This pattern addresses how to control access by processes to specific areas of their virtual address space (VAS) according to a set of predefined access types. Divide the VAS into segments that correspond to logical units in the programs. Use special words (descriptors) to represent access rights for these segments.

## 82.2. Example

Our operating system improved by using a reference monitor. However, hackers discovered that the unit of access control to memory was coarse. By taking advantage of the lack of precision in controlling access they were able to access other processes' areas.

## 82.3. Context

Multiprogramming systems with a variety of users. Processes executing on behalf of these users must be able to share memory areas in a controlled way. Each process runs in its own address space. The total VAS at a given moment includes the union of the VASs of the individual processes, including user and system processes. Typical allowed accesses are read, write, and execute, although finer typing is possible.

## 82.4. Problem

Processes must be controlled when accessing memory, otherwise they could overwrite each other's memory areas or gain access to private information. While relatively small amounts of data can be directly compromised, illegal access to system areas could allow a process to force a higher execution privilege level and thus access files and other resources.

The solution to this problem must resolve the following forces:

- There is a need for a variety of access rights for each separate logical unit of VAS (segment). In this way security and controlled sharing are possible.
- There is a variety of virtual memory address space structures: some systems use a set of separate address spaces, others a single-level address space. Further, the VAS may be split between the users and the operating system. We would like to control access to all of these types in a uniform manner.
- For any approach to be efficient, hardware assistance is necessary. This implies that an implementation of the solution will require a specific hardware architecture. However, the generic solution must be hardware independent.

## 82.5. Solution

Divide the VAS into segments that correspond to logical units in the programs. Use special words (descriptors) to indicate access rights that show the starting address of the accessible

segment, the limit of the accessible segment, and the type of access permitted (read, write, execute).

## 82.6. Structure

Figure 178 below shows a class diagram for the solution. A process (the Process class) must have a descriptor (the Descriptor class) to access segments in the VAS.

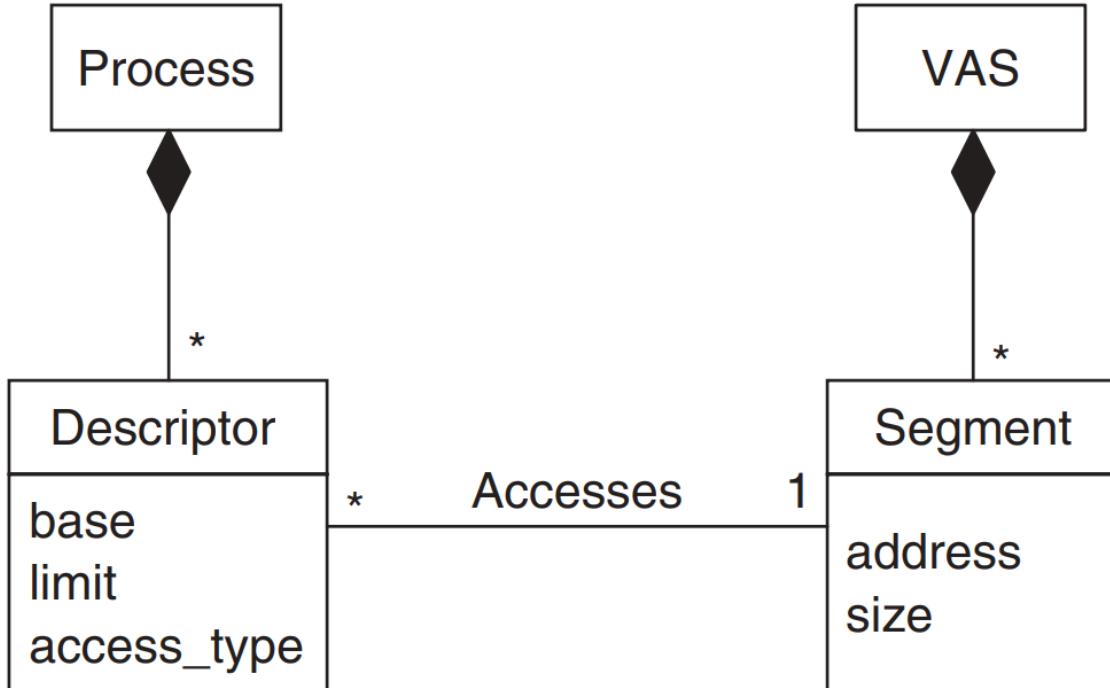


Figure 178: Class diagram for CONTROLLED VIRTUAL ADDRESS SPACE

## 82.7. Dynamics

## 82.8. Implementation

Some implementation aspects include:

- The limit check when accessing an address must be done by the instruction microcode or the overhead would not be acceptable. This check is part of an instance of REFERENCE MONITOR.
- The same idea applies to purely paging systems, except that the limit in the descriptor is defined by the page size. In paged systems pages do not correspond to logical units and cannot perform a fine security control.
- There are two basic ways to implement this pattern:
  - Property descriptor systems. The descriptors are loaded at process creation by the operating system. The descriptors are handled through special registers and disappear at the end of execution.
  - Capability systems. A special trusted portion of the operating system distributes capabilities to programs. Programs own these capabilities. To use

them, the operating system loads them into special registers or memory segments.

In both cases, access to files is derived from their ACLs.

## 82.9. Example Resolved

Descriptors can control areas of memory of any size. A process without a descriptor for an area cannot access it. If sharing is required, several processes can have a descriptor with the same addresses but with different access rights.

## 82.10. Consequences

The following benefits may be expected from applying this pattern:

- The pattern provides the required segment protection because a process cannot access a segment without a descriptor for it. Two processes with descriptors with the same memory address base–limit pair can conveniently share a segment.
- The pattern applies to any type of virtual address space: single, segregated, or split.
- If all resources are mapped to the virtual address space, the pattern can control access to any type of resource, including files.

The following potential liabilities may arise from applying this pattern:

- Segmentation makes storage allocation inefficient because of external fragmentation [1]. In most systems segments are paged for convenient allocation.
- Hardware support is needed, which puts an extra requirement on this solution.
- In systems that use multiple separate address spaces, it is necessary to add an extra identifier to the descriptor registers to indicate the address space number.

## 82.11. Known Uses

The Plessey 250 [2], Multics [3], IBM S/38, IBM S/6000, Intel X86 [4], and Intel Pentium use some type of descriptors for memory access control. The operating systems in these machines must use this approach for memory management. Specific uses include the Choices operating system [5] and AIX [6].

## 82.12. See Also

This pattern is a direct application of AUTHORIZATION to the processes' address space.

## 82.13. References

[1] Silberschatz, A., Galvin, P., Gagne, G. (2003). Operating System Concepts (6<sup>th</sup> Edition). John Wiley & Sons.

[2] Hodges, K. H. (1973). A fault-tolerant multiprocessor design for real-time control. *Computer Design*, 12(12), 75-81.

- [3] Graham, R. M. (1968). Protection in an information processing utility. *Communications of the ACM*, 11(5), 365-369.
- [4] Childs, R. E., Crawford, J. O. H. N., House, D. L., & Noyce, R. N. (1984). A processor family for personal computers. *Proceedings of the IEEE*, 72(3), 363-376.
- [5] Russo, V. F., & Campbell, R. H. (1989). Virtual memory and backing storage management in multiprocessor operating systems using object-oriented design techniques. *ACM Sigplan Notices*, 24(10), 267-278.
- [6] Camillone, N. A., Steves, D. H., & Witte, K. C. (1990). AIX operating system: A trustworthy computing system. *IBM RISC System/6000 Technology*, 168-172.

## 82.14. Source

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# 83. Controlled Object Factory

## 83.1. Intent

This pattern addresses how to specify the rights of processes with respect to a new object. When a process creates a new object through a factory (see FACTORY METHOD and ABSTRACT FACTORY [1]), the request includes the features of the new object. These features include a list of rights to access the object.

## 83.2. Example

In many operating systems the creator of an object gets all possible rights to the object. Other operating systems apply predefined sets of rights: for example, in Unix all the members of a file owner's group may receive equal rights for a new file. These approaches may result in unnecessary rights being given to some users, violating the principle of least privileges.

## 83.3. Context

A computing system that needs to control access to its created objects because of their different degrees of sensitivity. Rights for these objects are defined by authorization rules or policies that are enforced when a process attempts to access an object.

## 83.4. Problem

In a computing environment, executing applications need to create objects for their work. Some objects are created at program initialization, while others are created dynamically during execution. The access rights of processes with respect to objects must be defined when these objects are created, or there may be opportunities for the processes to misuse them. Applications also need resources such as I/O devices and others that may come from resource pools: when these resources are allocated, the application must be given rights to them.

The solution to this problem must resolve the following forces:

- Applications create objects of many different types, but we need to handle them uniformly with respect to their access rights, otherwise it would be difficult to apply standard security policies.
- We need to allow objects in a resource pool to be allocated and have their rights set dynamically: not doing so would be too rigid.
- There may be specific policies that define who can access a new object, and we need to apply these when creating the rights for an object. This is a basic aspect of security.

## 83.5. Solution

Whenever a new object is created, define a list of subjects that can access it, and in what way.

## 83.6. Structure

Figure 179 shows the class diagram for the solution. When a Process creates a new object through a Factory, the Creation\_Request includes the features of the new object. Among these features is a list of rights that define the access rights for a Subject to access the created Object. This implies that we need to intercept every access request: this is done by CONTROLLED OBJECT MONITOR.

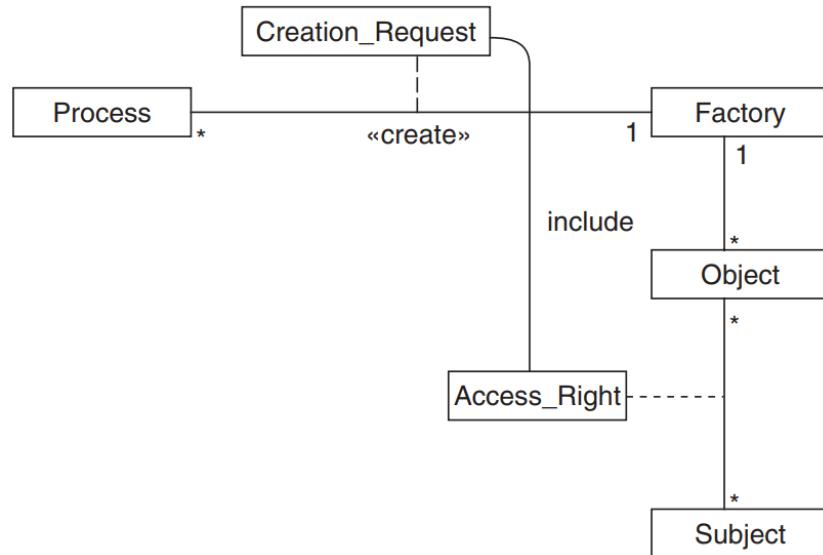


Figure 179: Class diagram for CONTROLLED OBJECT FACTORY

## 83.7. Dynamics

Figure 180 shows the dynamics of object creation. A process creating an object through a FACTORY defines the rights for other subjects with respect to this object.

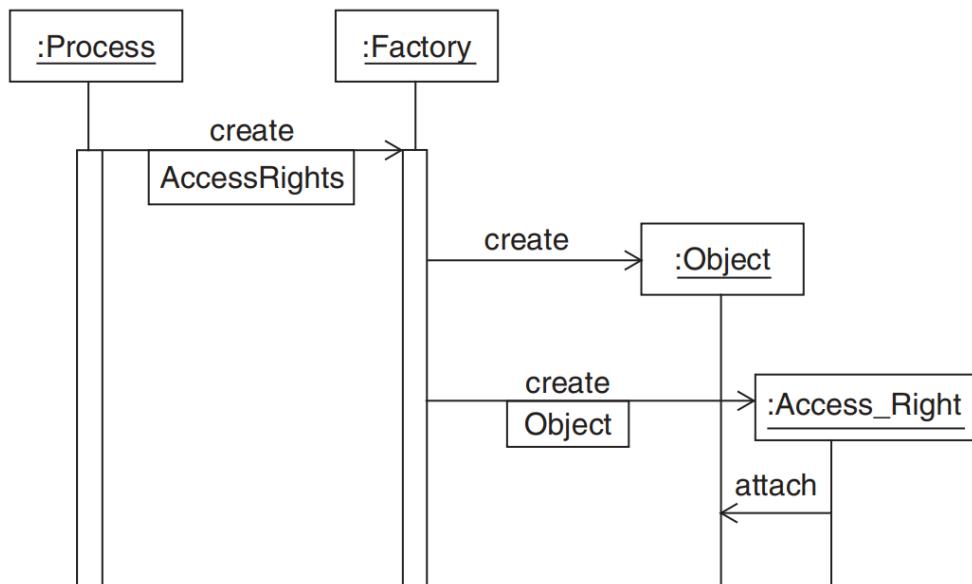


Figure 180: Object creation dynamics

## **83.8. Implementation**

Each object may have an associated access control list (ACL). This will list the rights each user has for the associated object. Each entry specifies the rights that any other object within the system can have. In general, each right can be an ‘allow’ or a ‘deny.’ These are also known as Access Control Entries (ACE) in the Windows environment (see [2,3,4]). The set of access rules is also known as the Access Control List (ACL) in Windows and most operating systems.

Capabilities are an alternative to an ACL. A capability corresponds to a row in an access matrix. This is in contrast to the ACL, which is associated with the object. The capability indicates to the secure object that the subject does indeed have the right to perform the operation. The capability may carry some authentication features in order to show that the object can trust the provided capability information. A global table can contain rows that represent capabilities for each authenticated user [5], or the capability may be implemented as a lists for each user which indicates which object each user has access to. [6]

## **83.9. Example Resolved**

Our users can now be given only the rights to the created objects that they need. This prevents them from having too many (possibly unnecessary) object rights. Many misuses occur through processes having too many rights.

## **83.10. Consequences**

The following benefits may be expected from applying this pattern:

- There will be no objects that have default access rights because somebody forgot to define rights to access them
- It is possible to define access rights to an object based on its sensitivity
- Objects allocated from a resource pool can have rights attached to them dynamically
- The operating system can apply ownership policies: for example, the creator of an object may receive all possible rights to the objects it creates.

The following potential liabilities may arise from applying this pattern:

- There is a process creation overhead
- It may not be clear what initial rights to define

## **83.11. Known Uses**

The Win32 API allows a process to create objects with various Create system calls using a structure that contains access control information (DACL) passed as a reference. When the object is created, the access control information is associated with the object by the kernel. The kernel returns a handle to the caller to be used for access to the object.

## **83.12. See Also**

BUILDER and other creation patterns [1].

### **83.13. References**

- [1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [2] Hart, J. M. (1997). *Win32 systems programming*. Addison-Wesley Longman Publishing Co., Inc.
- [3] Microsoft Corporation. (2000). Windows 2000 Security Technical Reference. Microsoft Press.
- [4] Zachman, J. A. (1987). A framework for information systems architecture. *IBM systems journal*, 26(3), 276-292.
- [5] Anderson, R. (2001). *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons.
- [6] King, C. M., & Dalton, C. (2001). *Security Architecture: Design, Deployment, and Operations*. McGraw-Hill Professional.

### **83.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **84. Execution Domain**

## **84.1. Intent**

Unauthorized processes could destroy or modify information in files or databases, with obvious results, or could interfere with the execution of other processes. Therefore, define an execution environment for processes, indicating explicitly all the resources that a process can use during its execution, as well as the type of access to the resources.

## **84.2. Example**

In our operating system we know now how to assign access rights to processes and how to enforce these rights at execution time. However, a process may have different functions and in each functional mode it may need different rights. For example, if a process needs to read some files to collect some data, this should happen only at the specific time of access to the file, otherwise a hacker could take advantage of the extra rights to perform illegal accesses.

## **84.3. Context**

A process executes on behalf of a user, group, or role (a subject). A process must have access rights to use the resources defined for its subject during execution. The set of access rights given to a process define its execution domain. At times the process may also need to enter other domains to perform its work: for example, for example, to extract data from a file in another user's domain. Frequently, users structure their domains as a hierarchical tree of domains with one root domain.

## **84.4. Problem**

Restricting a process to a specific set of resources is a basic step towards controlling malicious behavior. Otherwise, unauthorized processes could destroy or modify information in files or databases, with obvious results, or could interfere with the execution of other processes.

The solution to this problem must resolve the following forces:

- There is a need to restrict the actions of a process during its execution; otherwise, it could perform illegal actions.
- Resources typically include memory and I/O devices but can also be system data structures and special instructions. Although resources are heterogeneous, we want to treat them uniformly.
- A process needs the flexibility to create multiple domains and to enter inner domains for specific purposes.
- There should be no restrictions on how to implement the domain.

## **84.5. Solution**

Attach a set of descriptors to the process that represent the rights of the process. Collect them into an execution domain. Execution domains can be nested.

## 84.6. Structure

In figure 181, the Domain class represents domains, and, in conjunction with COMPOSITE, it describes nested domains. The operation enter() in Domain lets a process enter a new domain. A domain includes a set of descriptors that define rights for resources.

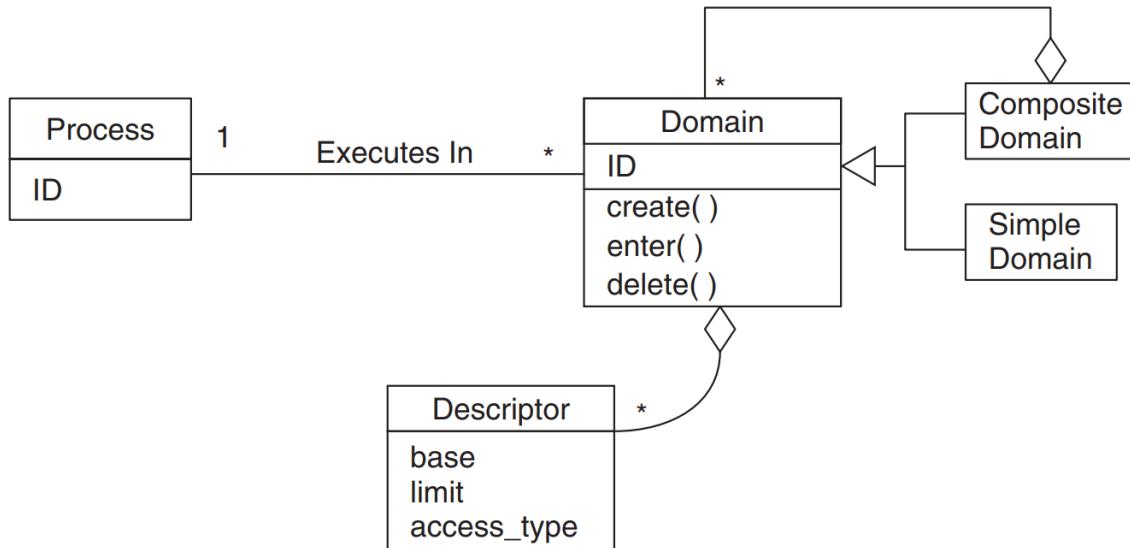


Figure 181: Class diagram for EXECUTION DOMAIN

## 84.7. Dynamics

## 84.8. Implementation

## 84.9. Example Resolved

Using the concept of execution domain, we have a set of well-defined environments with explicit rights. A process can change domains according to its tasks and acquire only the needed rights in each domain.

## 84.10. Consequences

The following benefits may be expected from applying this pattern:

- The pattern lets users apply the principle of least privilege to processes— they can be given only the rights they need to perform their functions.
- It could be applied to describe access to any type of resource if the resource is mapped to a specific memory address.
- Processes may have several execution domains, either peer or nested.
- The model does not restrict the implementation of domains. A domain could be represented in many ways. For example, the Plessey 250, IBM S/38, and IBM S/6000

use capabilities. The Intel X86 and Pentium series and their corresponding operating systems use descriptors for memory access control.

- One can define special domains with predefined rights or types of rights. For example, Multics and the Intel X86 series use Protection Rings, in which each ring is assigned to a type of program, for example Supervisor, Utilities, User programs, and External programs. The rings are hierarchically structured based on their level of trust. Descriptors are used to cross rings in program calls.
- As shown, the descriptors refer to VAS segments, which is the most usual implementation. However, they could indicate resources not mapped to memory.

The following potential liabilities may arise from applying this pattern:

- Extra complexity—special hardware is needed to handle descriptors and set up domains.
- Performance overhead in setting up domains and in entering and leaving domains. Because of this, some operating systems for Intel processors use only two rings, improving performance but reducing security.
- Setting up the execution domain is implementation dependent. In descriptor systems the operating system creates a descriptor segment with the required descriptors. In capability systems the descriptors are part of the process code and are enabled during execution.

## 84.11. Known Uses

The concept of domains comes from Multics [1]. Segments or pages (as in EROS [2]) are structured as tree directories. The Plessey 250 and the IBM S/6000 running AIX [3] are good examples of the use of this pattern. The Java Virtual Machine defines restricted execution environments in a similar way [4].

## 84.12. See Also

CONTROLLED PROCESS CREATOR and CONTROLLED OBJECT MONITOR work in conjunction with this pattern.

## 84.13. References

- [1] Graham, R. M. (1968). Protection in an information processing utility. *Communications of the ACM*, 11(5), 365-369.
- [2] Shapiro, J. S., & Hardy, N. (2002). EROS: A principle-driven operating system from the ground up. *IEEE software*, 19(1), 26-33.
- [3] Camillone, N. A., Steves, D. H., & Witte, K. C. (1990). AIX operating system: A trustworthy computing system. *IBM RISC System/6000 Technology*, 168-172.
- [4] Oaks, S. (2001). *Java security: writing and deploying secure applications (2<sup>nd</sup> Edition)*. O'Reilly Media, Inc.

#### **84.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# 85. Access Control Requirements

## 85.1. Intent

The function of the access control security service is to permit or deny someone the right to perform an action on an asset, such as create, read, modify, or delete a data file. While each situation that calls for access control is unique, there are common generic requirements that apply to all access-control situations. This pattern provides a common generic set of access control requirements. The requirements address both the access control function and the properties of the access control service, such as ease of use and flexibility. The pattern also helps you to apply the general requirements to your specific situation and helps you to determine the relative importance of conflicting requirements.

## 85.2. Example

A new wing of an existing museum of gemstones is to be opened. The wing will house gems of varying value, some of which are owned by the museum and some of which are on loan. Some of the gems are famous stones whose loss would involve much media publicity. The wing will house valuable gems on display, low-value gems in a hands-on exhibit, and gems of all values in working areas of the wing that are not open to the public.

Based on the results of applying ENTERPRISE SECURITY SERVICES, Samuel the museum's system engineer understands that the museum needs to control access to the gems and to the information related to gems. An obvious example is that an attempted access by Theo the thief to steal a gem should be denied. Another example is that the recorded carat weight for gems should not be modifiable by unauthorized people. An unauthorized change in recorded carat weight could change a gem's value, change insurance costs, or even signal the beginning of an attempt to carve off a piece of the stone.

But Samuel also understands that the need to deny unauthorized access must be balanced against the need to permit authorized access. For example, the best safeguard against theft of a gem is to lock it up in a vault and not tell anyone where it is. But this would interfere with a primary goal of the museum, which is to display gems for public viewing. Therefore, Samuel needs to specify a balanced set of requirements for access control and the relative importance of those requirements, as a means of driving and evaluating an appropriate access control service for the museum. How can Samuel define such a set of requirements?

## 85.3. Context

An organization understands how it plans to use access control, for example, from applying ENTERPRISE SECURITY SERVICES. An organization understands the general types of actors, assets and actions that are to be subject to access control. An access control rule permits an actor to perform an action on an asset—for example, user A is granted permission to modify file F. Actor types can include humans, software, business or automated processes, or information systems. Actors can be internal to an organization, such as an employee, or external, such as a supplier or customer. Action types include both physical and automated

actions. Common actions include create, see, use, change, and destroy or delete. Asset types include both physical and informational assets.

## 85.4. Problem

You need a clear set of requirements to ensure that the strategy employed for access control actually satisfies the needs of the organization or system. Requirements for access control often conflict with each other, and trade-offs among them are often necessary. The conflict stated in the example is that the need to protect gems by denying unauthorized access must be balanced with the need to permit visitors to view the gems.

How can you determine the specific requirements for an access control service, and their relative importance?

The process of selecting and prioritizing access control requirements needs to balance the following forces:

- You can use access control to help achieve desired security properties, especially confidentiality and integrity.
- Access control has many associated costs, not only the money for its deployment, but also support personnel, software, latency, annoyance for users, and so on.
- Access control adds complexity for software, systems, users, and administration.
- Access control should be consistent with the organization's security policies, and specifically with access control policies.
- The complexity of administering access control must be reasonable or the administrator will make errors, resulting in vulnerabilities.
- You cannot deploy access control as a stand-alone facility, it needs to interface or integrate with other security services, thus increasing complexity.
- Extremely high levels of control tend to achieve the desired result of denying most unauthorized access, but also tend to achieve the undesired result of denying more authorized access and making the asset or system harder to use.
- Moderate levels of control tend to achieve the desired result of allowing most authorized access, but also tend to achieve the undesired result of allowing more unauthorized access.
- The elements of the access control service need protection if the service is to perform its function.

## 85.5. Solution

Specify a set of access control requirements for a specific domain such as a system or organization and determine the relative importance of each requirement. The solution has two aspects: a requirements process and a common set of generic requirements.

### 85.5.1. Requirements Specification and Prioritization Process

A system requirements engineer, in conjunction with an enterprise architect, typically perform requirements capture. An important first step is explicitly to define the domain for which you are specifying access control requirements, such as a specific system or facility. You also define

factors that affect specialization and importance of requirements, such as organization constraints. You then specify access control requirements for the target domain, using the generic requirements provided below. The final activity is to define the relative importance of the specified requirements.

### 85.5.2. Generic Access Control Requirements

The following is a general set of requirements appropriate to access control services. An engineer will need to consider each of these and determine its priority based on criteria specific to the target domain, as well as on broader organization constraints. Additional requirements may be added to this list to address the system's unique characteristics. Some of the general requirements below represent access control functional requirements. The remaining requirements represent access control non-functional requirements, including requirements for security of the access control service.

- Deny unauthorized access. One primary purpose of access control is to deny unauthorized access requests. No access control service is perfect, and therefore errors will be made in which unauthorized access will be permitted. The goal of this requirement is to keep such errors to a minimum. The importance of this requirement needs to be weighed against requirements for other functional services.
- Permit authorized access. The second primary purpose of access control is to permit authorized access requests. The goal of this requirement is to keep to a minimum errors in which authorized access will be denied. Sometimes this type of error is caused by an operational error in the access control service; sometimes it is caused by the service's inability to support a desired authorizations policy, and sometimes it is caused by an incorrect access control service policy statement.
- Limit the damage when unauthorized access is permitted. A strong security principle is to avoid relying on a single point of failure. This requirement says that a single error in which unauthorized access is permitted should not permit access to multiple actions. The well-known defense-in-depth approach, using multiple layers of security, could be used in addressing this requirement. This requirement needs to be weighed against the 'limit the blockage' and 'minimize burden' requirements below.
- Limit the blockage when authorized access is denied. Consider an access control error in which authorized access is denied. This requirement says that a single failure of this type should not cause a serious interruption of business by denying many actions. This requirement needs to be weighed against the 'limit the damage' requirement above and the 'minimize burden' requirement below.
- Minimize the burden of access control. The burden of access control is an issue that affects multiple players and activities, including system users, interaction with other security services, processing resources, and implementers of the access control service. Each of these will be discussed briefly.

The access control service should control similar actions in a similar way, to minimize the perceived complexity for human users and developers of non-human actors, and to minimize the likelihood of errors. The access control service functionality depends on effective I&A. I&A should therefore have an interface that accommodates the access control service easily.

Processing overheads can cause reduction in availability of operations on assets for authorized users. This reduction may be due to blocking requests that should be

permitted, or due to interruptions of the request flow caused by the access control service. Latency can become a factor. For example, when every action needs to request permission from a remote access control server, overhead can be significant. Factoring the commonalities among access control requirements to produce a small generic set, as in this pattern, has several purposes. One of them is to reduce the burden on access control implementers by enabling them to define the system with a minimal set of primitives.

- Support desired authorization policies. The function of the access control service is to enforce the authorization policies defined to meet the business needs for the system or domain for which the service has responsibility. The access control service should be designed to enforce the required policies.

Definition and selection of access control policies is a key element. In fact, access control is about defining policies for authorization and then enforcing these policies through specific mechanisms. For example, a fundamental policy is ‘open versus closed’ systems. In an open system everything is allowed unless explicitly forbidden. In a closed system everything is forbidden unless explicitly allowed. Another set of fundamental policies is defined by the choices among the access control models like access matrix, role-based access control, multi-level control, and attribute-based control. Any access control system must implement one or more of these.

At the most generic level, therefore, the requirement is that an access control service must support all desired authorization policies. At a more specific level, authorization policies are selected that the implementation must enforce.

- Make the access control service flexible. Authorization policy statements sometimes change. This requirement says that adaptation to those changes should be fast, easy, and reliable. That is, the access control service should accommodate policy changes without high cost, complex administration, or increased difficulty of validating that the access control service requirements accurately reflect the authorization policy statements.

Access control also needs to be flexible, to accommodate legitimate operational changes or exceptions. For example, when the threat of terrorist attack is perceived to be high the organization may require stringent checks at facility entry points at the cost of substantial delays. Employees and customers may tolerate such delays for a week or two, but not for months. An opposite example is the case of a hospital, where, if a patient’s life is at stake, blocking access to normally protected patient data may be wholly unacceptable. The system should make some provision that allows access, such as emergency override. At the same time, to provide protection in such incidents, the access control service should record the emergency activity automatically. This will enable a forensic activity or an audit to determine the facts about the violation of normal access rules, and to determine their legitimacy.

Another area of flexibility is granularity. An access control service must be able to support a policy that supports both fine-grained control, such as specific elements in a database, or coarse-grained control, such as a whole database or group of users. In addition, an access control service should be able to support conditional authorization, such as permitting access at certain times of the day but not at others.

An additional set of requirements applies to all service requirements patterns. Instead of duplicating the discussion of the same set in each requirements pattern, they are simply listed here, because they need to be considered in each requirements pattern. The requirements

are: minimize time and effort to use, mismatch with users, risks to user safety, costs of per-user setup, costs of maintenance, management, and overhead, and changes needed to existing system infrastructure. The final requirement is to provide security protection of the service and its assets. Further discussion of each of these cross-cutting requirements, including implementation factors, is given in I&A REQUIREMENTS.

## **85.6. Structure**

## **85.7. Dynamics**

## **85.8. Implementation**

This implementation section first provides more detail on the process that was summarized in the Solution section, then discusses factors for determining the relative importance of requirements.

### **85.8.1. Process Guidelines**

The requirements process typically includes these steps:

1. Establish the domain for which the access control service is needed. Ensure that the domain has been identified and scoped. Typical access control domains include information system, physical facility, network, portal, or entire organization. Typical scope definition includes a defined set of actors, of assets, and of actions on those assets. Other constraints may bound the domain—for example, the access control requirements for entering a designated facility during normal work hours may differ from the requirements during out of work hours such as night-time and weekends: these would represent two domains.
2. Specify a set of factors that affect specialization and importance of requirements. The factors include uses of access control, access control needs, organization constraints, and priorities. You can find a general candidate set of factors in the next section.
3. Select one or more appropriate access control policies, such as a closed system policy and a role-based model, as discussed above. For security sensitive areas, it is generally considered better practice to follow a closed system policy, that is, to default to denial of access when it is not explicitly permitted. In less sensitive situations, an open system policy may be more appropriate, in which anything is permitted unless it is explicitly denied—for example, most information on public Web pages.
4. Specify the granularity levels at which access control will be applied. The level of granularity of the domain or asset to which access is specified can vary. For example, access to a physical facility such as a campus may be defined at the level of the entire facility, or a specific building, or a floor in the building, or a specific room. Access to a relational database may be defined at the level of the entire database, or to a specific partition or region, or a specific table, or specific rows in the table, or specific fields.

- The requirements need to specify the desired granularity, and often the requirement is to support multiple levels simultaneously.
5. Specify access control requirements for the target access control domain. To do this, specialize the set of generic requirements given in the Solution section.
  6. Define the relative importance of specific requirements. You can find more details on the association of factors and requirements below.

### 85.8.2. Factors in Determining Relative Importance

Table 4 presents factors for judging the relative importance to the organization of the generic access control requirements that were identified in the Solution section. For each requirement, the table also describes how the factors affect the relative priority of the requirement. For an example of applying these factors to each requirement, see the Example section below.

Generic Requirement	Factor	Impact on Priority
Deny unauthorized access	When sensitivity of assets is very high, or ability to validate credentials of an actor is suspect, the preferred approach is to block all suspected unauthorized requests.	This requirement should have increased priority if allowing unauthorized access could cause significant damage to the system. This requirement needs to be balanced with the need to permit authorized access for business needs.
Permit authorized access	Users are a higher priority than assets and blocking authorized activities would create severe problems for the organization or system	This requirement should have increased priority if denying authorized access would cause excessive levels of disruption of business functions, or excessive levels of user dissatisfaction with system. This requirement needs to be balanced with the need to deny unauthorized access.
Limit the damage when unauthorized access is permitted	Can use multiple levels of protection by increasing the number of actions required to achieve complete access.	This requirement should have increased priority if failure to block unauthorized access is likely to cascade into additional failures of security services. This requirement needs to be balanced with the need for ease of use: users may become frustrated with any multiple control paths they must navigate to gain access.
Limit the blockage when authorized access is denied	Consider high priority for this if user accessibility is of high importance.	This requirement should have increased priority if the controls are likely to cascade into excessive frustration and productivity loss of legitimate users due to erroneous denial of access. This requirement needs to be balanced with the need to deny unauthorized access.
Minimize burden of access control	System has tight constraints for performance and asset availability, as well as	This requirement should have increased priority if a high burden of using the access control service would cause excessive levels

	functionality of other services in the system.	of user dissatisfaction with system or would disrupt business functions. This requirement needs to be balanced with the need to deny unauthorized access.
Support desired authorization policies	The access control service is useful only if it supports the designated policies.	This requirement should always have high priority.
Make access control service flexible	Some organizations or domains have a diverse set of authorization policies, or the policies or access context change often, or policies need to operate in two or more modes, such as normal, increased security, and emergency override.	This requirement should have increased priority if your organization or domain has the characteristics described in this factor. Flexibility is important to permit users needed access in emergency situations, or to increase system protection when specific threats increase significantly. This requirement needs to be balanced with the need for ease of use and simplicity of design.

Table 4: Access control requirements factors

## 85.9. Example Resolved

Samuel the museum's system engineer defines the domain for access control to include the gem assets themselves, as well as sensitive information about the gems. Although these may be regarded as two different domains for some purposes, Samuel decided to define a single requirements set for both. A clear starting point is a closed authorization policy, in which access to both the gems and information about the gems is forbidden unless explicitly allowed.

Samuel, in consultation with Edward the museum architect, has also determined that the access control service will give greater importance to protection of the assets and sensitive asset information than to immediate satisfaction of user requests. The museum is inclined to disallow even valid requests if anything suspicious is detected in the activity. On the other hand, they will strive to make their unsophisticated user base less aware of the security controls by not presenting multiple rechecking at every step. The actual system policy approaches are known, and Samuel does not anticipate any need for expansion of the number or type of policies enforced. Samuel sees two potential modes of operation: normal conditions and an emergency lock-down.

Table 5 shows the requirements Samuel specified for the stated domain.

Generic Requirement	Museum Requirement and Priority
Deny unauthorized access	High priority – the museum requires access control to provide a certainty of at least 0.9999 for denying unauthorized access to high value gems, meaning that the service shall allow no more than one successful access out of 10,000 unauthorized attempts. The museum requires that access control provide a certainty of at least 0.99 for denying access to the associated gems information.
Permit authorized access	Moderate priority – the museum regards user convenience as a lower priority than protecting the assets under its care. The museum requires access control to provide a certainty of at least 0.98 for permitting

	authorized access to gems or gem information, meaning that the service shall deny no more than one access out of 50 authorized requests for access.
Limit the damage when unauthorized access is permitted	High priority for gems – the museum places high priority on avoiding inadvertent access to all gems. If Theo the thief is successful at circumventing access control to get his hands on one gem, that success must not give him access to all the other gems. Moderate priority for gem information – the priority of this requirement for gem information is balanced by the need for access by gem researchers, with the assumption that the user base of researchers will not be overly knowledgeable with regard to the information system.
Limit the blockage when authorized access is denied	Low priority – the museum gives higher priority to asset protection than to user access. They would prefer to occasionally have to address a locked-out user rather than lose an asset, or sensitive information about that asset.
Minimize burden of access control	Moderate priority – the museum will try to attain a middle ground with this requirement. They want effective access controls, but they don't want to impact other functional services, create bottlenecks, or create denial of service scenarios.
Support desired authorization policies	High priority – the museum has defined a closed system access control policy that focuses on the gems they protect and associated information. Samuel does not see that scenario changing over the long term.
Make access control service flexible	Moderate priority – the museum requires the access controls to change when they need to operate in emergency lock-down mode, as opposed to normal operating conditions, but the policy is not expected to change significantly.

Table 5: Museum requirements for access control service

## 85.10. Consequences

The following benefits may be expected from applying this pattern:

- It facilitates conscious selection of access control requirements, so that decisions about selecting access control mechanisms have a clear basis, rather than occurring in a vacuum.
- It promotes explicit analysis of trade-offs that encourages balancing and prioritizing of conflicting requirements. It helps avoid stronger than necessary access control that makes it difficult for valid users, and at the same time it helps avoid weaker than necessary access control that makes it easy for unauthorized actors to defeat.
- It results in documentation of access control requirements that communicates to all interested parties and also provides information for security audits.
- The pattern fosters a clear connection of requirements to authorization policies: this also encourages organizations to make their policies more explicit.

The following potential liabilities may arise from applying this pattern:

- An investment of resources is required to apply the pattern, including time to analyze domains and access control needs. In some cases, the cost of applying the pattern may exceed its benefits.

- It poses a danger of over-engineering and complexity creep if stakeholders are offered too many options. You can mitigate this by using the requirements as guidelines only for analysis, or by selecting parts of the pattern that give the most benefit.
- The formal selection process may be too long and costly and produce too much overhead. You can mitigate this in the same way as noted above.
- Specific circumstances might not be covered by generic access control requirements. You can mitigate this by adding specific requirements and including them in the trade-offs.
- Documentation of requirements implies that they must be maintained as they change over time. You can mitigate this by keeping the requirements in a form that is easy to update, integrated with other system documentation.
- Perception of access control requirements can differ throughout an organization or in a particular domain. This may make it difficult to reach agreement on priorities of requirements. On the other hand, bringing such disagreements to the surface may be a benefit of the pattern, because then they can be properly discussed and resolved.

## **85.11. Known Uses**

The general access control requirements and the process of specifying access control requirements described in this pattern are widely known, but are generally used informally, as opposed to being codified or published. The requirements as stated in this pattern represent a consolidation of MITRE Corporation experience in working with multiple customers over several decades. However, some publications on access control requirements exist. The examples that follow emphasize the value of defining access control requirements explicitly, and the separation of policy from mechanism while maintaining adherence of mechanism to policy, consistent with this pattern.

- [1] is a discussion of access control requirements for LDAP. In addition to LDAP access control requirements, it discusses policy requirements, granularity, and nonfunctional requirements, especially usability.
- [2] discusses access control requirements in the context of virtual organizations. The authors discuss authorization and access control-related languages and standards, and access control policy requirements. They stress the importance of defining security domains for access control, and interoperability and composition among domains and their associated policies and models.
- [3] is a case study used to motivate access control requirements. It discusses granularity and some of the nonfunctional requirements identified in this pattern.
- [4] is an international standard that defines evaluation criteria for information technology security. It includes a class or family of criteria that address the requirements for functions to define authorization or access control policy, and explicitly authorize or deny access of a subject to perform an operation on an object in conformance with that policy.
- [5] identifies general desiderata or requirements for access control, and how they are expressed in policies. It discusses how the requirements are addressed in several current operating systems, database management systems, and network solutions.

## **85.12. See Also**

After applying this solution, the next step typically is to apply architecture or design patterns that help satisfy the specified requirements for access control.

Patterns like the SINGLE ACCESS POINT, CHECK POINT, and SECURITY SESSION build on each other, showing how to implement an architecture providing I&A and access control to an application or system.

## **85.13. References**

- [1] Stokes, E., Byrne, D., Blakley, B., & Behera, P. (2000). *Access Control Requirements for LDAP* (No. rfc2820).
- [2] Coetzee, M., & Elof, J. H. (2003, September). Virtual enterprise access control requirements. In *Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology* (pp. 285-294).
- [3] Evered, M., & Bögeholz, S. (2004, January). A case study in access control requirements for a health information system. In *Proceedings of the second workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation- Volume 32* (pp. 53-61).
- [4] Organisations, C. C. P. S. (1999). Common criteria for information technology security evaluation (version 2.1). *Common Criteria Project Sponsoring Organisations*, 1-3.
- [5] De Capitani di Vimercati, S., Paraboschi, S., & Samarati, P. (2003). Access control: principles and solutions. *Software: Practice and Experience*, 33(5), 397-421.

## **85.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# 86. Asset Valuation

## 86.1. Intent

Asset valuation helps you to determine the overall importance an enterprise places on the assets it owns and controls. Loss or compromise of such assets may result in anything from hard costs, such as fines and fees, to soft costs due to loss of market share and consumer confidence.

## 86.2. Example

The museum has begun a risk assessment and identified the following assets to be in scope:

- Information Assets
  - Museum employee data
  - Museum financial/insurance data, partner financial data
  - Museum contractual data and business planning
  - Museum research and associated data
  - Museum advertisements and other public data
  - Museum database of collections information
- Physical Assets
  - Museum building
  - Museum staff
  - Museum collections and exhibits
  - Museum transport vehicles

They must now determine the overall importance of these assets.

## 86.3. Context

An enterprise has determined which assets are to be included in the overall risk assessment process and must now ascertain the value it places on those assets.

## 86.4. Problem

The ability to define an asset's value is a key component of any risk assessment. Threats and vulnerabilities that target and expose an asset are only significant within the context of the asset's value. Without this determination, an enterprise is unable to properly assess the risks posed to its assets. How can an enterprise determine the overall value of its assets?

An enterprise must resolve the following forces:

- It must develop a standardized way of assessing and describing an asset's worth
- It must provide consistent results despite the subjectivity inherent in this process
- It must be able to assess, as much as possible, the soft costs due to loss or compromise of an asset
- It may not be able fully to evaluate the safety impact due to the loss of an asset without having previously experienced a harmful event

- When evaluating hard costs, an enterprise may waste time on incidental costs, or those of much lesser value relative to the asset

## **86.5. Solution**

Systematically determine the overall value of the assets identified in the scope of the risk assessment. This process means to perform the following four steps:

1. Determine the security value. Determine the security value of the asset based on the importance the enterprise places on guaranteeing the asset's information security properties: confidentiality, integrity, availability, and accountability.
2. Determine the financial value. Determine the financial value of the enterprise asset based on the cost of repair or replacement as well as the cost to maintain and operate the asset. Remember that costs such as electricity and storage are probably distributed across many assets.
3. Determine the impact to business. Determine the value of the asset in relation to the impact a compromise of the asset would have on the enterprise's business processes.
4. Determine the overall value and build an asset valuation table. Combine the results of the security, financial and business valuations and determine the overall value the enterprise places on the asset. Enter these results into the asset valuation table.

## **86.6. Structure**

## **86.7. Dynamics**

The allowable sequence for performing the asset valuation process is shown in figure 182. The three factors, security value, financial value, and business impact, can be assessed in any order. The results are collected and entered into the asset valuation table.

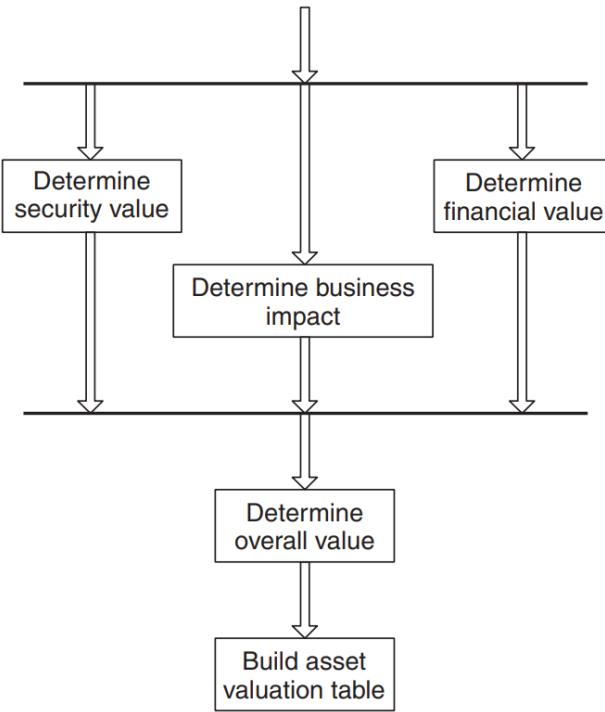


Figure 182: Sequence constraints of the asset valuation process

## 86.8. Implementation

For each section, create a value rating scale by defining a range, then providing an accompanying description. Examples for security value, financial value and business impact have been provided in Tables 6, 7, and 8 respectively. To maintain consistency with THREAT ASSESSMENT and VULNERABILITY ASSESSMENT, six ratings are defined. Note that the ratings may be modified according to the preference of the enterprise, although they should remain consistent throughout the risk assessment.

1. Determine the security value. Determine the security value of an asset by considering the following:
  - a. Demand for multiple security properties of a single asset. An asset that requires all four properties would have a significantly higher value than an asset that has a single requirement.
  - b. The extent of the consequence due to a compromise of an asset's security property. For example, the consequence of the unauthorized modification to a payment transaction could be far more severe than modification to a research and development document, though they both require integrity protection.
  - c. Health and safety implications resulting from the loss or damage of physical assets, for example ladders, bridges, security guards, or informational assets, such as evacuation procedures, fire containment and first aid instructions.

The information security requirements are normally provided by the asset owner. Otherwise, the security needs of the asset can be identified by applying SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS.

Use Table 6 to qualify the security value placed on the asset.

Rating	Qualitative	Description
--------	-------------	-------------

6	Extreme	The asset requires an extreme degree of confidentiality, integrity, and availability. Compromise of these security properties would expose massive amounts of confidential information and endanger public safety.
5	Very high	The asset requires a very high degree of confidentiality, integrity, availability, or accountability. Compromise of one or more of these properties would expose sensitive information and possibly endanger public safety.
4	High	The asset requires a high degree of confidentiality, integrity, availability, or accountability. Compromise of these properties would expose sensitive information, violating local or federal legislation.
3	Medium	The asset has a moderate requirement for information security controls. Compromise of the security properties would violate corporate policy and possibly local or federal legislation.
2	Low	The asset has a low requirement for security. Compromise of the asset would expose only non-critical information or data.
1	Negligible	The information is publicly available, or the asset has no information security value for the enterprise.

*Table 6: Security requirements rating*

2. Determine the financial value. Determine the financial value of an asset by considering the following:
  - a. Cost to replace or repair asset due to a damaging event
  - b. Regulatory or legal penalties, fines or fees incurred due to a security violation
  - c. Transaction value for which the asset is responsible
  - d. Time incurred by an employee or contractor to receive, configure, or maintain an asset
  - e. Loss of productive employee time, or work backlog incurred

Replacement and repair costs can be obtained from an enterprise's procurement department, which in turn obtains them either from a VAR (value-added reseller) or directly from the vendor. Regulatory fines and penalties are usually publicly and readily available from the entity that enforces the fines, such as a local or federal government. Hard costs associated with assets that serve a health and safety purpose include hospital and insurance fees, as well as the cost of a cleanup from a fire or flood.

Use Table 7 to qualify the financial value placed on the asset.

Rating	Qualitative	Description
6	Extreme	The asset has an extreme monetary value for the enterprise. Loss or damage of the asset would probably bankrupt the enterprise.
5	Very high	The asset has a major monetary value. Loss or damage of the asset would impose a substantial financial burden on the enterprise.
4	High	The asset has a significant monetary value. Repair or replacement would require significant funds.
3	Medium	The asset has moderate financial value. Loss or damage of the asset would require financial repurposing.
2	Low	The asset has low financial value to the company.
1	Negligible	The asset has no monetary value.

*Table 7: Financial value rating*

3. Determine the impact to business. Determine the impact to business processes by considering the following:
  - a. Loss of customer or investor confidence as a result in the compromise of the asset
  - b. Loss of competitive advantage due to compromise of security properties
  - c. Impact to enterprise partner relationships, or other contractual repercussions
  - d. Lack (or presence) of alternate service: that is, if an alternate service exists that can fulfill customer needs, then the loss of one service (or asset) may have reduced implications
  - e. Extent of disruption to other enterprise services due to asset dependencies
  - f. Percentage of customer base affected by outage or degradation of service

Disaster recovery and business continuity plans found in many enterprises may already sort assets by value to the organization. This can provide a starting point for defining the relative business value an enterprise places on the asset.

Business impact is inherently more subjective and difficult to assess than hard costs.

Quite often one may not be able to completely predict the loss of customer confidence, or the extent of loss of competitive advantage. One may, however, draw on events that have occurred to other enterprises of similar size in similar markets.

Use Table 8 to qualify the business value placed on the asset.

<b>Rating</b>	<b>Qualitative</b>	<b>Description</b>
6	Extreme	The enterprise cannot function without this asset. Its compromise or loss would result in immediate termination of critical business services.
5	Very high	This asset represents a major service of the enterprise. Its loss would result in termination of a critical service or severe degradation of many services.
4	High	This asset supports many enterprise services. Its loss would result in termination of a major service or degradation of services.
3	Medium	This asset supports a fair number of customers or supports a major service of the enterprise. Its loss would result in degradation of more important services.
2	Low	This asset supports an ancillary enterprise service. Its compromise would have a slight impact on business services.
1	Negligible	The loss of asset would have no impact to the business.

*Table 8: Business impact rating*

4. Determine the overall value. Determine the overall value the enterprise places on the asset from the results of the security, financial and business impact valuations. Use Table 9 to qualify the overall value of the asset to the enterprise and collect them in Table 10.

There will not be direct translation from these three ratings to the single overall value. It is more likely that the overall value will be the highest of the three ratings. That is, if an asset has a very high security value, but a low financial value, its overall value should still be appropriately high.

<b>Rating</b>	<b>Qualitative</b>	<b>Description</b>
6	Extreme	The enterprise places the highest possible value on this asset. Its compromise results in human deaths, immediate and total loss of business services or financial bankruptcy.

5	Very high	The asset represents or supports a critical business function for the enterprise. Loss or damage of it results in severe financial, security or health repercussions.
4	High	The asset is highly valued because of its security requirements or customer focus. Its loss would result in considerable harm to customer services and reputation.
3	Medium	The asset is of moderate value. It has some security needs and financial value. Compromise of it would impede the enterprise's mission.
2	Low	This asset is of minor financial value. Compromise of it results in little business impact.
1	Negligible	The asset has insignificant importance for the enterprise. It is easily replaced or repaired. It has little to no security requirements and represents no health impact.

Table 9: Overall asset value scale

Asset	Security Value	Financial Value	Business Impact	Overall Value

Table 10: Asset valuation table template

## 86.9. Example Resolved

After applying ASSET VALUATION, the museum has determined the value for its information and physical assets. These are shown in Tables 11 and 12 respectively.

Asset	Security Value	Financial Value	Business Impact	Overall Value
Museum employee data	5	3	5	5
Museum financial/insurance data, partner financial data	4	3	4	4
Museum contractual data and business planning	4	3	4	4
Museum research and associated data	3	2	3	3
Museum advertisements and other public data	1	2	2	2
Museum database of collections information	3	3	4	4

Table 11: Information asset valuation table

Asset	Security Value	Financial Value	Business Impact	Overall Value
Museum building	5	5	6	6
Museum staff	6	5	6	6
Museum collections and exhibits	5	6	5	6
Museum transport vehicles	3	2	2	3

Table 12: Physical asset valuation table

## 86.10. Consequences

This pattern has the following benefits:

- An enterprise obtains a realistic and complete view of which assets are most critical to its business.

- The results of the asset valuation can be used to develop or update an enterprise's disaster recovery and business continuity plan documents.
- A qualitative value can be easier to obtain than hard costs required by a quantitative asset valuation, thus expediting the overall risk assessment process.

As well as the following liabilities:

- An enterprise may be forced to change its practices if it determines that an asset is worth a great deal more than it thought—a change which may be difficult and expensive. This will undoubtedly become a benefit in the long-term.
- The individuals implementing the pattern may find difficulty in translating results from other parties—for example, hard costs or subjective results—into values consistent with the rating system used here.

## 86.11. Variants

An enterprise may choose a different scale or demand a more complete qualitative description. This is certainly acceptable: the important consideration is that the scale be consistent throughout the use of the pattern and for all assets. For example, [1] uses the following qualitative values for information sensitivity:

- High: extreme sensitivity—restricted to specific individual need to know. Its loss or compromise may cause severe financial, legal, or reputation damage to the enterprise.
- Medium: used only by specific authorized groups with legitimate business need. May have significant adverse impact, possible negative financial impact.
- Low: information for internal business use within the company. May have adverse impact, negligible financial impact.

[1] also uses the following quantitative table to describe financial loss:

Financial Loss	Valuation Score
Less than \$2,000	1
Between \$2k and \$15k	2
Between \$15k and \$40k	3
Between \$40k and \$100k	4
Between \$100k and \$300k	5
Between \$300k and \$1M	6
Between \$1M and \$3M	7
Between \$3M and \$10M	8
Between \$10M and \$30M	9
Over \$30M	10

Table 13: Financial loss scale

## 86.12. Known Uses

An asset valuation is a key component of all widely accepted risk assessments, including those from [1,2,3,4], and others. While they differ slightly in their approach, their purposes and overall goals are consistent.

## **86.13. See Also**

## **86.14. References**

- [1] Peltier, T. R. (2001). *Information security risk analysis*. Auerbach Publications
- [2] Stoneburner, G., Goguen, A., & Feringa, A. (2002). Risk management guide for information technology systems. *Nist special publication*, 800(30), 800-30.
- [3] "Technical Report ISO13335-3 Part 3: Techniques for the Management of IT Security" International Organization for Standardization, American National Standards Institute, 2002
- [4] "ISO/IEC 17799:2005 Information Technology – Code of Practice for Information Security Management" International Organization for Standardization, 2000.

## **86.15. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **87. Audit Requirements**

## **87.1. Intent**

An audit service must satisfy a set of requirements for both the service and the quality of service. The audit function is to analyze logs, audit trails or other captured information about an event, such as entering a building or accessing resources on a network, to find and report any indication of security violations. While each situation that calls for an audit is unique, there are common generic requirements that apply to all audit situations. This pattern provides a common generic set of audit requirements. The pattern also helps you to apply the general requirements to your specific situation, and helps you determine the relative importance of conflicting requirements.

## **87.2. Example**

The museum's research department has a network that they use for messaging and collaboration with various universities around the world. Among the types of information exchanged and stored are details about the location of various gemstone mines. Every six months the museum must present a report of the information exchanges to the board of trustees. The museum wants to take six months' worth of activity and summarize it into the critical and non-critical events that occurred over that six-month period, and who was involved in those events. Samuel the museum system engineer understands this goal, but at the same time he understands that capturing extensive audit information can degrade system performance and require significant resources for storage and analysis. Privacy considerations are also a constraint on the capture and use of audit data. Samuel needs to identify requirements for an audit service that will help the museum achieve the goals while balancing the constraints.

## **87.3. Context**

Accounting requirements and their relative importance are understood, for example, from applying SECURITY ACCOUNTING REQUIREMENTS. The planned uses of audit are understood.

## **87.4. Problem**

Audit is a security service that scrutinizes logs, audit trails or other captured information and attempts to discern more detailed information about an event. It analyzes the event information for any indication of security violations. You need a clear set of requirements to ensure that the audit strategy employed actually satisfies the needs of the organization or system. Requirements for audit often conflict with each other, and trade-offs among them are often necessary. The conflict stated above in the example is that the need to provide an audit trail must be balanced with resource and privacy constraints. What types of information are appropriate or required for an audit system to analyze?

How can you determine a balanced set of specific requirements for an audit service, and their relative importance?

The process of selecting and prioritizing audit requirements needs to balance the following forces:

- Collecting extensive relevant audit raw data increases the likelihood of achieving the desired security properties, especially accountability.
- Collecting extensive audit raw data increases the risk of violating privacy laws, or of abusing such data, or of damaging the reputation of the collector.
- Applying audit has many associated costs (support personnel, software, additional processing time, and so on) that are counter to the organization goal of minimizing total costs.
- Audit errors can result in lack of accountability in two ways. If person A commits an act that violates security, and the audit concludes that person B committed this act, then (1) person A is not held accountable for his action, and (2) person B is incorrectly held accountable and suffers consequences for an act he did not commit.

## **87.5. Solution**

Specify a set of audit requirements for a specific domain such as a system or organization and determine the relative importance of each requirement. The solution has two aspects: a requirements process and a common set of generic requirements.

### **87.5.1. Requirements Specification and Prioritization Process**

A system requirements engineer, in conjunction with an enterprise architect, typically perform the requirements process. An important first step is explicitly to define the domain for which you are specifying audit requirements, such as a specific system, or types of activities and events. You also define factors such as organization constraints that affect the specialization and importance of requirements. Then you specify audit requirements for the target domain, using the generic requirements provided below. The final activity is to define the relative importance of the specified requirements.

### **87.5.2. Generic Requirements Description**

The audit function is to analyze logs, audit trails or other captured information about an event, such as entering a building or accessing resources on a network, to find and report any indication of security violations. The following is a general set of requirements appropriate to an audit service.

- Provide information about malicious and unwanted events. An audit service must provide information about events that are actually or potentially harmful to the organization. The information is used to determine what the event was, when and where the event happened, and why and how the event happened. It is also used to determine where organization vulnerabilities exist, and help the organization determine how a threat may have become a reality. The circumstances of an undesirable event, including the location and time of day and date, should be captured. The location of the event needs to be included in the details so that planners and investigators of the event can examine the area for more clues about the event. Location might be physical, such as a building, room, or gate, or ‘virtual,’ such as a

network or a Web site. Other event information may include whether or not any elements of the attack were detected in advance, and responses that ensued. This requirement implies an ability to distinguish desirable or normal events from undesirable ones. Such a distinction is often not possible until after the data is captured and analyzed.

- Provide information that associates actors with events. An actor may be a person, or a hardware or software element. The audit service needs to provide information not only on events that occur, but also what actors were involved in the events. A minimum requirement is to provide actor identification. Other actor information may include the location of the actor during the event and role of the actor in the event. The information is used eventually to assign the responsibility of the event to the actors. Information on thwarted attacks will need to be fed back to security officers to ensure awareness about what does work.
- Provide information on actor activity over a period of time. Predicting behavior can be used to prevent malicious activity from harming the organization. Over the course of time users develop habits when using a system, and those habits can be gathered into a user profile. An audit system can be used to provide details of these habits to distinguish one user from another. This information can also be used to better understand the vulnerabilities in the system and allow decision makers the opportunity to dictate what vulnerabilities should be addressed. Decision makers want to be sure that actors who engage in activity in the organization are performing their duty in a manner that does not violate or threaten security.
- Be able to determine what captured information is relevant. An audit usually takes place over an extended period of time. Audit also examines information about events that have been captured over an extended period of time. In order to identify security violations that have occurred over that time, an audit activity must take care to examine the information carefully and thoroughly to ensure that the relevant information has been discerned from the captured events. Audit logs typically contain a large amount of captured information, but the information that pertains to an event of interest is by comparison very small. Finding the relevant information is often not an easy task. Sometimes even determining which events are of interest is not easy.
- Perform its service when needed. An audit system needs to be able to provide its services during times when the tracking of events is absolutely important, yet the ability to do so may be hampered by attacks. This requirement is essential to support availability and concerns the readiness of the audit service.
- Provide reliable and accurate information. An audit system provides information relative to a specific event, and the user of the audit information wants to have confidence in the reliability and integrity of the information. Although audit was not responsible for capturing or storing the raw logging information that was input to the audit process, it may also need to provide information about the reliability and integrity of that information as well. Those working with the audit mechanisms must be trusted not to alter any information previously captured. In addition, any automated tools supporting the audit process need to be fully understood with regard to how they process the information and the rules used for establishing associations between disparate pieces of information. This requirement is essential to support integrity and concerns the trustworthiness of the information the audit service provides.

An additional set of requirements applies to all service requirements patterns. Instead of duplicating the discussion of the same set in each requirements pattern, they are simply listed here, because they do need to be considered in each requirements pattern. The requirements are: minimize time and effort to use, minimize mismatch with user characteristics, risks to user safety, costs of per-user set-up, costs of maintenance, management, and overhead, and changes needed to existing system infrastructure. Further discussion of each of these cross-cutting requirements, including implementation factors, is given in I&A REQUIREMENTS.

The remainder of this pattern focuses on the audit-specific requirements identified and discussed above.

## **87.6. Structure**

## **87.7. Dynamics**

## **87.8. Implementation**

This section first provides more detail about the process that was summarized in the Solution section, then discusses factors in determining relative importance of requirements.

### **87.8.1. Process Guidelines**

The requirements process is typically performed by a system requirements engineer, in conjunction with an enterprise architect, and includes several steps:

1. Establish the domain for which the audit service is needed. Ensure that the domain has been identified and scoped: typical audit domains include information system, physical facility, network, portal, category of events, or entire organization. The domain consists of at least three parts: a defined scope of actors or users, a defined scope of assets, and a defined scope or set of events that involve actions or operations on those assets. Other constraints may bound the domain—for example, the audit requirements for a real-time service may differ from those for a multi-year service: these might represent two domains.
2. Specify a set of factors that affect the specialization and importance of requirements. The factors include use of audit, audit needs, organization constraints, and priorities. You can find a general candidate set of factors below.
3. Specify the audit requirements for the target audit domain. To do this, specialize the set of generic requirements given in the Solution section.
4. Define the relative importance of specific requirements.

## 87.8.2. Factors in Determining Relative Importance

Table 14 reiterates the generic requirements described in the Solution section and identifies factors for judging their relative importance to an organization or system. For each factor, the table also indicates the resulting requirement priority, in terms of High, Medium, and Low.

Generic Requirement	Factor	Resulting Priority
Provide information about malicious and unwanted events	Required by law or other mandate outside of the organization, or events involve highly sensitive or valuable assets.	High
	Internal organization concern rather than external mandate, or events involve assets of medium value	Medium
	Only prevention approach used, not detection or response, or events involve low value assets.	Low
Provide information associating actors with events	Assigning responsibility is high priority, because it is required by law, or events involve highly sensitive or valuable assets.	High
	Accountability is an organization concern and not a legal or external mandate, or events involve assets of medium value, or losses are covered by insurance or fall within the boundaries of acceptable risk.	Medium
	No action will be taken against individuals, or events involve low value assets.	Low
Provide information on actor activity over a period of time	Actor behavior is a concern to boards, customers, or regulatory entities, who require the organization to provide this information.	High
	Accountability is an organization concern and not a legal or external mandate, or activities and behavior patterns involve assets of medium value.	Medium
	Actions are not long-lasting and are of minimal impact.	Low
Be able to determine what captured information is relevant	Event information is to be used by boards, customers or regulatory entities who require the organization to provide this information.	High
	Accountability is an organization concern and not a legal or external mandate, or activities and behavior patterns involve assets of medium value.	Medium
	Only a small amount of audit log information is captured, or the overall need for audit service is low.	Low
Perform its service when needed	The need for accountability is high, and security accounting is the only source of this information.	High
	The need for accountability is moderate, or alternative sources of accounting information are available.	Medium

Provide reliable and accurate information	The need for accountability is high, or security accounting information must be provided to an outside organization.  The need for accountability is moderate, and only required inside the organization.	High  Medium
---	---	--------------------

Table 14: Audit service requirements factors

## 87.9. Example Resolved

Samuel the museum systems engineer defines the museum's research network as an audit domain. Table 15 shows the requirements ratings Samuel has specified for this domain.

Requirement	Museum Priority and Concern
Provide information about malicious and unwanted events	HIGH – The museum decision makers want to audit all activity across the organization.
Provide information associating actors with events	HIGH – The museum decision makers want information that can hold people responsible for malicious activities.
Provide information on actor activity over a period of time	HIGH – The museum decision makers want information audited over a six-month period.
Be able to determine what captured information is relevant	MEDIUM – It is important for the museum to get as much factual data as possible. However, they would not expend a large number of resources to do so.
Perform its service when needed	LOW – Although important to have audit ready when it is needed, the museum decision makers need audit every six months. They would not need to expend resources to make audit highly available.
Provide reliable and accurate information	HIGH – The integrity of data is critical to the museum obtaining an accurate report. The museum would allocate resources to ensure that the information they start with is in fact accurate.

Table 15: Resolution of example problem for AUDIT REQUIREMENTS

## 87.10. Consequences

The following benefits may be expected from applying this pattern:

- It facilitates conscious selection of audit requirements, so that decisions about selecting audit mechanisms have a clear basis, rather than occurring in a vacuum.
- It promotes explicit analysis of trade-offs that encourages balancing and prioritizing of conflicting requirements and forces. This includes balancing the need for accountability with the need for privacy. This helps to avoid stronger than necessary audit mechanisms that would make it difficult for valid users, and at the same time it helps to avoid weaker than necessary audit that makes it easy for unauthorized actors to avoid.
- It results in documentation of audit requirements that communicates to all interested parties and is useful in determining the adequacy of accounting services such as audits.
- The explicit requirements resulting from the pattern foster a clear connection of requirements to audit policies. This also encourages organizations to make their accounting policies more explicit.

The following potential liabilities may arise from applying this pattern:

- It requires an investment of resources to apply the pattern, including time to analyze domains and audit needs. In some cases, the cost of applying the pattern may exceed its benefits.
- It poses a danger of possible violation of privacy rights if extensive data is captured and analyzed. You can mitigate this by capturing and analyzing the minimum amount of data, and by working closely with your legal department.
- The formal selection process may be too long and costly and produce too much overhead. You can mitigate this in the same ways as noted above.
- Specific circumstances might not be covered by generic audit requirements. You can mitigate this by adding specific requirements and including them in the trade-offs.
- Documentation of requirements implies that they must be maintained as they change over time. You can mitigate this by keeping the requirements in a form that is easy to update, integrated with other system documentation.
- Perception of audit requirements can differ throughout an organization or in a particular domain. This may make it difficult to reach agreement on the relative priorities of requirements. On the other hand, bringing such disagreements to the surface may be a benefit of the pattern, because they can then be properly discussed and resolved.

## **87.11. Known Uses**

The general audit requirements and the process of specifying audit requirements described in this pattern are widely known, but are generally used informally, as opposed to being codified or published. The requirements as stated in this pattern represent a consolidation of MITRE Corporation’s experience in working with multiple customers over several decades. However, some publications on security audits and audit requirements exist:

- ISO standards [1] and [2] discuss security audits as one of the primary safeguards.
- [3] is an international standard that defines evaluation criteria for information technology security. It includes a class or family of criteria that address audit requirements, including data to be generated, analysis to be performed, and event storage.
- [4] discusses the COBRA method of security audit that includes questionnaires, checklists, and a tool to help automate audits.

## **87.12. See Also**

After applying this solution, or in parallel, you can apply AUDIT TRAILS AND LOGGING REQUIREMENTS. This pattern captures the information used by AUDIT REQUIREMENTS.

## **87.13. References**

- [1] “TR 13335-4:2000 Information Technology - Guidelines for the Management of IT Security - Part 4: Selection of Safeguards” International Organization for Standardization, 2000.
- [2] “ISO/IEC 17799:2005 Information Technology – Code of Practice for Information Security Management” International Organization for Standardization, 2000.

[3] Organisations, C. C. P. S. (1999). Common criteria for information technology security evaluation (version 2.1). *Common Criteria Project Sponsoring Organisations*, 1-3.

[4] Computer Audit. (2002), Systems Audit and Information Security Audit Made Easy!.  
<http://www.securitypolicy.co.uk/securityaudit/index.htm>

## **87.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# **88. Audit Trails and Logging Requirements**

## **88.1. Intent**

A service that captures security audit trails and audit logs must satisfy a set of requirements for both the service and the quality of service. The audit trails and logging function is to capture audit logs and audit trails about events and activities that occur within an organization or system, to enable reconstruction and analysis of those events and activities. While each situation that calls for an audit trail is unique, there are common generic requirements that apply to all audit trails and logging situations. This pattern provides a common generic set of audit trail requirements. The pattern also helps you to apply the general requirements to your specific situation and helps you to determine the relative importance of conflicting requirements.

## **88.2. Example**

The new museum wing for gemstones keeps its most precious gems in a room with limited access. The room's access is controlled by electronic badge access. Cleaning personnel, scientists and other authorized personnel need special badges to access the room. As an extra precaution, the museum would like a way to track access to the room by individuals and by roles. Samuel the museum system engineer needs to specify the requirements for audit trails and logging (AT&L) of activities related to this limited access room, and the relative importance of those requirements, as a means to drive and evaluate an AT&L service.

## **88.3. Context**

Audit requirements and their relative importance are understood, for example, from applying AUDIT REQUIREMENTS. The planned uses of audit trails and logging are understood.

## **88.4. Problem**

An organization needs to observe events and to revisit data related to those events to help achieve security properties in a system or domain, and to understand when and how security properties have been compromised. Audit trails and logging (AT&L) is a security service that automates the capturing of information about events and activities that occur within the organization. Audit trails are a series of records about system events or user activities. Audit trails can be used to reconstruct events, determine who is responsible for events, what malicious or unwanted activities have occurred, and analysis of any problems. Logs are individual trails of information that may be combined into an audit trail.

You need a clear set of AT&L requirements to guide selection or implementation of an AT&L service and to determine if it is adequate to address organization or system needs. These requirements need to be prioritized to determine under what circumstances an organization should put more emphasis on one requirement over another.

How can you determine a balanced set of specific requirements for an AT&L service, and their relative importance?

The process of selecting and prioritizing AT&L requirements needs to balance the following forces:

- Capturing logs and audit trails increases the likelihood of achieving desired security properties, especially accountability
- Capturing logs and audit trails requires resources and entails cost
- Capturing logs and audit trails increases the risk of violating privacy laws, or of abusing such data, or of damaging the reputation of the collector
- A higher capacity of logs and audit trails enables greater volume and frequency of data acquisition, and a greater length of time for which data is available, which in turn supports increased accounting capability
- A higher capacity of logs and audit trails requires greater processing and storage resources
- Following accepted community AT&L requirements tends to save implementation cost because tools are available to use
- Following accepted community requirements on collecting AT&L data may not give your organization exactly what you need
- AT&L data compression reduces required storage but requires compression and decompression tools

## **88.5. Solution**

Specify a set of AT&L requirements for a specific domain such as a system or organization and determine the relative importance of each requirement. The solution has two aspects: a requirements process and a common set of generic requirements.

### **88.5.1. Requirements Specification and Prioritization Process**

A system requirements engineer, in conjunction with an enterprise architect, typically performs the requirements process. An important first step is explicitly to define the domain for which you are specifying audit trails and logging requirements, such as a specific system, or type of activities and events. You also define factors, such as organization constraints, that affect the specialization and importance of requirements. You then specify AT&L requirements for the target domain, using the generic requirements provided below. The final activity is to define the relative importance of the specified requirements.

### **88.5.2. Generic Requirements Description**

The following is a general set of requirements appropriate to an AT&L service.

- Acquire information about designated types of activities and events. An AT&L service must support the capture and storage of information related to security events that are potentially harmful or undesirable to the organization in audit trails or logs. This requirement is essential for stakeholders, who use the details provided to determine what the event was, when and where the event happened, and why and how the event happened. Significant related information should be stored along with the event information. For example, the time of day and date should be included in details of an event. Best practice does not require audit trails or logs to be provided for immediate

viewing, although sometimes they are streamed to available workstations. Generally, audit trails and logs are subjected to audit analysis after the fact.

- Ensure that information acquired can help establish links between users and events. The AT&L service should ensure that the information acquired can be used to establish links between user activity and some event. The AT&L service needs to allow its users to acquire identifiers that represent the identity of a user uniquely and a description of their activities at the time the event was captured. This requirement is essential for accounting for user actions. Stakeholders use the provided details to determine who the actors are who engage in malicious or unwanted activity, and eventually assign the responsibility of the event to those actors.
- Ensure that information acquired is in a form that users can interpret. An AT&L service must not only capture information about events, but also ensure that the information is in a form that the user can understand. This requirement is essential for facilitating understanding of events and making informed decisions.
- Enable users to reconstruct events captured from disparate sources. Regardless of where or when parts of an event are captured, an audit trail creates a comprehensive view of the event. The audit trail may come from disparate sources, but collectively it forms a more complete view of the event. Users of the AT&L service should be able to acquire information as a single view about events even though parts of the information are gathered from multiple sources. This requirement is essential for determining what an event was, performing investigations into malicious events, and piecing together information to determine event history.
- Enable users to repeatedly examine the information derived from an event. Scrutinizing events can help address future security breaches. Audit trails and logs gathered by this service need to be generally available for all accounting mechanisms and for extended periods of time, for potential event clarification or elaboration, as necessary. This requirement is essential to support users who need to revisit events to derive more information or re-examine conclusions drawn from earlier scrutiny.
- Perform its service when needed. An AT&L service needs to be able to provide its services during times where the tracking of events is absolutely important. During operation the AT&L service is processing information about events that could cause significant damage to the organization, and the AT&L service needs to be able to continue functioning during those high-impact events. This requirement is essential to support availability and concerns the readiness of the AT&L service.
- Protect the information it captures. The AT&L service needs sufficient protection for its activity within the organization and must afford a reasonable level of protection for the information being processed. The AT&L service should ensure that information intended for authorized users is not accessible to malicious actors. The AT&L service should also ensure that the information it provides to a user retains its accuracy. This information gives decision makers insight into how well the AT&L information is protected from malicious actors and how reliable the AT&L information is to use. This requirement is essential to support confidentiality, integrity, and privacy, and concerns the trustworthiness of the information the AT&L service provides.
- Provide accountability for changes to audit trails and logs. The AT&L service should provide information about an event that resulted in unauthorized or authorized access to information that the AT&L service provides. Event information needs to include all actor identifiers and events that occurred. This requirement is essential to support accountability.

An additional set of requirements applies to all service requirements patterns. Instead of duplicating the discussion of the same set in each requirements pattern, they are simply listed here, because they do need to be considered in each requirements pattern. The requirements are: minimize time and effort to use, minimize mismatch with user characteristics, risks to user safety, costs of per-user set-up, costs of maintenance, management, and overhead, and changes needed to existing system infrastructure.

Further discussion of each of these cross-cutting requirements, including implementation factors, is given in I&A REQUIREMENTS.

## **88.6. Structure**

## **88.7. Dynamics**

## **88.8. Implementation**

This section first provides more detail on the process that was summarized in the Solution section, then discusses factors in determining the relative importance of requirements.

### **88.8.1. Process Guidelines**

The requirements process is typically performed by a system requirements engineer in conjunction with an enterprise architect, and includes several steps:

1. Establish the domain for which the AT&L service is needed. Ensure that the domain has been identified and scoped: typical AT&L domains include information system, physical facility, network, portal, or entire organization. The domain consists of at least three parts: a defined scope of actors or users, a defined scope of assets, and a defined scope or set of events that involve actions or operations on those assets. Other constraints may bound the domain—for example, the AT&L requirements for a real-time service may differ from those for a multi-year service: these might represent two domains.
2. Specify a set of factors that affect the specialization and importance of requirements. The factors include uses of AT&L, AT&L needs, organization constraints, and priorities. You can find a general candidate set of factors below.
3. Specify AT&L requirements for the target AT&L domain. To do this, specialize the set of generic requirements given in the Solution section.
4. Define the relative importance of specific requirements. You can find more details about the association of factors and requirements below.

### 88.8.2. Factors in Determining Relative Importance

Table 16 reiterates the generic requirements described in the Solution section and identifies factors for judging their relative importance to an organization or system. For each factor, the table also indicates the resulting requirement priority, in terms of High, Medium, and Low.

Generic Requirement	Factor	Resulting Priority
Acquire information about designated types of activities and events	Required by law or other mandate outside of the organization, or events involve highly sensitive or valuable assets.	High
	Internal organization concern rather than external mandate, or events involve assets of medium value.	Medium
	Only prevention approach used, not detection or response, or events involve low-value assets.	Low
Ensure that the information acquired can help establish links between users and events	Assigning responsibility is high priority, because it is required by law, or events involve highly sensitive or valuable assets.	High
	Accountability is an organization concern and not a legal or external mandate, or events involve assets of medium value, or losses are covered by insurance.	Medium
	No action will be taken against individuals, or events involve low value assets.	Low
Ensure that information acquired is in a form that users can interpret	Immediate response is needed to a critical event, and precise understanding is essential.	High
	Event responses allow for reasonable delay in reaction, or only general understanding is needed.	Medium
	Event responses are not time critical.	Low
Enable users to reconstruct events captured from disparate sources	Insurance or recoup of financial losses is critical.	High
	Organization is aware that it has events that span multiple areas.	Medium
	Events are localized or do not have multiple sources.	Low
Enable users repeatedly to examine the information derived from an event	Events and the information derived from event capture are critical to organization operations.	High
	Event information can be derived with a reasonable amount of scrutiny.	Medium
	Events are short-lived and simple.	Low
Perform its service when needed	Available AT&L is critical to event traceability.	High
	Losses due to unavailable AT&L are covered by insurance or fall within the boundaries of acceptable risk.	Medium
	No immediate need to respond to events.	Low

Protect the information it captures	Information found in audit trails is sensitive or information must be provided to an outside organization.  Information is used only internally, or the information is not sensitive.	High  Medium
Provide accountability for changes to audit trails and logs	Legal mandate to provide that information or needed for insurance purposes.  Internal organization decision determines the consequences for the malicious actors.	High  Medium

*Table 16: Audit trail and logging service requirements factors*

## 88.9. Example Resolved

Samuel the museum systems engineer defines the museum rooms where precious gems are kept as an AT&L domain. Table 17 shows the museum concerns and associated requirements priorities Samuel has specified for this domain.

Requirement	Museum Priority and Concern
Acquire information about designated types of activities and events	HIGH – The museum decision makers want to enforce AT&L services for this domain across the organization.
Ensure that information acquired can help establish links between users and events.	HIGH – The museum decision makers want information from the AT&L service to be immediately usable to substantiate user involvement with events.
Ensure that information acquired is in a form that users can interpret	MEDIUM – Obviously, the museum would want the information to be as coherent as possible, but the priority is tracking of activities. The museum would be willing to trade off users taking a bit longer to understand information against having all information available to scrutinize.
Enable users to reconstruct events captured from disparate sources	LOW – In general this is an important requirement, but in this case the museum is interested in tracking the activity of access to the badge-protected room specifically, so logs from this room are most important.
Enable users repeatedly to examine the information derived from an event	MEDIUM – The ability to scrutinize the information facilitates the tracking of activities in the long term. This is useful but not critical for this domain.
Perform its service when needed	HIGH – The museum absolutely wants to have this ability to track activities for this domain even under emergency conditions. AT&L needs to be able to demonstrate that it can do this.
Protect the information it captures	HIGH – To have trust in the tracking information, AT&L needs to demonstrate that its information can be trusted.
Provide accountability for changes to audit trails and logs	MEDIUM – The museum wants to know who changes the information, but this is less important than acquiring and protecting the information.

*Table 17: Problem example resolution for audit trails and logging requirements*

## **88.10. Consequences**

The primary benefit is the existence of a set of explicit AT&L requirements for a given system or security domain. The relative importance of the requirements is identified. You may expect the following benefits from applying this pattern:

- It facilitates the conscious selection of AT&L requirements, so that decisions about selecting AT&L mechanisms have a clear basis, rather than occurring in a vacuum.
- It promotes explicit analysis of trade-offs that encourages balancing and prioritizing of conflicting requirements and forces. This includes balancing the need for accountability with the need for privacy. This helps to avoid stronger than necessary AT&L mechanisms that would make it difficult for valid users, and at the same time it helps to avoid weaker than necessary AT&L that makes it easy for unauthorized actors to avoid.
- It results in documentation of AT&L requirements that communicates to all interested parties and is useful in comparing the adequacy of alternative implementations of AT&L services.
- The explicit requirements resulting from the pattern foster a clear connection of requirements to audit and logging policies: this also encourages organizations to make their accounting policies more explicit.

The following potential liabilities may arise from applying this pattern:

- It requires an investment of resources to apply the pattern, including time to analyze domains and AT&L needs. In some cases, the cost of applying the pattern may exceed its benefits.
- It poses a danger of possible violation of privacy rights if extensive data is captured and analyzed. You can mitigate this by capturing and analyzing the minimum amount of data, and by working closely with your legal department.
- The formal selection process may be too long and costly and produce too much overhead. You can mitigate this in the same ways as noted above.
- Specific circumstances might not be covered by generic AT&L requirements. You can mitigate this by adding specific requirements and including them in the trade-offs.
- Documentation of requirements implies that they must be maintained as they change over time. You can mitigate this by keeping the requirements in a form that is easy to update, integrated with other system documentation.
- Perception of AT&L requirements can differ throughout an organization or in a particular domain. This may make it difficult to reach agreement on the relative priorities of requirements. On the other hand, bringing such disagreements to the surface may be a benefit of the pattern, because then they can be properly discussed and resolved.

## **88.11. Known Uses**

The general AT&L requirements and the process of specifying AT&L requirements described in this pattern are widely known, but are generally used informally, as opposed to being codified or published. The requirements as stated in this pattern represent a consolidation of MITRE Corporation's experience in working with multiple customers over several decades. However, some publications on security AT&L and AT&L requirements exist. Examples are:

- ISO standard [1] discusses AT&L as one of the primary safeguards.
- [2] is an international standard that defines evaluation criteria for information technology security. It includes a class or family of criteria that address AT&L requirements, including event storage and audit trail availability

Other more general discussions of AT&L practice are available in [3,4,5,6,7,8].

## **88.12. See Also**

## **88.13. References**

- [1] "TR 13335-4:2000 Information Technology - Guidelines for the Management of IT Security - Part 4: Selection of Safeguards" International Organization for Standardization, 2000.
- [2] Organisations, C. C. P. S. (1999). Common criteria for information technology security evaluation (version 2.1). *Common Criteria Project Sponsoring Organisations*, 1-3.
- [3] Abrams, M. D., Jajodia, S. G., & Podell, H. J. (Eds.). (1995). *Information security: an integrated collection of essays*. IEEE computer society press.
- [4] Bace, R. G., & Mell, P. (2001). Intrusion detection systems.
- [5] Cugini, J. V. (2000). Web Usability Logging: Tools and Formats.
- [6] DiDuro, J., Crosslin, R., Dennie, D., Jung, P., & Louden, C. (2002). *A Practical Approach to Integrating Information Security into Federal Enterprise Architecture*. LOGISTICS MANAGEMENT INST MCLEAN VA.
- [7] National Institute of Standards and Technology (NIST). (2003, August 25). An Introduction to Computer Security: The NIST Handbook. NIST Special Publication 800-12. Information Technology Laboratory. <http://sbc.nist.gov/cyber-security-tips/800-12/800-12Word.html>
- [8] Wheeler, A. (1999, May). Payment Security and Internet References.  
<http://www.garlic.com/~lynn>

## **88.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **89. Authorization**

## **89.1. Intent**

This pattern describes who is authorized to access specific resources in a system, in an environment in which we have resources whose access needs to be controlled. It indicates, for each active entity that can access resources, which resources it can access, and how it can access them.

## **89.2. Example**

In a medical information system, we keep sensitive information about patients. Unrestricted disclosure of this data would violate the privacy of the patients, while unrestricted modification could jeopardize the health of the patients.

## **89.3. Context**

Any environment in which we have resources whose access needs to be controlled.

## **89.4. Problem**

We need to have a way to control access to resources, including information. The first step is to declare who is authorized to access resources in specific ways. Otherwise, any active entity (user, process) could access any resource and we could have confidentiality and integrity problems.

How do we describe who is authorized to access specific resources in a system?

The solution to this problem must balance the following forces:

- The authorization structure must be independent of the type of resources. For example, it should describe access by users to conceptual entities, access by programs to operating system resources, and so on, in a uniform way.
- The authorization structure should be flexible enough to accommodate different types of subjects, objects, and rights.
- It should be easy to modify the rights of a subject in response to changes in their duties or responsibilities.

## **89.5. Solution**

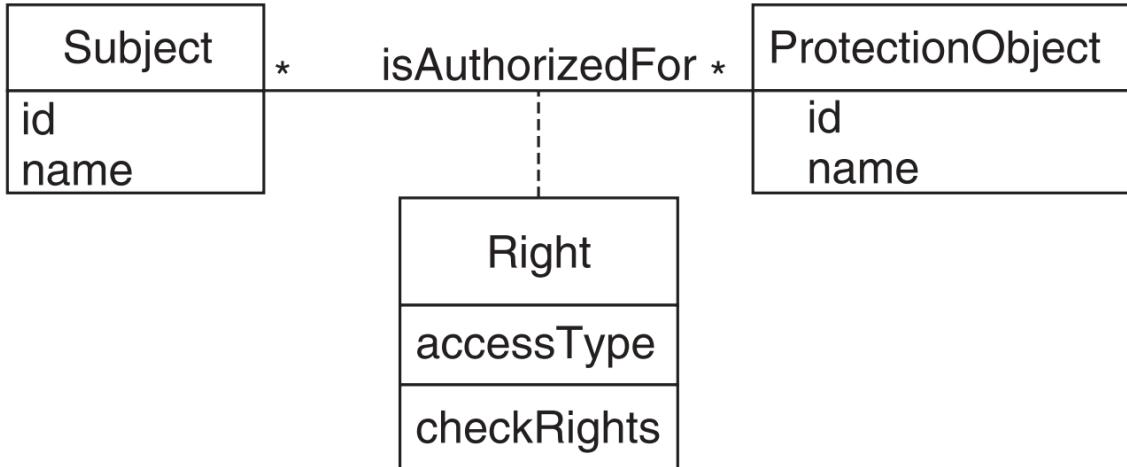
Indicate, for each active entity that can access resources, which resources it can access and how.

## **89.6. Structure**

The Subject() class describes an active entity that attempts to access a resource (Protection Object) in some way. The ProtectionObject() class represents the resource to be protected.

The association between the subject and the object defines an authorization, from which the pattern gets its name. The association class Right() describes the access type (for example, read, write) the subject is allowed to perform on the corresponding object. Through this class one can check the rights that a subject has on some object, or who is allowed to access a given object.

Figure 183 shows the elements of an authorization in form of a class diagram.



*Figure 183: Class model for AUTHORIZATION*

## 89.7. Dynamics

## 89.8. Implementation

An organization, according to its policies, should define all the required accesses to resources. The most common policy is need-to-know, in which active entities receive access rights according to their needs.

This pattern is abstract and there are many implementations: the two most common approaches are Access Control Lists and Capabilities [1]. Access Control Lists (ACLs) are kept with the objects to indicate who is authorized to access them, while Capabilities are assigned to processes to define their execution rights. Access types should be application oriented.

## 89.9. Example Resolved

A hospital using an authorization system can define rules that allow only doctors or nurses to modify patient records, and only medical personnel to read patient records. This approach allows only qualified personnel to read and modify records.

## 89.10. Consequences

The following benefits may be expected from applying this pattern:

- The pattern applies to any type of resource. Subjects can be executing processes, users, roles, user groups. Protection objects can be transactions, data, memory areas, I/O devices, files, or other resources. Access types are individually definable and can be application-specific in addition to the usual read and write.
- It is convenient to add or remove authorizations.
- Some systems separate administrative authorizations from user authorizations for further security, on the principle of separation of duties [2].
- The request may not need to specify the exact object in the rule: the object may be implied by an existing protected object [3]. Subjects and access types may also be implied. This improves flexibility at the cost of some extra processing time to deduce the specific rule needed.

The following potential liabilities may arise from applying this pattern:

- If there are many users or many objects, a large number of rules must be written.
- It may be hard for the security administrator to realize why a given subject needs a right, or the implications of a new rule.
- Defining authorization rules is not enough, we also need an enforcement mechanism.

## 89.11. Variants

The full access matrix model usually described in textbooks also includes:

- Predicates or guards, which may restrict the use of the authorization according to specific conditions
- Delegation of some of the authorizations by their holders to other subjects through the use of a Boolean ‘copy’ flag

Figure 184 extends AUTHORIZATION to include those aspects. Right now, includes not only the type of access allowed, but also a predicate that must be true for the authorization to hold, and a copy flag that can be true or false, indicating whether or not the right can be transferred. CheckRights is an operation to determine the rights of a subject or to find who has the rights to access a given object.

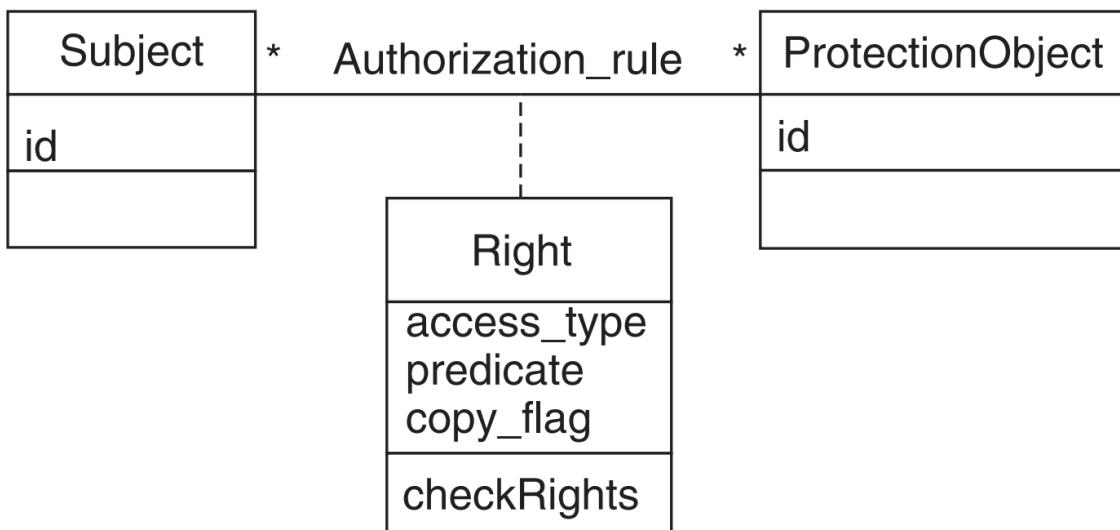


Figure 184: Extended AUTHORIZATION

## **89.12. Known Uses**

This pattern defines the most basic type of authorization rule, on which most more complex access-control models are based. It is based on the concept of access matrix, a fundamental security model [1,4]. Its first object-oriented form appeared in [5]. Subsequently, it has appeared in several other papers and products [6,7]. It is the basis for the access control systems of most commercial products, such as Unix, Windows, Oracle, and many others. PACKET FILTER FIREWALL implements a variety of this pattern in which the subjects and objects are defined by Internet addresses.

## **89.13. See Also**

ROLE-BASED ACCESS CONTROL is a specialization of this pattern. REFERENCE MONITOR complements this pattern by defining how to enforce the defined rights.

## **89.14. References**

- [1] Pfleeger, C. P. (2003). *Security in Computing* (Third Edition). Prentice-Hall.
- [2] Wood, C., & Fernandez, E. B. (1979). *Authorization in a decentralized database system*. IBM.
- [3] Fernández, E. B., Summers, R. C., & Lang, T. (1975, September). Definition and evaluation of access rules in data management systems. In *Proceedings of the 1st International Conference on Very Large Data Bases* (pp. 268-285).
- [4] Summers, R. C. (1997). *Secure computing: threats and safeguards*. McGraw-Hill, Inc..
- [5] Fernandez, E. B., Larrondo-Petrie, M. M., & Gudes, E. (1994). A method-based authorization model for object-oriented databases. In *Security for Object-Oriented Systems* (pp. 135-150). Springer, London.
- [6] Essmayr, W., Pernul, G., & Tjoa, A. M. (1998). Access controls by object-oriented concepts. In *Database Security XI* (pp. 325-340). Springer, Boston, MA.
- [7] Kodituwakku, S. R., Bertok, P., & Zhao, L. (2000). A pattern language for designing and implementing role-based access control. *Procs. KoalaPLoP 2001*.

## **89.15. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **90. Automated I&A Design Alternatives**

## **90.1. Intent**

This pattern describes alternative techniques for automated I&A, as opposed to procedural or physical I&A. It helps you to select an appropriate I&A strategy that consists of a single technique, or a combination of techniques, to satisfy I&A requirements. Techniques considered include password, biometrics, hardware token, PKI, and I&A of unregistered users.

## **90.2. Example**

Indiana Jones, a museum employee, needs to gain access to the museum intranet while collecting artifacts for the museum from around the world. He wants to check his e-mail abroad and also access the museum's database to evaluate a found artifact. From Jones' perspective, the most important requirements for this I&A service are to support I&A from remote locations and to be easy to use. From the perspective of Samuel, the museum systems engineer, the most important requirements for this I&A service are to have high accuracy, especially to reject attempts by non-employees to gain access to the intranet, and to limit I&A overhead. Samuel and his systems engineering group have used I&A REQUIREMENTS to define all four of these intranet I&A requirements as high priority. Now Ivan the intranet architect needs to select an I&A service to satisfy these requirements. The choices available to Ivan are many. They include identifier and password, PKI certificates, multiple biometrics options, and a hardware token with a one-time password. How can Ivan choose among the alternatives?

## **90.3. Context**

The person applying this pattern understands the requirements for I&A, along with their relative importance—for example, from the results of applying I&A REQUIREMENTS. A decision has been made to use automated I&A.

## **90.4. Problem**

I&A is a common need for systems and enterprises. Multiple techniques exist for achieving I&A. Different techniques emphasize different types of authenticators. No one technique is the best in all situations. Trade-offs and weighting are typically necessary, because in general the techniques have differing and often complementary strengths and weaknesses. For example, PKI provides high accuracy, but has relatively high infrastructure and cost impact, while passwords provide less accuracy, but have low infrastructure and cost impact.

In addition, certain combinations of techniques can produce an I&A strategy that in some circumstances satisfies requirements better than any of the individual techniques. For example, a combination of password and hardware token is typically stronger than either individual technique, because each compensates for a weakness of the other.

A common perspective for comparing and combining techniques is the following categorization:

- Something you know, for example a password.
- Something you have, for example a hardware token.
- Something you are, for example a biometric characteristic such as an iris image.
- Recently a fourth category has emerged: where you are, for example, derived from either your IP address or through the use of GPS, which is now included in some cell phones and PDAs. This is an additional kind of information available for authentication.

An I&A strategy may be influenced by the selection of strategies for other I&A domains within an enterprise. The enterprise may find it more efficient—in terms of cost, training, and maintenance—if all I&A domains that have similar requirements use the same strategy. For example, the enterprise may decide that the I&A used in granting out-of-hours physical access to all enterprise facilities throughout the country should use the same technique, such as biometrics.

Using a single technique for I&A in an organization is attractive, for example for achieving single sign-on (SSO). On the other hand, using a single technique is also dangerous, because it is a single point of failure, thus violating the ‘defence in depth’ principle. For example, if you are an imposter and your identity claim is accepted, you may be given access to multiple critical resources.

How can a strategy for I&A be selected that satisfies I&A requirements?

Based on the foregoing discussion, we can summarize the forces that influence selection of a strategy that balances techniques to satisfy I&A requirements:

- Some techniques satisfy some I&A requirements better than others.
- In many cases certain combinations of I&A techniques can satisfy requirements better than any individual technique. A common strategy is to combine techniques from two or more of these categories: something you know, something you have, something you are, and where you are.
- An I&A strategy may be influenced by the selection of strategies for other I&A domains within an enterprise.
- Using a single technique for I&A across an organization may be efficient, but it is also dangerous, because it is a single point of failure.

## 90.5. Solution

Systematically review the characteristics of the available I&A techniques and select a strategy that consists of one or more techniques. Proven techniques include user ID/ password, hardware token, biometrics, PKI, and I&A of unregistered users. These are not the only techniques that exist, or that will exist in the future, but they are the techniques described in this pattern.

The selection process is typically performed by a person or team serving in the role of system architect, security architect, or enterprise architect, depending on the nature and scope of the domain. The process includes several activities: explicitly assembling the necessary inputs for decision making is an important first step. Inputs include a definition of the I&A domain or scope of the strategy, I&A requirements, and the general values of factors for each I&A technique. The inputs are then used to define specific technique profiles for the chosen domain. With this information, you can compare the I&A requirements with techniques to

determine the best matches. Finally, if no individual technique adequately matches the requirements, you can look at combinations of techniques.

## 90.6. Structure

## 90.7. Dynamics

## 90.8. Implementation

This section first provides further detail on the process that was summarized in the Solution section, then presents information on technique profiles. Finally, considerations are given for combining techniques and selecting a strategy.

### 90.8.1. Process Guidelines

The selection process includes the following steps:

1. Assemble the necessary inputs for decision making. Two of the inputs are a definition of the I&A domain or scope and the I&A requirements. If you have applied the pattern, both of these inputs should be available. The requirements should include enterprise constraints, and an indication of the importance of each requirement—for example via ranking, weighting, or criticality indicators. The third input is a technique factor profile summary, that is, general values of factors for each technique. Table 18 provides a summary that you can use for certain I&A techniques.
2. Define the specific technique profiles for this domain. The next step is to specialize the general technique factor profile for your specific I&A domain. You can use the technique profiles discussion below to tailor the value of each technique in your domain. For example, if your domain excludes software actors, then satisfaction of the requirement to support a variety of user types (that is, the entry for User Types) is high for all techniques with respect to your domain.
3. Compare the I&A requirements with individual technique profiles. If one technique satisfies the requirements, select that technique as the I&A strategy: if not, perform step 4.
4. If no single technique is adequate, look at combinations of techniques. Combine techniques that have complementary strengths and weaknesses. You might benefit from the discussion of combinations and the overall organizational perspective that follows Table 18.

Requirement	User ID/Password	Biometrics	PKI	Hardware Token	Unregistered Users
Avoid confirming imposters	Med–Low	High–Med	High	Med–High	Med–Low

Avoid denying legitimate users	Med	High-Low	High	Med-High	Med-Low
User types	Med-High	Med-High	High	Med-High	Med-High
User location	High	High	Med	High	High
User mobility	High	Low-Med	High	Low-Med	High
Easy to use	Med-High	Med	Med	High-Med	High
Speed of use	High	Med-High	Med	High	High
Safety of use	High	Low-High	High	High	High
Cost effective per user	High	Low-High	Med-Low	Med-High	High
Cost effective per connection	High	Low	High	Low-High	High
Infrastructure compatibility	High	Med-Low	Low	High-Med	High
Cost effective maintenance	High	Med-Low	Med-High	Med	High
Protection for authenticators	Low-Med	Med	High	Med-High	Med
Availability	High	Med-High	Med-High	Med-High	High

Table 18: Summary of I&A technique profiles

### 90.8.2. Techniques Profiles

I&A techniques differ in what they use for IDs, identifiers, and authenticators, as well as other characteristics that affect their selection. A description of each technique is given. The purpose of this section is to define their comparative characteristics. Each I&A technique has a characteristic profile with respect to factors affecting the ability of the technique to satisfy the requirements. The profile for each technique is discussed here and summarized in Table 18.

### 90.8.3. User ID/Password

This technique generally scores high on cost effectiveness and usage requirements, but lower on reliability and protection of passwords. Password's ability to avoid confirming imposters is medium at best, because passwords can be obtained through theft or other means. This ability depends on good password practice—for example, the use of hard-to-guess passwords, and not recording passwords in easy-to-find locations. Password's ability to avoid denying legitimate users depends on the likelihood of remembering passwords: good passwords can be somewhat difficult to remember.

Regarding user types, passwords as they are typically defined may not be suitable for software actors. A common belief is that passwords are easy to use. It is true that poor password practice is easy. Good password practice is harder to achieve, but it can be made easier through schemes such as one-time passwords, which is described below.

### 90.8.4. Biometrics

The biometrics technique profile varies more than other techniques. This is due to the fact that multiple biometric techniques exist. A general profile is described here, and the various biometric techniques are described further in BIOMETRICS DESIGN ALTERNATIVES.

Biometric techniques have the potential for high reliability, depending on the type of biometric selected. On the other hand, biometric techniques generally cost more and are not as easy to use as some other techniques. Biometric techniques often do well in recognizing legitimate users. However, environmental, or aging factors may affect biometric readings. Such factors include poor lighting, sunglasses, facial hair, and change due to injury or disease.

Biometrics techniques are not suitable for software actors. Some biometric techniques may not be suitable for some types of mobile computing (for example cell phones). Safety depends on the type of biometric technique: retinal scans can cause damage to retina, so its safety is low. The cost is increased due to the need for biometric devices or scanners, as well as additional processor, storage, network loads and in some cases additional processing software. It is possible to steal biometric information, which has the potential for severe problems for a user. It is difficult and rather painful to change your biometric characteristics, such as fingerprints.

#### **90.8.5. PKI**

This technique depends somewhat on the population to which it is applied. It can score very highly on reliability with a relatively sophisticated user base but has high cost. It may not be suitable for world-wide computing—that is, from locations where communications to registration servers have low availability. You not only have to trust the third party issuing the certificates, but you also have to trust your computer hardware and software not to compromise your private keys or use weak encryption. In addition, you have to trust yourself or your employees to be able to validate the certificates and to actually do so.

Infrastructure impact is very high, including software development practices. It has moderate to high management costs, because of the third party involved. A PKI can work well with a defined user population where an established body issues certificates and carries a directory of public keys related to the individuals and organizations within that closed community. For example, it seems to work well within the Swiss medical community.

#### **90.8.6. Hardware Token**

The reliability of this technique can vary. The ability to avoid confirming imposters depends on the degree of protection of the token. Reliability is high if combined with password for use of the token. Stand-alone token ability to avoid denying legitimate users is high. If combined with password, this ability is medium, because of the possibility of mis-typed or forgotten passwords. Token techniques are not suitable for software actors. Some token types may not be suitable for some types of mobile computing (for example cell phones). Some types of tokens require moderate to high costs per connection because they use token readers, while other types may only require installation of additional software. Authenticator protection depends on users to report lost tokens.

#### **90.8.7. Unregistered Users**

This technique generally scores highly on ease of use and cost effectiveness but is low on reliability. The technique scales up to a very large user base. It is not suitable for software actors.

In Table 18, the requirements listed in the requirements column are described in detail in I&A REQUIREMENTS. The value or range of values indicates the extent to which a technique satisfies an I&A requirement. High indicates high satisfaction of the requirement, and Low indicates low satisfaction.

To determine the I&A technique(s) that will meet the I&A needs, a general approach is to compare the technique profiles with your results from applying I&A REQUIREMENTS to find a technique that is most compatible with your specific requirements.

### 90.8.8. Considerations for Combining Techniques

From Table 18 it is clear that different techniques have different strengths and weaknesses. None of the techniques resolves all forces, and each one resolves certain forces better than others. In many situations, no single technique satisfies all important requirements. However, some techniques complement others, so that certain combinations of techniques can satisfy more requirements. It is often useful to combine techniques from different categories: what you know, what you have, what you are, where you are.

A common example of combined techniques is a hardware token combined with a user ID/password. Typically, a small hand-held device is synchronized with the target system's authentication scheme and displays a one-time password (OTP). To access the target system, the user enters an assigned user ID and password, or PIN (personal identification number) followed by the OTP displayed on the hand-held device. Some implementations, such as SecurID are time-driven, that is, the OTP changes periodically, perhaps every minute. Other schemes are event-driven, using a button to press to get the next OTP. The latter have fewer problems with re-synchronization. The advantage of this strategy of combined token and password/PIN techniques is that it helps to prevent the replay of a compromised password. This combination increases accuracy by avoiding confirming imposters more than that of either individual technique. It also improves the protection of authenticators—unless of course you write the PIN on the token! This improvement is because the two-part authenticator (OTP and PIN) means that an impostor must now obtain both parts, using different means, in order to fool the system. This strategy illustrates the technique of combining something you know (the password) with something you have (the token).

### 90.8.9. Other Considerations for Selecting a Strategy

While the scope of this pattern is the selection of a single strategy for one I&A domain, this decision does not occur in a vacuum. Decisions made about strategies in other similar I&A domains within the enterprise may influence the decision for a given I&A domain. A trade-off is involved in these decisions between a homogeneous and a heterogeneous approach across the organization. In a homogeneous approach, you use the same technique everywhere. The benefits of this include ease of single sign-on (SSO), efficiency of cost, training, and technical support, and establishing a standard for future application developments. On the negative side, this approach weakens the defense in depth achieved.

In a heterogeneous approach, you explicitly choose different I&A mechanisms for different I&A domains. The primary benefit of this is stronger defense in depth. On the negative side, this approach makes SSO more difficult and loses the efficiency of cost, training, and technical support. A small example of enforced heterogeneity is the Front door product that is provided

for HTTP and FTP. Since the FTP password is sent in plain text over an unencrypted TCP connection, the software requires a password that is not the same as the password for HTTP connections that are protected by SSL connections. If these passwords were the same, the ‘weak’ FTP password would be the weak link for the (presumed) secure SSL-channel.

## 90.9. Example Resolved

How can Ivan the architect apply this pattern solution for I&A support of remote access to the museum intranet? The most important requirements for this museum I&A component are to have high accuracy, especially the ability to detect nonemployees, to be easy to use, provide strong support of I&A from remote locations, and limit overhead. Based on the technique profiles, the high accuracy requirement suggests that PKI would be best, and biometrics and tokens may also be candidate techniques. The ease of use and low overhead requirements indicate that biometrics and PKI are not good candidates. Therefore, of the individual techniques, a token appears to be the best one, but it is not optimum.

To obtain a solution closer to optimum, Ivan considers combinations. He concludes that combining password and token techniques gives the best overall match with requirements, because the combination increases accuracy and protection, as discussed above in considerations for combining techniques. The combination also achieves the ease of use desired by Indiana Jones when he needs to log in from some exotic location. Ivan therefore chooses this combination as the I&A strategy for remote access to the museum intranet.

## 90.10. Consequences

The following benefits may be expected from applying this pattern:

- The pattern fosters engineer and manager awareness of the elements of the decision needed on selecting I&A techniques.
- It facilitates conscious and informed decision making about I&A to support identified I&A requirements, as well as clear traceability to requirements
- It encourages better balance among competing I&A selection forces and factors, by matching technique profiles to requirements in the context of your specific domain. The result is increased likelihood that an I&A technique will be selected that satisfies your most important requirements.
- It provides some assistance on how you can combine I&A techniques to provide a complete I&A service.
- It facilitates broader enterprise optimization by promoting integration of I&A choices across multiple domains and systems across the enterprise.

The following potential liabilities may result from applying this pattern:

- It requires an investment of resources to apply the pattern, including time to analyze I&A mechanisms.
- This pattern focuses on certain selected I&A techniques. Using the pattern may mean that other techniques applicable to your specific domain are ignored, and a sub-optimum strategy may be selected. You can mitigate this by explicitly bringing other selected techniques into the decision process.

- Perception of identification and authentication (I&A) needs can differ throughout an organization. This may make it difficult to reach agreement on priorities of I&A and therefore difficult to select a I&A mechanism. On the other hand, bringing such disagreements to the surface may be a benefit, because then they can be properly discussed and resolved. This is true of individual strategies for a given domain. It is even more true of organization-wide coordination of I&A strategies-for example, by having different domains use different I&A techniques.

## **90.11. Known Uses**

The approach to selection of I&A described in this pattern is a consolidation of MITRE Corporation's experience in working with multiple customers over several decades. The approach is generally used informally by those customers, as opposed to being codified or published. One discussion of trade-off factors for selecting an I&A strategy is presented in [1].

The individual techniques considered in this pattern are widely known and used. Passwords have been ubiquitous for decades in information systems. Hardware tokens are often used for remote access, and a common strategy is to combine a token with a pin or password (for example, the MITRE Corporation uses this strategy). Biometrics and PKI are becoming more widely used.

## **90.12. See Also**

A discussion of trade-off factors for selecting an I&A strategy is presented in [1].

The registration or enrolment function complements this pattern. The operation of most I&A techniques, and in this pattern all techniques except UNREGISTERED USERS I&A REQUIREMENTS, require that the domain of users for which I&A is to be performed must first be registered or enrolled, to obtain the independent user information.

## **90.13. References**

[1] Smith, R. E. (2001). *Authentication: from passwords to public keys*. Addison-Wesley Longman Publishing Co., Inc.

## **90.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# 91. Controlled Execution Environment

## 91.1. Intent

If a process execution environment is uncontrolled, processes can scavenge information by searching memory and accessing the disk drives where files reside. They might also take control of the operating system itself, in which case they have access to everything. Use AUTHORIZATION to define the rights of a subject. From these rights we can set up the rights of processes running on behalf of the subject. Process requests are validated by CONTROLLED OBJECT MONITOR or REFERENCE MONITOR respectively.

## 91.2. Example

Jim the hacker discovers that the customer's files have authorizations and cannot be accessed directly, so he tries another approach. He realizes that processes are not given only the rights of their owners, but also have rights with which they can access memory and other resources belonging to other users. He systematically searches areas of memory and I/O devices being used by other processes until he can scavenge a few credit card numbers that he can use in his illicit activities.

## 91.3. Context

A process executes on behalf of a user or role (a subject). A process must have access rights to use these resources during execution. The set of access rights given to a process define its execution domain. Processes must be able to share resources in a controlled way. The rights of the process are derived from the rights of its invoker.

## 91.4. Problem

Even if direct access to files is restricted, users can do 'tunneling,' attacking them through a lower level. If the process execution environment is uncontrolled, processes can scavenge information by searching memory and accessing disk drives. They might also take control of the operating system itself, in which case they have access to everything.

The solution to this problem must resolve the following forces:

- We need to constrain the execution of processes and restrict them to use only resources that have been authorized based on the rights of the activator of the process.
- Subjects can be users, roles, or groups. We want to deal with them uniformly.
- Resources typically include memory and I/O devices but can also be files and special instructions. We want to consider them in a uniform way.
- A subject may need to activate several processes, and a process may need to create multiple domains. Execution domains may need to be nested. We want flexibility for our processes.

- Typically, only a subset of a subject's rights needs to be used in a specific execution. We need to provide to a process only the rights it needs during its execution (on the principle of least privileges).
- The solution should put no constraints on implementation.

## 91.5. Solution

Use the AUTHORIZATION pattern to define the rights of a subject. From these rights, we can set up the rights of processes running on behalf of the subject. Process requests are validated by CONTROLLED OBJECT MONITOR or REFERENCE MONITOR respectively.

## 91.6. Structure

Figure 185 shows the UML class diagram of CONTROLLED EXECUTION ENVIRONMENT. This model combines AUTHORIZATION, EXECUTION DOMAIN, and REFERENCE MONITOR to let processes operate in an environment with controlled actions based on the rights of their invoker. Process execution follows EXECUTION DOMAIN—as a process executes it creates one or more domains. Domains can be recursively composed. The descriptors used in the process' domains are a subset of the authorizations that the subject has for some ProtectionObjects (defined by an instance of AUTHORIZATION). ProtectionObject is a superclass of the abstract Resource class, and ConcreteResource defines a specific resource. Process requests go through a ReferenceMonitor that can check the domain descriptors for compliance.

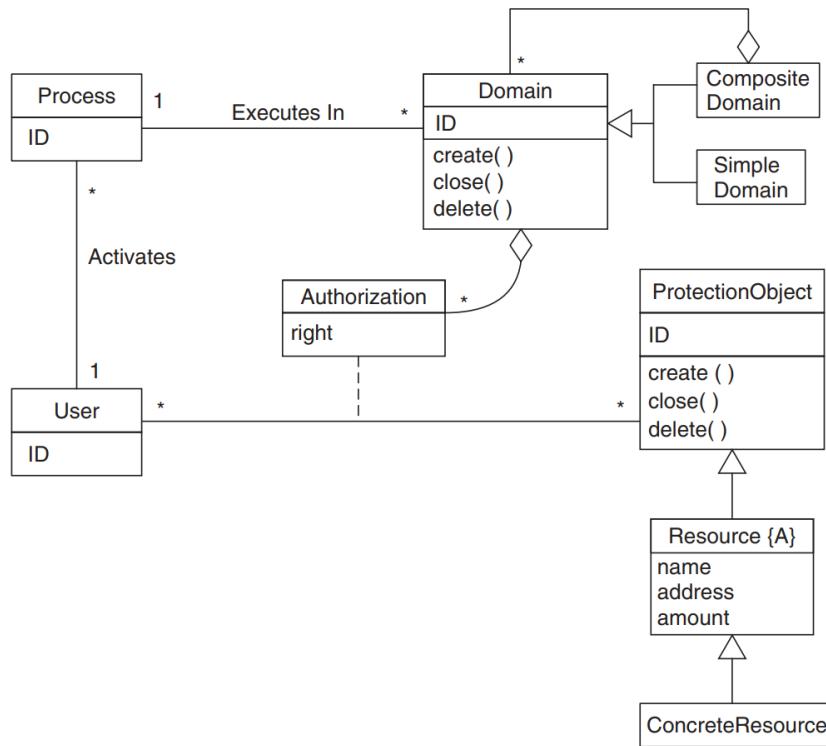


Figure 185: Class diagram for CONTROLLED EXECUTION ENVIRONMENT

## 91.7. Dynamics

Figure 186 shows a sequence diagram representing the use of a right after entering a domain. Here X denotes a segment requested by the process. An instance of REFERENCE MONITOR controls the process requests. This diagram assumes that the descriptors of the domain have been previously set up.

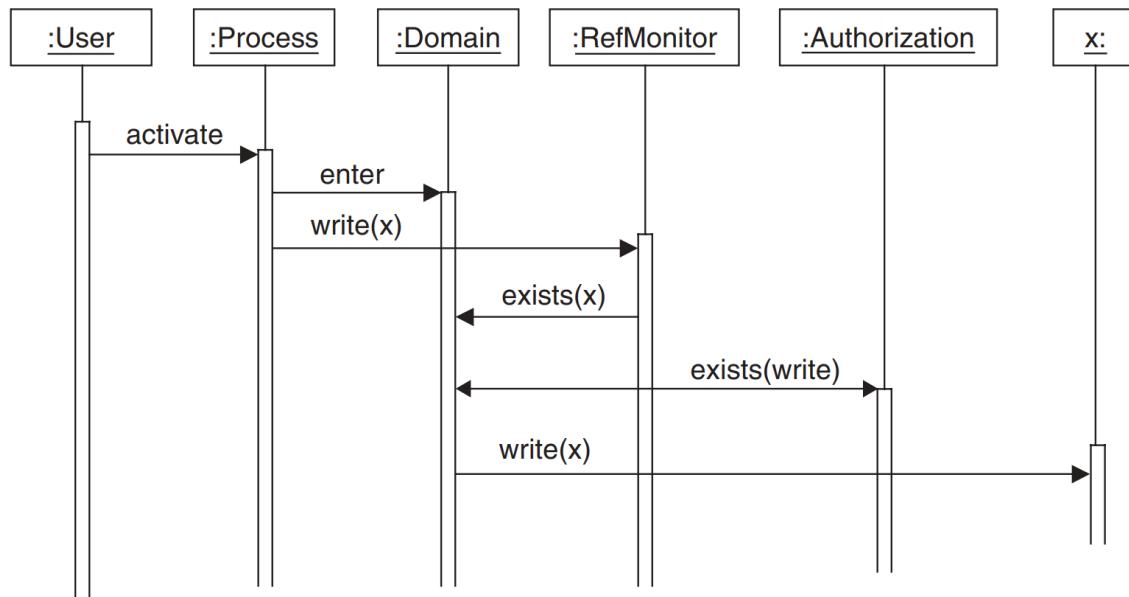


Figure 186: Sequence diagram for entering a domain and using a right in that domain

## 91.8. Implementation

### 91.9. Example Resolved

A new operating system was installed, with mechanisms to make processes operate with the rights of their activator. Jim does not have access to customer files, which makes his processes also unable to access these files. Now he cannot scavenge in other users' areas, so his illicit actions are thwarted.

### 91.10. Consequences

The following benefits may be expected from applying this pattern:

- We can apply the principle of least privileges to processes based on the rights of their activators. This also provides accountability.
- It can be applied to any type of resource.
- Subjects may activate any number of processes, and processes may have several execution domains.
- The same structure can also provide fault tolerance [1].
- Execution domains are defined according to DOMAIN and may include any subset of the subject's rights.

The following potential liabilities may arise from applying this pattern:

- Some extra complexity and performance overhead may be required.
- It can be dependent on the hardware architecture.

### **91.11. Known Uses**

The IBM S/38, the IBM S/6000 running AIX, the Plessey 250 [1], and EROS [2] have applied this pattern using capabilities. Property descriptor systems such as the Intel architectures may use this approach although their operating systems not always do so.

### **91.12. See Also**

This pattern uses AUTHORIZATION, EXECUTION DOMAIN, and REFERENCE MONITOR. CONTROLLED VIRTUAL ADDRESS SPACE pattern may be indirectly used by EXECUTION DOMAIN.

### **91.13. References**

- [1] Hodges, K. H. (1973). A fault-tolerant multiprocessor design for real-time control. *Computer Design*, 12(12), 75-81.
- [2] Shapiro, J. S., & Hardy, N. (2002). EROS: A principle-driven operating system from the ground up. *IEEE software*, 19(1), 26-33.

### **91.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

## **92. Biometrics Design Alternatives**

### **92.1. Intent**

This pattern aids the selection of appropriate biometric mechanisms to satisfy I&A requirements. Biometric mechanisms considered are face recognition, finger image, hand geometry, iris recognition, retinal scanning, signature verification, and speaker verification. Additional mechanisms, including DNA, are identified for completeness.

### **92.2. Example**

The internal maintenance and research areas of the new gemstone wing of the museum essentially afford staff access directly to high-value assets and to the information on those assets. While physical entry for these activities is being tightly controlled, access to sensitive asset information must also be restricted. To gain access to the Web server with strictly controlled asset information, staff are required to log-on to the Web server. Part of the log-on process will be use of a biometric to provide additional verification of employee identities. Alvin the system architect must determine which biometric mechanism is most appropriate for the museum.

### **92.3. Context**

The person applying this pattern understands the requirements for I&A, along with their relative importance, for example from the results of applying I&A REQUIREMENTS.

A decision has been made to use biometrics for I&A, for example from the results of applying AUTOMATED I&A DESIGN ALTERNATIVES, but which biometrics technique to use has not been decided. The decision to use some form of biometrics is typically made in the context of a user population of limited size, because of the enrolment effort required.

### **92.4. Problem**

Each technique has different strengths and weaknesses, which are described in the Implementation section. Therefore, no one technique or combination of techniques is best for all enterprises. Decisions are needed to determine the best biometric mechanisms for the given purpose.

It should be noted that biometrics, at least in human-readable form, have been available for a long time, even before the term ‘biometrics’ was used. For example, badges, licenses, and passports have often included photographs as well as physical characteristics such as height and eye color. Fingerprints have long been used in criminal justice and other security contexts.

Given that biometrics has been selected to perform some I&A purpose, what biometric mechanisms would best satisfy this purpose?

Selection of appropriate biometrics mechanisms needs to resolve the following forces:

- Some biometric information can be stolen, for example, by obtaining and using pictures, images, imprints, or other models of another person's biometric information.
- Biometric information can be erroneously associated with the wrong identity at enrollment: for example, actor B can enroll his biometric information with actor A's identity.
- Stolen or erroneously enrolled biometric information can be used to masquerade as another person, which leads to false acceptance.
- Some biometric measurements can vary due to environmental conditions, or can change over time due to age, or can change quickly due to injury, surgery, or other significant episode. Such variations can lead to false rejection.
- False acceptance can lead to unauthorized access to assets, in cases in which an access control service relies on the biometric mechanism's results.
- False acceptance can lead to lack of accountability, in cases in which an accounting service such as audit relies on biometric mechanism results. If actor B successfully masquerades as actor A, then actor A is erroneously held accountable for the actions of actor B.
- False rejection can lead to reduced productivity and increased user frustration.
- False rejection can also lead to lack of accountability. For example, actor A may take steps to change certain biometric characteristics via surgery with the goal of being falsely rejected as actor A. This may allow him to avoid accountability for an action such as a serious crime.
- In general, low false acceptance rate (FAR) and low false rejection rate (FRR) are conflicting goals: configuring a biometric mechanism to achieve a very low FAR tends to increase the FRR. Conversely, achieving a very low FRR tends to increase the FAR. When comparing biometric systems, a low FAR is most important when security is the priority. On the other hand, a low FRR is most important when convenience is the priority. [1] discusses the inverse relation between these two error types.
- Some biometric mechanisms cost more than others.
- Some biometric mechanisms require more equipment and changes to the infrastructure than others.
- Some biometric mechanisms are less safe than others.
- Enterprise-wide optimization affects selection of biometric techniques. An enterprise may find it more efficient—for example, for cost, training, and maintenance reasons—if all I&A domains that select biometrics use the same biometrics technique. For example, an enterprise may decide that the biometrics used in granting physical access to all enterprise facilities throughout the country should use the same technique. Therefore, the selection of specific mechanisms may be a significant decision.

## 92.5. Solution

Systematically review the characteristics of available biometric mechanisms or techniques and select a mechanism. Several well-known biometrics mechanisms exist. Different mechanisms have different strengths and weaknesses and emphasize different characteristics. Each technique resolves each force to a different degree than the others. The solution provides information about alternative biometric mechanisms that is intended to help differentiate them and to help select the best technique for a given purpose, enterprise, and I&A use.

All biometric techniques can be used for verification, but only a few are capable of performing identification, especially in a large population of users or actors. This is because the task of matching a live sample with one designated template is much simpler than finding a template from a large number of possible templates. According to [2], the only biometric mechanisms with the capability to operate realistically in identification mode are finger image, iris recognition, retinal scan, and, to a lesser degree, facial scan.

## 92.6. Structure

Table 19 shows elements of the structure of this solution. Required capabilities and properties in the first column are derived from the general I&A REQUIREMENTS pattern. Specialized selection criteria in the second column are additional factors related specifically to biometric mechanisms. Together, requirements and specialized criteria drive the selection of biometric mechanisms listed in the third column. Specialized criteria are further explained in the Implementation section.

Required Capabilities/Properties	Specialized Selection Criteria	Biometric Mechanisms
<ul style="list-style-type: none"> <li>• Avoid false positives</li> <li>• Avoid false negatives</li> <li>• Variety of user types</li> <li>• Variety of user locations</li> <li>• Variety of user mobility</li> <li>• Easy to use</li> <li>• Speed to use</li> <li>• Safety of use</li> <li>• Cost effective</li> <li>• Compatible with infrastructure</li> <li>• Able to protect authenticators</li> <li>• Provide availability of process</li> </ul>	<ul style="list-style-type: none"> <li>• Devices needed</li> <li>• Obtrusiveness</li> <li>• Accuracy</li> <li>• Resistance to attack (secure)</li> <li>• Public acceptance</li> <li>• Biometric long-term stability</li> <li>• Potential interference</li> <li>• Template size</li> </ul>	<ul style="list-style-type: none"> <li>• Face recognition</li> <li>• Finger image</li> <li>• Hand geometry</li> <li>• Iris recognition</li> <li>• Retinal scanning</li> <li>• Signature verification</li> <li>• Speaker verification</li> </ul>

Table 19: Elements of biometrics design solution structure

## 92.7. Dynamics

This section describes the steps in the process of applying the pattern. Biometrics I&A inputs, including domain definition and requirements, are assembled first. Next, the specific characteristics of each biometric technique are defined, followed by selecting the best individual technique. If this technique is to be used as a stand-alone I&A mechanism, the process is then complete. If the technique is to be combined with another I&A technique—typically a non-biometrics technique, the combined strategy defined, for example, by AUTOMATED I&A DESIGN ALTERNATIVES—then the selected biometric technique must be integrated with the other technique to form an integrated I&A solution.

## 92.8. Implementation

The description and characteristics of biometric mechanisms provided here is intended to help select appropriate biometrics for a specific context. Differentiating factors include degree of accuracy, ease of use, processing speed, and size of template—the amount of data to be captured and processed. The set of biometrics, and definition of each, are obtained primarily from [3].

Each technique is classified as being based on either a physical or a behavioral characteristic. In the set identified here, the behavioral biometrics are signature verification and speaker verification, although the latter is part behavioral and part physical. The remaining biometric techniques are classified as physical.

Additional biometric techniques that exist include:

- DNA, which carries the unique genetic instructions for an individual
- Keystroke dynamics, the typing rhythm when a user types onto a keyboard, ear shape, the outer ear, lobes, bone structure
- Finger geometry, the shape, and dimensions of one or more fingers
- Palm geometry, the shape of the lines on the palm of the hand
- Veincheck/Vein tree, which uses pattern of veins in the back of the hand

We do not consider these techniques further in this pattern because they are not yet commonly used for I&A. Keystroke dynamics shows promise but has not yet reached a high level of accuracy.

Characteristics of the more common biometric mechanisms are summarized in Table 20. The value indicates the extent to which a technique satisfies a requirement for a particular factor. ‘High’ indicates high satisfaction of the factor, ‘Low’ indicates low satisfaction, and so on.

The potential interference factor identifies conditions that can inhibit successful operation of the mechanism. In general, one has to consider the basic characteristic of the concrete implementation of the technique. For example, background noise can interfere with voice recognition, or poor lighting can interfere with face recognition.

To determine the biometric mechanism that will best satisfy the biometric’s purpose, you can compare the technique profiles with your results of applying I&A REQUIREMENTS to find a mechanism that is most compatible with your specific requirements.

Technique Factor	Face	Finger	Hand	Iris	Retina	Signature	Voice
Accuracy	High	High	Medium/ High	Very high	Very high	Medium	Medium
Ease of use	Medium	High	High	Medium	Low	High	High
Resistant to attack, secure	Medium	High	High	Very high	Very high	Medium	Medium
Public acceptance	Medium/ High	Medium	High	Medium	Medium	Very high	High
Long-term stability	Medium	High	Medium	High	High	Medium	Medium

Potential interference	Lighting, aging, glasses, hair	Dryness, dirt, age, race	Hand injury, age	Poor lighting	Glasses	Changing signatures	Noise, colds, weather
Safety	High	High	High	High	Medium	High	High

Table 20: Characteristics of common biometrics techniques

### 92.8.1. Characteristics of Each Biometric Mechanism

The more common techniques are now described in more detail, especially the features and considerations that affect their selection. The details are obtained primarily from [4].

Table 21 describes the characteristics of face recognition, which is a physical biometric technique that analyzes distinguishing facial features.

Capture Devices	Features (+)	Considerations (-)
Still camera, video, thermal imaging	<ul style="list-style-type: none"> <li>Can use standard video camera input</li> <li>Can be used passively (unobtrusively) and with existing photo databases</li> <li>Socially acceptable</li> <li>Compatible with existing ID systems such as driver's license, passport</li> </ul>	<ul style="list-style-type: none"> <li>Can be affected by lighting and sometimes by skin tone, eyeglasses, facial hair, or expression</li> <li>Twins harder to distinguish</li> <li>Changes over time may require update/adaptation</li> <li>Occasional religious objections and recent privacy objections to covert use</li> <li>600–3500-byte template size</li> </ul>

Table 21: Face recognition

Table 22 describes the characteristics of finger image, which is a physical biometric technique that looks at the patterns found on the tip of the finger. Finger images may be captured by placing a finger on a scanner, or by electronically scanning inked impressions on paper. It is one of the oldest biometric approaches.

Capture Devices	Features (+)	Considerations (-)
Usually a small reader (sensor) embedded within a stand-alone device or a peripheral, such as a keyboard, PCMCIA card or mouse.  Sensor types include optical, silicon chip, ultrasonic	<ul style="list-style-type: none"> <li>Significant proven use since largely easy to use and very quick</li> <li>Relatively high accuracy</li> <li>Variety of applications and products from numerous vendors</li> </ul>	<ul style="list-style-type: none"> <li>Requires dedicated device</li> <li>A small percentage of population have poor images due to injury, disease, or occupation</li> <li>Dry skin can reduce accuracy</li> <li>Some lingering criminal connotations</li> <li>Overt action generally required, somewhat obtrusive</li> <li>250 B–1 Kbytes template size</li> </ul>

Table 22: Finger image

Table 23 describes the characteristics of hand geometry, which is a physical biometric technique that involves analyzing and measuring the shape of the hand from a 3-D perspective. This is one of the oldest biometric approaches.

Capture Devices	Features (+)	Considerations (-)
Hand reader, including camera	<ul style="list-style-type: none"> <li>• Ease of use, fast capture, and processing</li> <li>• Very small template size (~9 bytes)</li> <li>• Outdoor environments</li> </ul>	<ul style="list-style-type: none"> <li>• Requires bulky device</li> <li>• Only moderate differentiation and accuracy</li> <li>• Used mostly for verification, not identification</li> </ul>

Table 23: Hand geometry

Table 24 describes the characteristics of iris recognition. This is a physical biometric technique that analyses iris features found in the colored ring of tissue that surrounds the pupils.

Capture Devices	Features (+)	Considerations (-)
Cameras, standard video technology	<ul style="list-style-type: none"> <li>• Highly accurate, highly differentiating (each eye averages 266 unique features)</li> <li>• Can support identification as well as verification</li> <li>• Very stable over lifetime</li> <li>• Passive collection (non-obtrusive)</li> <li>• Not affected by common eye surgical procedures</li> </ul>	<ul style="list-style-type: none"> <li>• Requires dedicated device (some dual-use devices are available)</li> <li>• Mirrored sunglasses can interfere</li> <li>• Affected by some eye diseases such as cataracts</li> <li>• Limited focal length (4" to 3'), depending on device</li> <li>• 500-byte template size</li> </ul>

Table 24: Iris recognition

Table 25 describes the characteristics of retinal scanning. This is a physical biometric technique that analyses the layer of blood vessels situated at the back of the eye.

Capture Devices	Features (+)	Considerations (-)
Low intensity light source (laser) with optical coupler	<ul style="list-style-type: none"> <li>• High accuracy and stability, difficult to falsify</li> <li>• Minimal alignment and focus problems</li> <li>• Can support identification as well as verification</li> </ul>	<ul style="list-style-type: none"> <li>• User interface generally considered intrusive and uncomfortable</li> <li>• Safety concerns, possible damage if laser intensity too high</li> <li>• Capture can take several seconds</li> <li>• Devices still somewhat expensive</li> <li>• 96-byte template size</li> </ul>

Table 25: Retinal scanning

Table 26 describes the characteristics of signature verification. This is a behavioral biometric technique that analyses the way someone signs their name. The signing features such as speed, velocity and pressure exerted by the hand are as important as the static shape of the finished signature.

Capture Devices	Features (+)	Considerations (-)

Signature or graphics tablets, special pens	<ul style="list-style-type: none"> <li>Non-intrusive, natural act, highly acceptable</li> <li>Particularly compatible with financial or legal transactions, orders, document signing</li> <li>Many can already use built in graphics devices, such as those in PDAs</li> <li>Can work with Arabic lettering or Asian characters</li> </ul>	<ul style="list-style-type: none"> <li>Requires multiple consistent captures for enrolment</li> <li>Can be affected by behavioral factors such as stress, distractions</li> <li>May change over time, require update/adaptation</li> <li>Best used in 1:1 contexts, that is, verification, not identification</li> <li>1–3 Kbyte template size</li> </ul>
---	--	---

Table 26: Signature verification

Table 27 describes the characteristics of speaker verification. This is a part physical, part behavioral biometric that analyses patterns in speech. It compares live speech with a previously created speech model of a person's voice.

Capture Devices	Features (+)	Considerations (-)
Audio capture devices (sound cards, microphones)	<ul style="list-style-type: none"> <li>Socially acceptable and non-intrusive</li> <li>Can use standard handset, sound cards, microphones, over existing audio channels such as telephone lines</li> <li>Can be combined with challenge/response mechanisms</li> <li>Algorithms are typically language independent</li> <li>Generally, cannot be defeated by tape recordings or mimics</li> </ul>	<ul style="list-style-type: none"> <li>Can be affected by illness, stress, or background noise</li> <li>Can be susceptible to high quality digital audio playback attack</li> <li>Requires similar microphones for enrolment and verification</li> <li>May change over time, require update/adaptation</li> <li>Best used in 1:1 contexts, that is, authentication, not identification</li> <li>6 Kbyte template size</li> </ul>

Table 27: Speaker verification

### 92.8.2. Combining Mechanisms

If the purpose of the selected biometric mechanism is to perform verification, then the mechanism may need to be combined with a non-biometric I&A technique for a full I&A solution. The recommendation in this case is to apply AUTOMATED I&A DESIGN ALTERNATIVES if you have decided to use only automated I&A, either prior to, concurrent with, or subsequent to, the application of BIOMETRICS DESIGN ALTERNATIVES.

### 92.8.3. Selecting a Biometric Mechanism

While the scope of this pattern is the selection of a biometric mechanism for one I&A use, this decision does not occur in a vacuum. Decisions made on biometric mechanisms for other similar I&A uses within the enterprise may influence the decision for a given biometric approach. In addition, more than one biometric may be needed and consideration will need to be given to the interaction of those mechanisms.

## 92.9. Example Resolved

Alvin the system architect determines that for the museum, part of the log-on process will use a biometric to provide additional verification of employee identities. The museum wants at least high confidence with regard to the accuracy, ease of use, and resistance to attack of the biometric selected. Only the iris scanning, and fingerprint approaches can provide high confidence for those important criteria, as shown in Table 28.

To address the concerns of their staff, Alvin chooses fingerprint detection as the preferred biometric mechanism, as the technology is known to be safe and easy to use, and the potential interference factors are not expected to be extreme for this environment.

Technique Factor	Iris	Finger
Accuracy	Very high	High
Easy to use	Medium	High
Resists attack (secure)	Very high	High
Public accepts	Medium	Medium
Long-term stability	High	High
Potential interference	Poor lighting	Dryness, dirt, age, race
Safety	Medium	High

Table 28: Museum resolution for biometrics

## 92.10. Consequences

The following benefits may be expected from applying this pattern:

- It fosters engineer awareness of the elements of the decisions needed for selecting biometrics techniques.
- It facilitates conscious and informed decision making about biometrics to support identified identification and authentication service needs.
- It encourages better balance among competing biometrics selection forces and factors, including the inherent trade-off between the rates of false acceptance and false rejection, as well as theft, environmental impact, cost, and infrastructure impact. The result is increased likelihood that a biometrics technique will be selected that satisfies the most important requirements.
- It provides some assistance about how you can combine biometrics with other mechanisms to provide a complete I&A service.
- It facilitates broader enterprise optimization by promoting integration of biometrics choices across multiple domains and systems across the enterprise.

The following potential liabilities may result from applying this pattern:

- It requires an investment of resources to apply the pattern, including time to analyze biometrics mechanisms.
- Perception of identification and authentication (I&A) needs can differ throughout an organization. This may make it difficult to reach agreement on priorities for I&A, and therefore difficult to select a biometric mechanism. On the other hand, bringing such disagreements to the surface may be a benefit, because then they can be properly discussed and resolved.

- Although biometric techniques work well today for authentication with a given ID, the techniques are less reliable for identification from a large user base. This point is often neglected by decision makers.
- Users and organizations may have a false sense of increased security because they are using technology that is more expensive and more sophisticated. The cautions of this pattern over theft and other limitations of biometrics may not overcome the general perception promoted in some of the literature that biometrics is infallible.
- The enrolment process for biometrics can be expensive, because its users need to provide samples in a protected environment, otherwise an imposter might be able to submit their sample under a false identity.
- If the biometrics sensor and the storage of the templates or the checking mechanism are coupled by a network, an intruder can either steal valid samples or templates for later misuse or can perform a denial-of-service attack.

## **92.11. Known Uses**

Use of biometrics techniques is increasing, but the decision process for deciding among biometrics alternatives is generally tacit and informal, as opposed to being codified or published. However, discussion of the characteristics of various biometrics techniques does exist. [5] describes common characteristics and processes for biometrics I&A, as well as security of biometrics information. [4] and [1] provide more details of variations among biometrics techniques and are the sources of much of the implementation information in this pattern.

## **92.12. See Also**

After applying this solution, the next step typically is to apply the selected technique, which might be any of these:

- FACE RECOGNITION [6]
- FINGER IMAGE [6]
- HAND GEOMETRY [6]
- IRIS RECOGNITION [6]
- RETINAL SCANNING [6]
- SIGNATURE VERIFICATION [6]
- SPEAKER VERIFICATION [6]

## **92.13. References**

[1] Liu, S., & Silverman, M. (2001). A practical guide to biometric security technology. *IT Professional*, 3(1), 27-32.

[2] Ashbourn, J. (2000). The distinction between authentication and identification. Paper available at the Avanti Biometric Reference Site.  
<http://www.avanti.1to1.org/authenticate.html>

[3] Association for Biometrics (AfB), & International Computer Security Association (ICSA). (1999). Glossary of Biometric Terms. <http://www.afb.org.uk/docs/glossary.htm>

- [4] Tilton, C. (2002, December 9). Understanding Biometric Technology and Its Implementation. Tutorial M2. 18th Annual Computer Security Applications Conference. Las Vegas, USA.
- [5] Smith, R. E. (2001). *Authentication: from passwords to public keys*. Addison-Wesley Longman Publishing Co., Inc.
- [6] Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

## **92.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# 93. Controlled Object Monitor

## 93.1. Intent

This pattern addresses how to control access by a process to an object. Use a reference monitor to intercept access requests from processes. The reference monitor checks whether the process has the requested type of access to the object.

## 93.2. Example

Our operating system does not check all user requests to access resources such as files or memory areas. A hacker discovered that some accesses are not checked and was able to steal customer information from our files. He also left a program that randomly overwrites memory areas and produces serious disruption to the other users.

## 93.3. Context

An operating system that consists of many users, objects that may contain sensitive data, and where we need to have controlled access to resources.

## 93.4. Problem

When objects are created, we define the rights processes have to them. These authorization rules or policies must be enforced when a process attempts to access an object.

The solution to this problem must resolve the following forces:

- There may be many objects with different access restrictions defined by authorization rules: we need to enforce these restrictions when a process attempts to access an object
- We need to control different types of access, or the object may be misused

## 93.5. Solution

Use a REFERENCE MONITOR to intercept access requests from processes. The REFERENCE MONITOR checks whether the process has the requested type of access to the object according to some access rule.

## 93.6. Structure

Figure 187 shows the class diagram for this pattern. This is a specific implementation of REFERENCE MONITOR. The modification shows how the system associates the rules to the secure object in question.

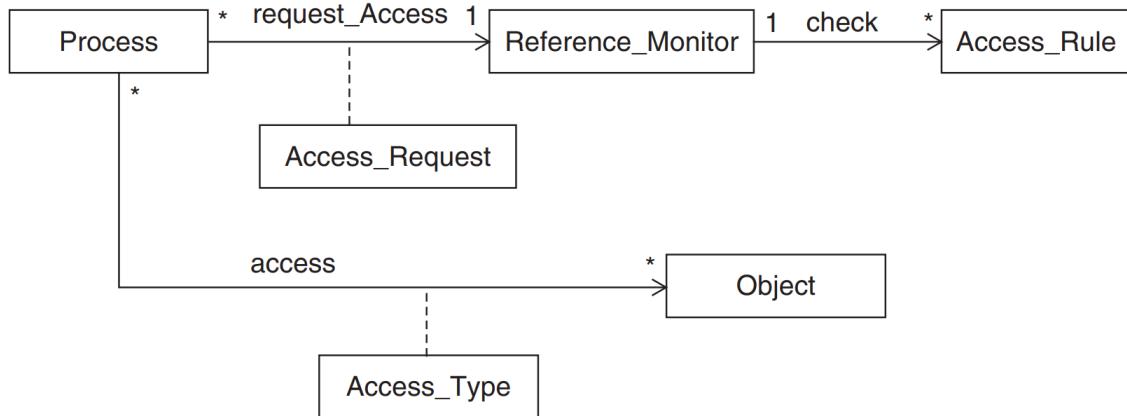


Figure 187: Class diagram for CONTROLLED OBJECT MONITOR

### 93.7. Dynamics

Figure 188 shows the dynamics of secure subject access to a secure object. Here the request is sent to the REFERENCE MONITOR where it checks the Access Rules. If the access is allowed, it is performed, and result returned to the subject. Note that here, a handle or ticket is returned to the Subject so that future access to the secure object can be directly performed without additional checking.

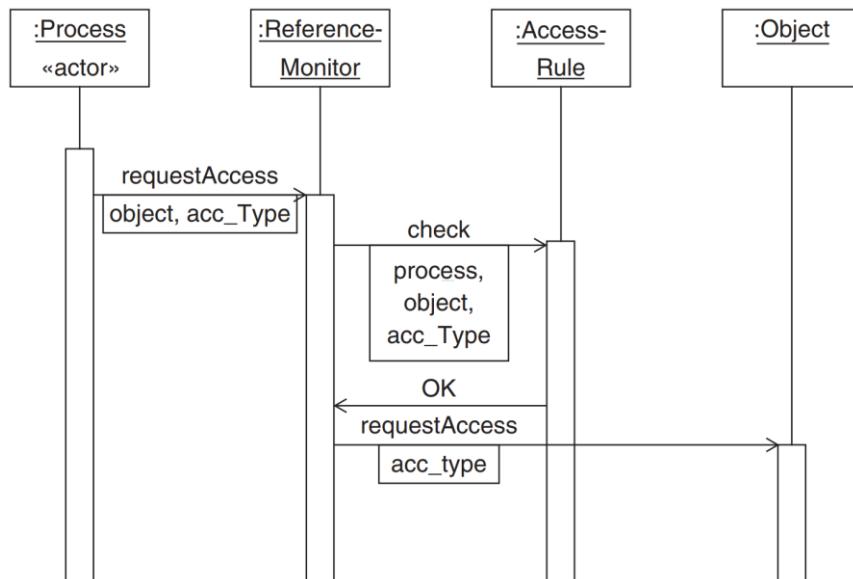


Figure 188: Sequence diagram for validating an access request

### 93.8. Implementation

### **93.9. Example Resolved**

A REFERENCE MONITOR mediates all requests. There are now no unchecked requests, so a hacker cannot get access to unauthorized files or memory areas.

### **93.10. Consequences**

The following benefits may be expected from applying this pattern:

- Each access request can be intercepted and accepted or rejected depending on the authorization rules.
- The access rules can implement an access matrix defining different types of access for each subject. We can add content-dependent rules if required.

The following potential liabilities may arise from applying this pattern:

- There is a need to protect the authorization rules. However, the same mechanism that protects resources can also protect the rules.
- There is an overhead involved in controlling each access. This is especially heavy for content-dependent rules. However, some accesses may be compiled for efficiency.

### **93.11. Known Uses**

Windows NT. The Windows NT security subsystem provides security using the patterns described here. It has the following three components [1,2,3]:

- Local Security Authority
- Security Account Manager
- Security Reference Monitor

The Local Security Authority (LSA) and Security Account Manager (SAM) work together to authenticate the user and create the user's access token. The security reference monitor runs in kernel mode and is responsible for the enforcement of access validation. When an access to an object is requested, a comparison is made between the file's security descriptor and the Secure ID (SID) information stored in the user's access token. The security descriptor is made up of Access Control Entries (ACE's) included in the object's Access Control List (ACL). When an object has an ACL the SRM checks each ACE in the ACL to determine if access is to be granted. After the Security Reference Monitor (SRM) grants access to the object, further access checks are not needed, as a handle to the object that allows further access is returned the first time.

Types of object permissions are no access, read, change, full control, and special access. For directory access, the following are added: list, add, and read.

Windows use the concept of a handle for access to protected objects within the system. Each object has a Security Descriptor (SD) that contains a Discretionary Access Control List (DACL) for the object. Each process has a security token that contains an SID which identifies the process. This is used by the kernel to determine whether access is allowed. The ACL contains Access Control Entries (ACE's) that indicate what access is allowed for a particular process SID. The kernel scans the ACL for the rights corresponding to the requested access.

A process requests access to the object when it asks for a handle using, for example, a call to CreateFile(), which is used both to create a new file or open an existing file. When the file is created, a pointer to an SD is passed as a parameter. When an existing file is opened, the request parameters, in addition to the file handle, contain the desired access, such as GENERIC\_READ. If the process has the desired rights for the access, the request succeeds and an access handle is returned, so that different handles to the same object may have different accesses [1].

Once the handle is obtained, additional access to read a file will not require further authorization. The handle may also be passed to another trusted function for further processing.

Java 1.2 Security. The Java security subsystem provides security using the patterns described here. The Java Access Controller builds access permissions based on permission and policy. It has a checkPermission method that determines the codesource object of each calling method and uses the current Policy object to determine the permission objects associated with it. Note that the checkPermission method will traverse the call stack to determine the access of all calling methods in the stack. The java.policy file is used by the security manager that contains the grant statements for each codesource.

### **93.12. See Also**

The REFERENCE MONITOR is the pattern from which this pattern is derived.

### **93.13. References**

[1] Hart, J. M. (1997). Win32 System Programming (Second Edition). Addison-Wesley Professional.

[2] Kelley, M., & Mayson, W. (1997). *Window's NT Network Security: A Manager's Guide*. LAWRENCE LIVERMORE NATIONAL LAB CA.

[3] Microsoft Corporation. (2000). Windows 2000 Security Technical Reference. Microsoft Press.

### **93.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# 94. Controlled Process Creator

## 94.1. Intent

This pattern addresses how to define and grant appropriate access rights for a new process.

## 94.2. Example

Most operating systems create a process with the same rights as its parent. If a hacker can trick an operating system into creating a child of the supervisor process, this runs with all the rights of the supervisor.

## 94.3. Context

An operating system in which processes or threads need to be created according to application needs.

## 94.4. Problem

A user executes an application composed of several concurrent processes. Processes are usually created through system calls to the operating system [1]. A process that needs to create a new process gets the operating system to create a child process that is given access to some resources. A computing system uses many processes or threads. Processes need to be created according to application needs, and the operating system itself is composed of processes. If processes are not controlled, they can interfere with each other and access data illegally. Their rights for resources should be carefully defined according to appropriate policies, for example ‘need-to-know.’

The solution to this problem must resolve the following forces:

- There should be a convenient way to select a policy to define process’ rights. Defining rights without a policy brings contradictory and non-systematic access restrictions that can be easily circumvented.
- A child process may need to impersonate its parent in specific actions, but this should be carefully controlled, otherwise a compromised child could leak information or destroy data.
- The number of child processes created by a process must be restricted, or process spawning could be used to carry out denial-of-service attacks.
- There are situations in which a process needs to act with more than its normal rights, for example to access data in a file to which it doesn’t normally have access.

## 94.5. Solution

Because new processes are created through system calls or messages to the operating system, we have a chance to control the rights given to a new process. Typically, operating systems create a new process as a child process. We let the parent assign a specific set of rights to its children, which is more secure because a more precise control of rights is possible.

## 94.6. Structure

Figure 189 shows the class diagram for this pattern. The Controlled Process Creator is a part of the operating system in charge of creating processes. The Creation Request contains the access rights that the parent defines for the created child. These access rights must be a subset of the parent's access rights.

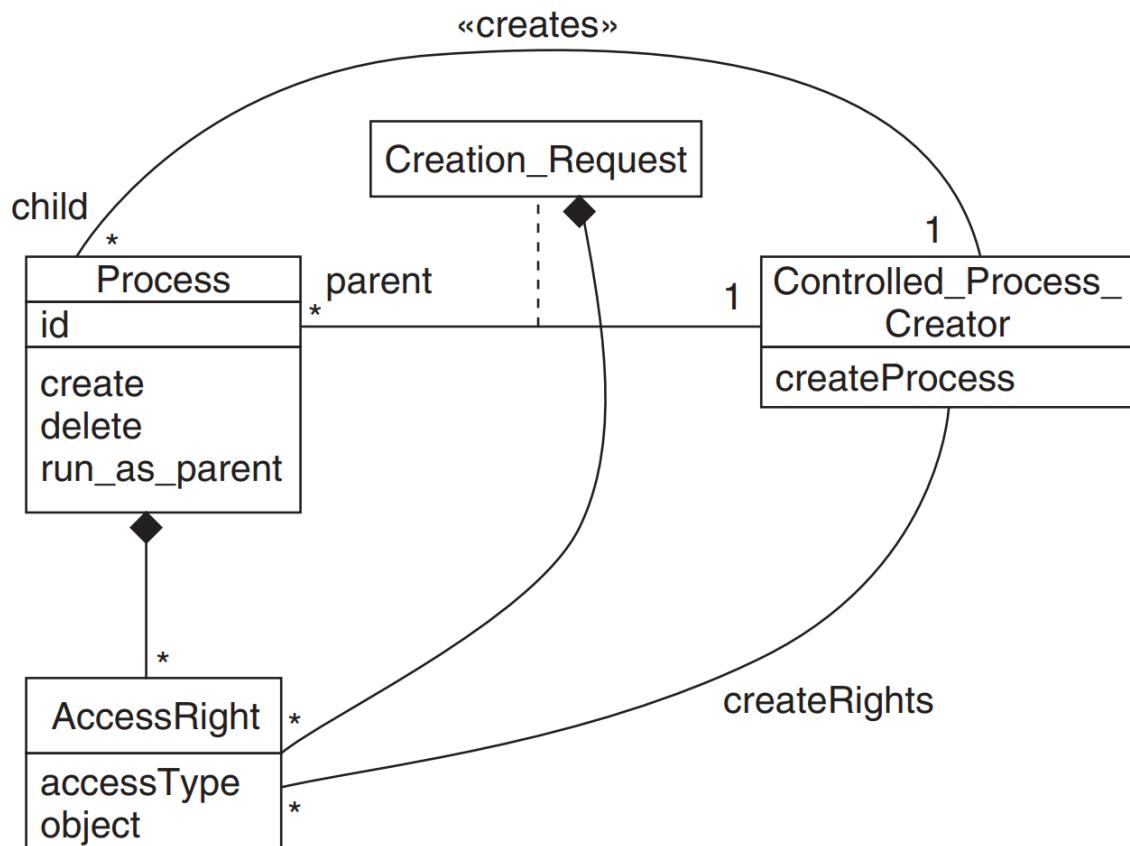
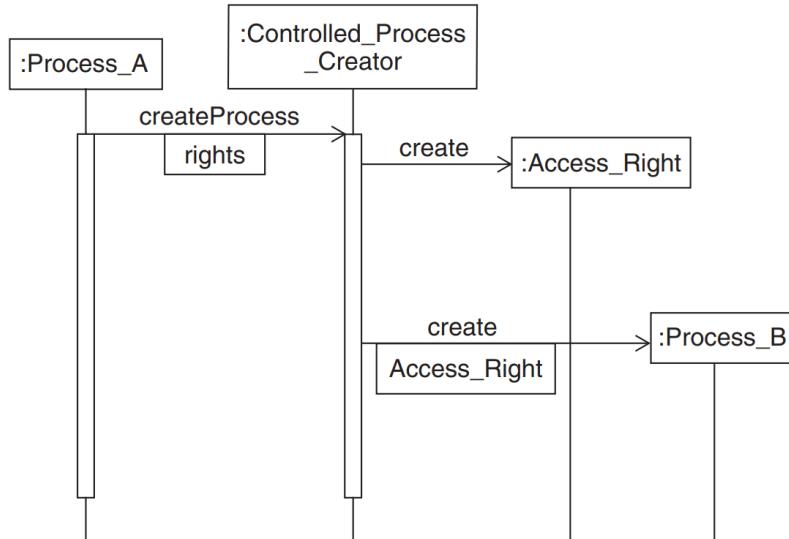


Figure 189: Class diagram for CONTROLLED PROCESS CREATOR

## 94.7. Dynamics

Figure 190 shows the dynamics of process creation. A process requests the creation of a new process. The access rights passed in the creation request is used to create the new access rights for the new process.



*Figure 190: Process creation dynamics*

## 94.8. Implementation

For each required application of kernel threads, define their rights according to their intended function.

## 94.9. Example Resolved

There is now no automatic inheritance of rights in the creation of children processes, so creating a child process confers no advantage for a hacker.

## 94.10. Consequences

The following benefits may be expected from applying this pattern:

- The created process can receive rights according to required security policies.
- The number of children produced by a process can be controlled. This is useful to control denial of service attacks.
- The rights may include the parent's id, allowing the child to run with the rights of its parent.

The following potential liability may arise from applying this pattern:

- Explicit rights transfer takes more time than using a default transfer.

## 94.11. Known Uses

In some hardened operating systems such as Hewlett Packard's Virtual Vault, a new set of rights must be defined for each child [2].

## **94.12. See Also**

CONTROLLED EXECUTION ENVIRONMENT could use this pattern to define the execution domain of new processes.

## **94.13. References**

- [1] Silberschatz, A., Galvin, P. B., & Gagne, G. (2003). *Operating system concepts (Sixth Edition)*. John Wiley & Sons.
- [2] Hewlett Packard Corporation. (n.d.). Virtual Vault.  
<http://www.hp.com/security/products/virtualvault>

## **94.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# 95. Check Point

## 95.1. Intent

Once you have secured a system using SINGLE ACCESS POINT, a means of identification, authentication (I&A) and response to unauthorized break-in attempts is required for securing the system. CHECK POINT makes such an effective I&A and access control mechanism easy to deploy and evolve.

## 95.2. Example

The mayor of our medieval town that established SINGLE ACCESS POINT with their gate and guard is concerned about their protection during times when different threats come close to the town. For example, the merchants would like to have the gate freely open during daytime, to let traders in and out easily. However, they are concerned about burglars sneaking into their warehouses during nighttime.

## 95.3. Context

You have a system protected from unauthorized access in general, for example by applying SINGLE ACCESS POINT. Nevertheless, you want authorized clients to be able to enter your system.

## 95.4. Problem

Whenever you introduce a security measure, you often do not know in advance about all its weaknesses. Also, you only learn how it influences usability if you deploy it to actual users.

A protected system needs to be secure from break-in attempts, and appropriate actions should be taken when such attempts occur. On the other hand, authorized clients should still be able to enter the protected system and should not be impeded too much when they (in the case of a human) make a mistake when providing their credentials.

In addition, you want to consider the change of requirements for identification and authorization (I&A) that might occur over time, either because you need to address new threats, or because you learn from its use. One example for handling that situation is the development of a protected system in which a developer will use a dummy I&A implementation to test the system, without the hassle of logging into the system for every test. Later on, the deployed system needs to be protected by a log-in mechanism that authenticates and authorizes its users.

How can you provide an architecture that allows you to effectively protect system access while still being able to tune I&A to evolving needs without impact to the system you protect?

The solution to this problem must resolve the following forces:

- Having a way to authenticate users and provide validation about what they can do is important.

- Human users make mistakes and should not be punished too harshly for them. However, too many consecutive mistakes at authentication by a user can indicate an attack to the system and should be dealt with.
- Different actions need to be taken depending on the severity of the mistake and current context.
- Spreading checks throughout your protected system increase complexity and make it hard to change. It would be helpful to have a single place to which to refer for authentication and authorization of users.
- You might learn better ways and techniques for I&A after you have deployed an initial system, you might have to change your system after you recognized that it is vulnerable to specific attacks, or you might have to modify the protection because your risks have changed.
- Security-providing code is critical and requires thorough validation through reviews and tests. The smaller such components, the easier are these validations. Reuse of well-proven security components minimizes expensive validations.

## 95.5. Solution

Apply the STRATEGY design pattern [1] to vary the checking behavior at the SINGLE ACCESS POINT. CHECK POINT defines the interface to be supported by concrete implementations to provide the I&A service to the SINGLE ACCESS POINT. A separate configuration (mechanism) defines which concrete implementation of the CHECK POINT interface to use.

The check point interface might provide further security-related functionality in addition to performing I&A. For example, it might define hooks for creating a SECURITY SESSION, checking access rights for a user or session by other system components, logging security-related information, or detecting attack patterns when unauthenticated access attempts occur.

By changing the configuration and thus the concrete CHECK POINT implementation, the behavior at the SINGLE ACCESS POINT changes. For example, Linux provides pluggable authentication modules (PAM) allowing the source of user identities and passwords to be changed [2]. PAM defines a module interface and a configuration mechanism in /etc/pam.d that allows system administrators to adapt the authentication mechanism easily by exchanging the corresponding modules.

The check point effectively encapsulates the security policy to be applied. This allows the development of systems to be independent of a concrete security policy, which might not be available during development. It also allows for easier later adaptation of the security policy of a system whenever external pressure or better knowledge require it.

If not, all security decisions can be made at the time of passing the single access point, CHECK POINT should supply an interface to be used later on by the system's applications. This interface can be used to determine application-specific access rights that might rely on values of application variables not within the scope of the initial access control at the single access point. For example, a bank's application might allow posting of transactions up to \$10,000 for all internal users and up to \$1,000,000 for managers and require a director to acknowledge higher-valued transactions. Hard wiring such decisions within the applications would hinder the evolution of the security policy.

## 95.6. Structure

The following figure shows the elements of CHECK POINT:

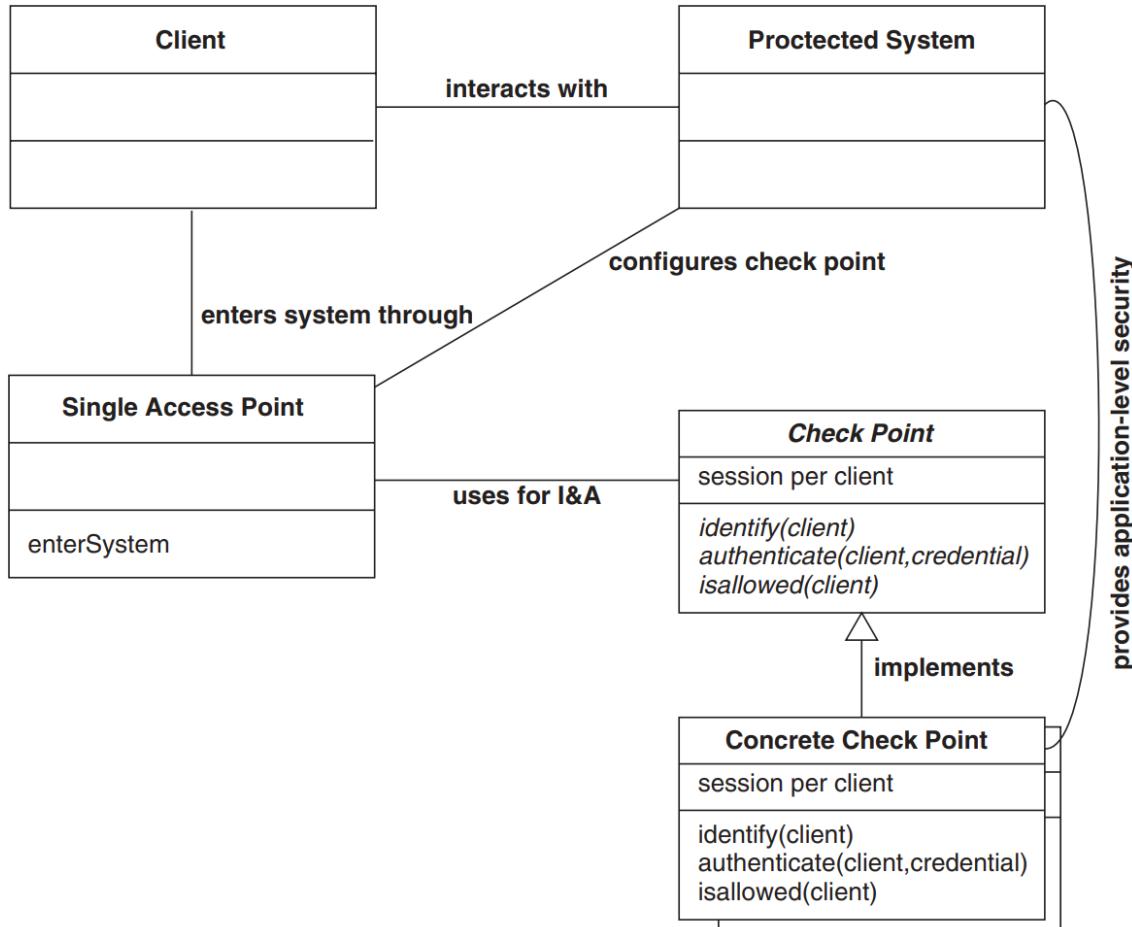


Figure 191: Check Point structure

## 95.7. Dynamics

The scenario shows how a SINGLE ACCESS POINT employs a CHECK POINT implementation to identify, authenticate and authorize a client. The potential creation of a SECURITY SESSION for a client successfully logged in is not shown.

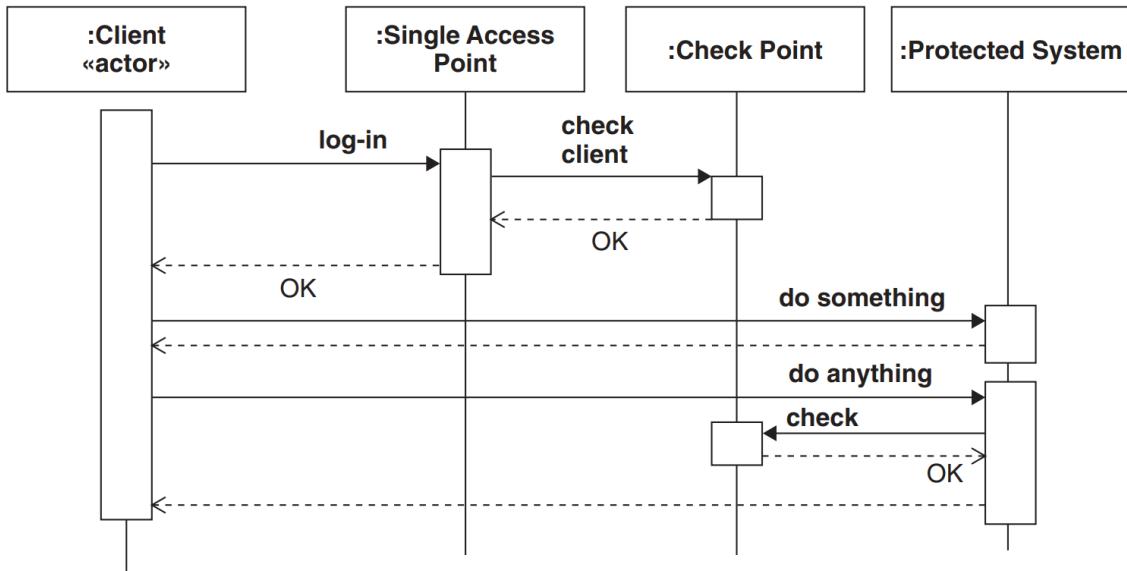


Figure 192: Check Point sequence diagram

## 95.8. Implementation

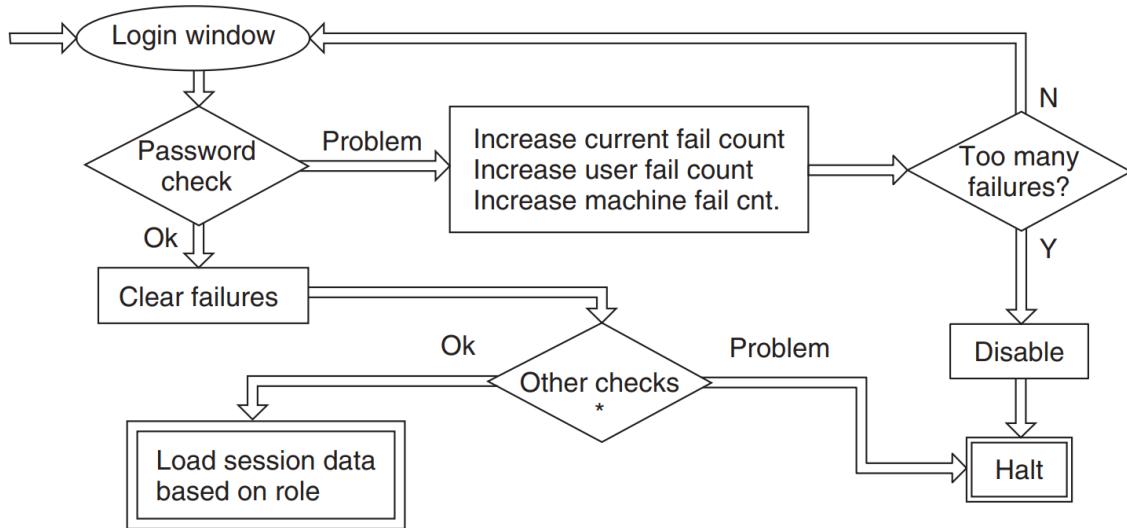
To implement CHECK POINT, several tasks need to be done:

1. Define or re-use an interface to be used by your CHECK POINT components. If implemented in an object-oriented language, this can be an abstract class or an interface. Other languages or implementation techniques might require different means appropriate for the chosen technology. For example, Linux PAM uses an object module with pre-defined function entry points in a table for different operations supported by PAM (authentication, access control, session management, password management). This CHECK POINT interface corresponds to the abstract strategy in STRATEGY [1]. The interface will provide hooks for I&A, authorization, handling unsuccessful attempts.
2. Implement the entry check at the single access point. A single access point ensures that CHECK POINT is initialized and used correctly and cannot be bypassed by intruders. SINGLE ACCESS POINT usually calls CHECK POINT, providing a client's identification and the authentication information they provided. On successful authentication CHECK POINT establishes SECURITY SESSION for the client. If ROLE-BASED ACCESS CONTROL is used, the SECURITY SESSION gets initialized with the client's valid roles.
3. Provide a configuration mechanism to select a concrete CHECK POINT implementation. To make it easy to adjust a system to use a different security policy, and thus a different concrete CHECK POINT implementation, provide a means to configure it. This configuration mechanism must be protected as well, since changing the configuration effectively changes the security policy. Some implementations provide simple configuration files to be maintained by a privileged user. If different concrete CHECK POINT implementations implement different parts of the CHECK POINT interface, it can be handy to be able to combine these concrete CHECK POINT in a CHAIN OF RESPONSIBILITY [1]. For example, if local Unix users should be authenticated by a system as well as users stored in a corporate LDAP directory, one can implement two concrete CHECK POINT, one accessing the /etc/passwd file and one accessing an LDAP

directory. By configuring them to be applied one after the other and allowing access if either succeeds allows local users to log in as well as those in the directory. You can also change the configuration so that both checkpoints must be passed successfully. That way, only users that are registered both locally and in the corporate LDAP directory can pass. Such a change of the policy is possible without changing the code of any of the concrete check points—however, this is at the expense of increased configuration complexity.

4. Implement required concrete CHECK POINT. At least one concrete CHECK POINT implementation is needed. More than one makes it useful for different use scenarios, or their combination by configuration in the system. For example, you can apply the NULL OBJECT pattern [3] to implement a CHECK POINT that is always successful, allowing easier testing during development. A regular concrete CHECK POINT will definitely authenticate clients accessing a system. Usually, it stores the client's identification in a SECURITY SESSION object. If ROLE-BASED ACCESS CONTROL is used, the concrete CHECK POINT initializes the session object with the corresponding role set of the client.
5. Dealing with client errors by the check point. Depending on the security violation or error, different types of failure actions may be taken. Failure actions can be broken down by level of severity. These types of failures and actions are contingent upon the security policy you are implementing. For example, the simplest action is to return a warning or error message to the user. If the error is non-critical, the security algorithm could treat it as a warning and continue. A second level of failure could force the user to start over. The next level of severity could force an abort of the log-in process or quit the program. The highest level of severity could lock out a machine or username. In this case, an administrator might have to reset the username and/or machine access. Unfortunately, this could cause problems when a legitimate user tries to log in later, so if the violation is not extremely critical, the username and machine-disabled flags could be time-stamped and automatically re-enabled after an hour or so. All security failures could also be logged.  
Sometimes, the level of severity of a security violation depends on how many times the violation is repeated. A user who types a password incorrectly once or twice should not be punished too harshly. Three or four consecutive failures could indicate that a hacker is trying to guess a password. To handle this situation, STRATEGY can include counters to keep track of the frequency of security violations and parameterize the algorithm.  
Figure 193 shows an example for such an algorithm [4].
6. Provide application-level API to check point. If some security checks cannot be performed in the check point as the user enters the system, they must be deferred until later. For these cases, the check point could have a secondary interface for application components to use, or a separate authorization component will be required.  
Because a consistent security concept is difficult to achieve in an application, it may be desirable to try to make a reusable security module for use in several applications. That goal is difficult when security requirements vary between the applications. All application teams will need to be involved to ensure the framework will satisfy each application's requirements.  
One approach to reusing security code is to create a library of pluggable security components and a framework for incorporating these components into applications.

However, the algorithm for putting together components will almost always be overridden, making the framework difficult to generalize. Another approach is to use the configuration mechanism explained above to allow small parts of the security algorithm to be combined.



- \* These other checks could include: Is the machine legal? Is the machine disabled? Is user's account disabled? Does user have valid role? Has the user's password expired? These other checks are related to the companies security policy.

*Figure 193: example algorithm*

## 95.9. Example Resolved

The growing medieval town does not yet know the best security policy to use at the gate that is its single access point, but they learn that a single one is insufficient for all their needs at different times.

The mayor decides to provide a more flexible policy at the city gate. During daytime, the gate remains open, while the day-time guard observes the traffic and picks out suspicious-looking people and interrogates them to find out their objectives. During nighttime the gate remains strictly closed and observed by well-armed night guards. Only town dwellers are let in or out during nighttime.

The single gate with a newly built watch house attached to it allows the town leaders to provide a more flexible security policy at their gate, and to change it if harder times require better protection, or prosperous times require easier access.

## 95.10. Consequences

The following benefits may be expected from applying this pattern:

- Concentrate implementation of a security policy. All aspects of a security policy are implemented in a single place and are thus easily accessible for assessment.
- Flexibility in security policy. The common interface to be used for CHECK POINT allows for easy exchange of a concrete implementation if required.

- Easier testing and development. Applying a null CHECK POINT allows more efficient testing and development without the need to provide correct user credentials for every run.
- Independent testing of security policy implementation. CHECK POINT implementations can be tested independently of their surrounding system, allowing testing of this component more thoroughly than would be economic for the integrated system.
- Reuse of security components. Applying CHAIN OF RESPONSIBILITY by configuration of concrete CHECK POINT implementations allows for reuse of these components in different contexts or combinations, effectively providing different security policies with a single code base.

The following potential liabilities may arise from applying this pattern:

- Criticality. Concrete CHECK POINT implementations also localize critical sections. Vulnerabilities contained within concrete CHECK POINT can severely undermine security. Thus, concrete CHECK POINT implementations must be validated thoroughly.
- Algorithm complexity. Dealing with invalid access attempts and detecting malicious users can require complex algorithms. While this complexity is unavoidable, CHECK POINT at least concentrates it in a single defined location.
- State complexity. Some security checks cannot be done at start-up. CHECK POINT must have a secondary interface for parts of applications that require such checks. Usually, the necessary information is already collected at login of a client and stored in its SECURITY SESSION for reuse by these later checks.
- Interface complexity. Designing a good and future-proof check point interface for applications can be challenging. Enforcing its use in the complex application landscape of a corporation can take years.
- Configuration complexity. In addition to the implementations of concrete CHECK POINT and its user applications, the specific configuration also needs to be considered when assessing security. If CHAIN OF RESPONSIBILITY is applied, such as with Linux PAM, understanding the implications of such chaining of concrete CHECK POINT implementations is no longer trivial.

## **95.11. Known Uses**

There are numerous systems and applications that implement CHECK POINT.

PAM [2] implements CHECK POINT. It allows different modules to implement different user authentication strategies. In addition, it allows different applications to be configured using different modules. Once a new technology for user authentication becomes available, for example storing user information in a new kind of database, a new corresponding PAM module allows this technology to be used immediately by all PAM-aware applications.

The Apache Web server implements CHECK POINT with CHAIN OF RESPONSIBILITY within its modular extension mechanism. Extension modules get the chance to validate each HTTP request according to a configured and implicit activation sequence. Modules might reject a request (that is, its URL), modify it, or allow it for further processing.

The log-in process for an FTP server uses CHECK POINT. Depending on the server's configuration files, anonymous logins may or may not be allowed. For anonymous logins, a

valid e-mail is sometimes required. This is similar for Telnet. Linux versions of these applications rely on PAM today.

Xauth uses a cookie to provide a CHECK POINT that X-Windows applications can use for securely communicating between clients and servers.

A Swiss bank uses CHECK POINT based on a CORBA interface throughout all their application systems. In addition to variation of access control by different implementations of the interface, they also allow variation by changing a corporate configuration of user roles, organizational structure, and access rights.

## 95.12. See Also

CHECK POINT uses STRATEGY [1] for gaining flexibility in application security.

CHECK POINT implementations can employ CHAIN OF RESPONSIBILITY [1] to delegate decisions among several concrete CHECK POINT implementations. PAM allows chaining of its modules in this way based on its configuration.

SINGLE ACCESS POINT is used to ensure that CHECK POINT gets initialized correctly and that none of the security checks are skipped.

CHECK POINT usually configures a SECURITY SESSION and stores the necessary security information in it.

ROLE-BASED ACCESS CONTROL is often used to implement CHECK POINT security checks. CHECK POINT sets or evaluates a user's roles stored in its SECURITY SESSION.

For development purposes, or in domains in which security is not a requirement, NULL OBJECT can be used for implementing a concrete CHECK POINT that permits everything.

## 95.13. References

- [1] Gamma, E., Helm, R., Johnson, R., Johnson, R. E., & Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH.
- [2] Morgan, A. (2005, September 05) Linux-PAM. <http://www.kernel.org/pub/linux/libs/pam>
- [3] Woolf, B. (1997). The Null object pattern. In *Pattern languages of program design 3* (pp. 5-18).
- [4] Yoder, J., & Barcalow, J. (1997, September). Architectural patterns for enabling application security. In *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97)* (Vol. 2, p. 30).

## 95.14. Source

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# 96. Demilitarized Zone

## 96.1. Intent

Any organization conducting e-commerce or publishing information over Web technologies must make their service easily accessible to their users. However, any form of Web site or e-commerce system is a potential target for attack, especially those on the Internet. A Demilitarized Zone (DMZ) separates the business functionality and information from the Web servers that deliver it and places the Web servers in a secure area. This reduces the ‘surface area’ of the system that is open to attack.

## 96.2. Example

A commercial Internet system holds customer profiling information, dealer order information and commercially sensitive sales information, any of which could be stolen or corrupted by an attacker. This information must be shared with the organization’s corporate systems, making them liable to attack as well.

You could use a firewall to control access to your systems from the outside world as shown below.

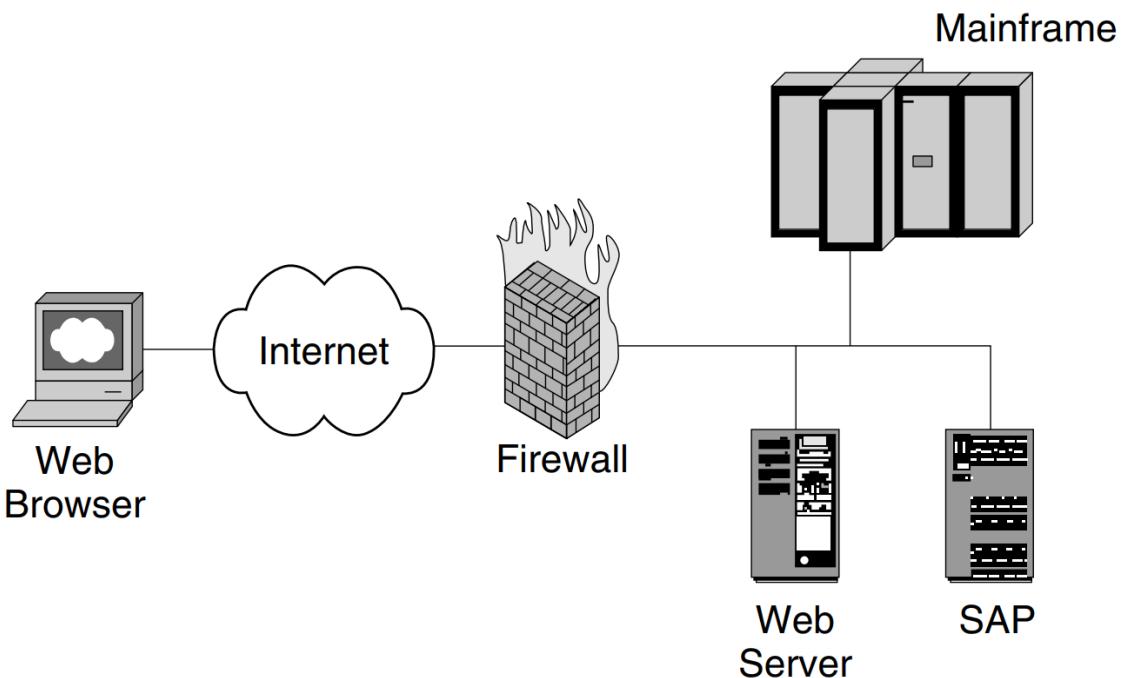


Figure 194: Firewall protection against outside attacks

The firewall would be configured to allow only inbound traffic to access the Web server. However, this places a large onus on the system administrators to configure the firewall correctly, and on the firewall software to operate correctly. If the firewall fails, an attacker could potentially have direct access to other business resources such as the SAP system or mainframe shown in the diagram. The configuration of the firewall is further complicated by the fact that for any highly available Web based system, multiple servers must be exposed to support either load balancing or failover. If the Web-based system is also high-functionality,

additional protocols must be allowed through the firewall. All of this makes a configuration error more likely.

### 96.3. Context

An APPLICATION SERVER ARCHITECTURE [1] has been adopted to deliver an Internet technology application. The business logic and dynamic Web content generation of the application resides on application servers, while all static content is provided by Web servers that also act as a PROTECTION REVERSE PROXY for the dynamic Web content. The application holds information on users and provides important functionality for users, but the application is exposed to an environment that contains potential attackers.

### 96.4. Problem

Internet technology systems, particularly those facing the public Internet, are regularly subject to attacks against their functionality, resources, and information. How do we protect our systems from direct attacks?

Solving this problem requires you to resolve the following forces:

- The cost of an extensive security solution will be high, but the cost of an intrusion may also be high in terms of system damage, theft, and loss of customer confidence. If the potential rewards from the attack are high in terms of financial gain or publicity, the risk of such an attack will be higher. The scope, and hence cost, of any countermeasure must be commensurate with the level of perceived threat and the potential cost of the intrusion.
- To prevent attack, we must make intrusion into any part of the system as difficult as possible, especially an organization's internal business systems. However, increasing the level of security will generally make the system more difficult to use, which conflicts with the goal of making the system open and easy for legitimate users.

### 96.5. Solution

Provide a region of the system that is separated from both the external users and the internal data and functionality—commonly known as a demilitarized zone (DMZ). This region will contain the servers, such as Web servers, that expose the functionality of the Web-based application. Restrict access to this region from the outside by limiting network traffic flow to certain physical servers. Use the same techniques to restrict access from servers in the DMZ to the internal systems.

### 96.6. Structure

A DMZ requires the following elements as shown in figure 195:

- External router, a filtering router whose principal responsibility is to ensure that all inbound traffic is directed to the firewall. Its secondary responsibility may be to keep out random traffic generated by attackers.

- Firewall, responsible for receiving inbound requests from the external router and subjecting them to more sophisticated analysis, such as stateful inspection. If a request is judged to be legitimate, it will be forwarded to an appropriate Web server.
- Web servers, providing access to the application's functionality and information. There may be multiple Web servers that are accessed through a load balancer. A Web server will receive a request from the firewall and service that request. A request for a static resource, such as a fixed page of HTML or an image, may be delivered from a cache held on a local disk. A request for a dynamic resource will be proxied through to an application server that is shielded from the outside world in the style of a PROTECTION REVERSE PROXY. No application functionality, such as servlets or ASP.NET pages, will run on the Web servers, as this makes them open to direct attack. Although described here as 'Web' servers, these servers may support access through other protocols such as FTP.
- Internal router, a filtering router whose principal responsibility is to ensure that it only passes legitimate traffic from the Web servers through to the internal network.
- Application servers, a platform on which the application's code runs, typically in the form of Web components such as servlets and business components such as EJBs.

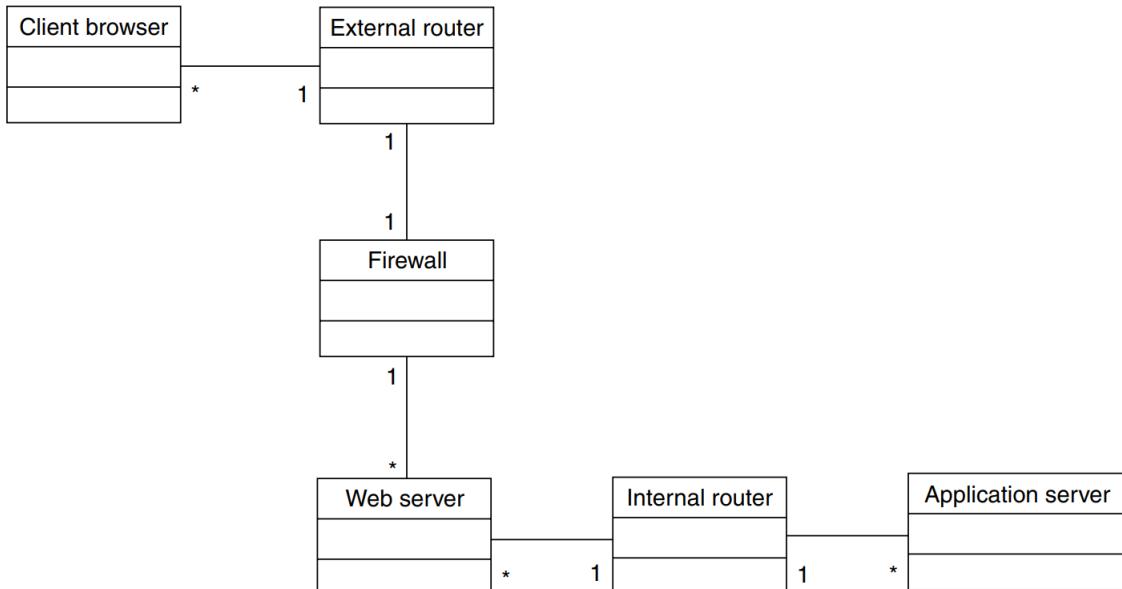


Figure 195: DEMILITARIZED ZONE (DMZ) structure

## 96.7. Dynamics

The first scenario shows a successful client request for some business functionality. The client browser request is filtered by the external router to ensure that it is destined for a valid server. The request is forwarded to the firewall to undergo more rigorous checking. If the firewall is happy with the protocol use, the request goes onwards to the server requested by the client.

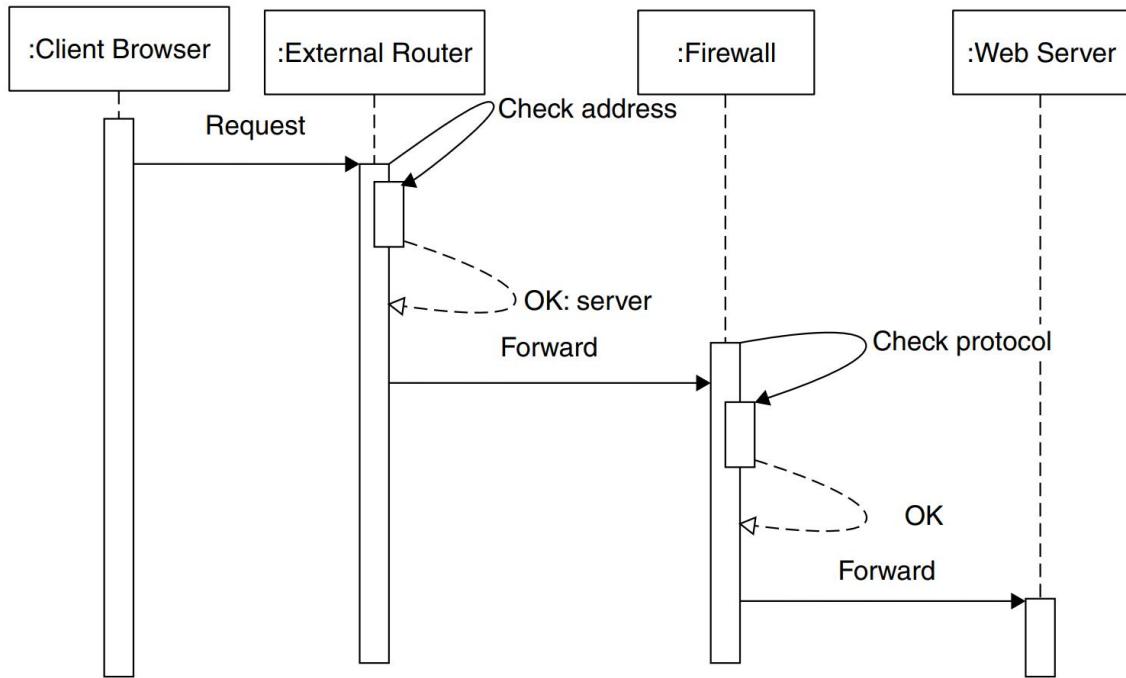


Figure 196: Filtering a client request in a DMZ

The second scenario shows a malicious client call being blocked by the firewall. The client browser request is again filtered by the external router to ensure that it is destined for a valid server. The request is then forwarded to the firewall to undergo more rigorous checking. At this stage, the firewall detects invalid protocol use—maybe some form of protocol-based attack, or an attempt to flood the server. The request is rejected, and the suspicious activity is logged. See figure 197.

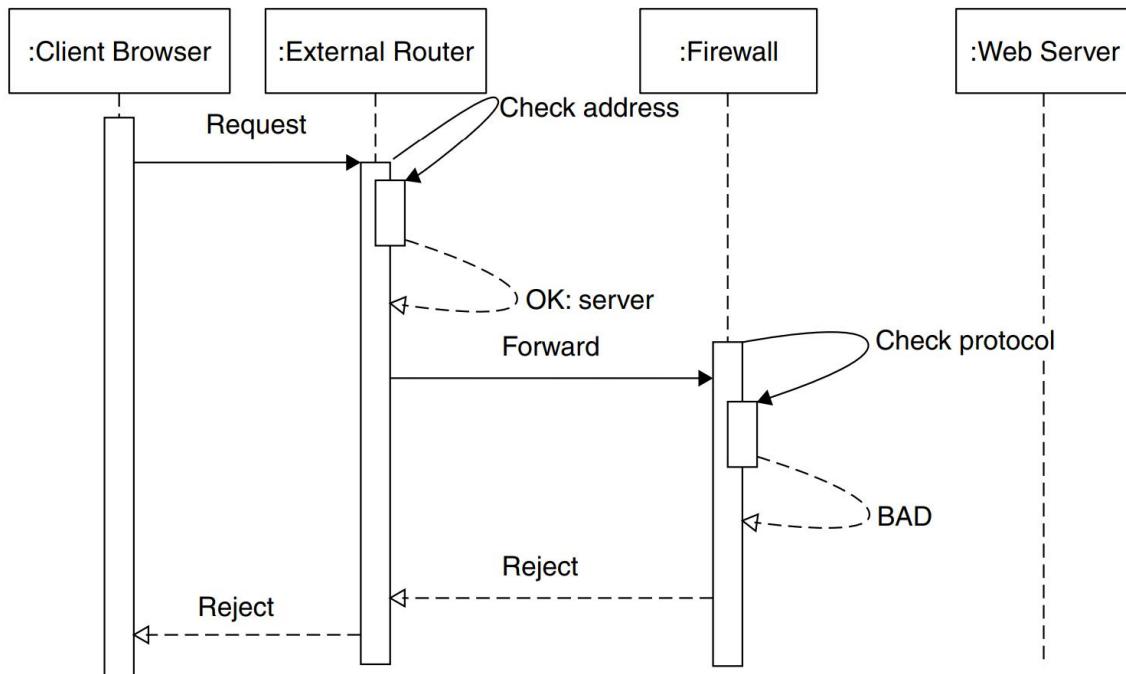


Figure 197: Rejecting a client request in a DMZ

## **96.8. Implementation**

Since the request handling and business functionality must be separated by a filter, it is best to use DEDICATED WEB and APPLICATION SERVERS [1] where any programmatic functionality, whether business or presentation, is deployed on an application server that is physically separate from the Web server. These application servers can be placed on a more protected network than the Web servers. This protected network will have easier (possibly direct) access to the corporate information and services required by the Web-based application.

The external router should be configured to deny any attempted access to any network addresses outside of those known in the DMZ. To increase security, any requests with a destination address that does not match the Web server address (or that of the Web server cluster) may be rejected. The external router may also reject requests based on the port number of the request, for example rejecting any request that is not for port 80. The external router will therefore block direct attacks on the internal router, and possibly the firewall.

The Web servers will be built solely for the purpose of delivering static Web content or proxying requests through to the application servers. These Web servers should be locked down (or ‘hardened’) by removing unnecessary functionality. Such hardening helps to prevent other, unintended, access to the servers.

The internal router will limit network traffic to connections between the Web servers on the DMZ and specific internal servers, such as the application servers, using a fixed set of protocols. This restriction reduces the risk of attack on other internal systems. The use of an internal router helps to reduce the risk of attack should the external router be breached. Because of this threat, no traffic should be allowed directly from the external router to the internal router.

The whole operation of the routers and the traffic filtering may be controlled from a machine running specific firewall software. This makes it easier to apply consistent rules to the routers and to use statistical analysis to detect potential attacks. The firewall applies more sophisticated traffic filtering rules to detect more complex attacks. Depending on the type of firewall, the network traffic may or may not pass through the firewall itself.

Because the number of servers exposed to the outside world is reduced, it means that fewer parts of the system need a high level of security. In the scenario described, the application servers will not need to be hardened to the same level as the Web servers. To access those servers not directly exposed (and hence less securely configured), any attacker will have to breach several security elements that form part of the DMZ. Hopefully, they will set off various intruder alerts as they do so—if, indeed, they are capable of doing so.

Applying a DMZ to a system is a good way to provide protection for the system. However, you must remember that protecting the platforms on which the system is built is only part of the solution. Since security is a matter of policy as well as technology, all protection mechanisms—such as a DMZ must be backed up with appropriate procedures and processes to ensure that the level of security remains high. If there is a high level of concern about possible attacks on the system, an intrusion detection system (IDS) (see INTRUSION DETECTION REQUIREMENTS) may also be used. An IDS monitors the traffic on the network, or on specific hosts, looking for suspicious activity. If the IDS identifies a pattern of network or host traffic that indicates an attack is underway, it will notify the system administrators. An IDS could be used on the DMZ itself, on the internal network, or both.

## **96.9. Example Resolved**

The commercial organization implements a typical DMZ configuration. The system only allows HTTP and FTP traffic into the organization, and even then, such traffic is only allowed to the Web servers. The external router drops any traffic that tries to reach the internal router, firewall, or the external router itself. This rogue traffic is also logged at the firewall and notified to the system administrators to assist in the detection of potential intruders.

The internal router allows inbound traffic only from the Web servers, and even then, it limits it to specific protocols (IIOP), specific hosts and specific port ranges. This means that any hacker who achieves a beachhead within the DMZ must either attack the internal router directly (and risk setting off alarms from the router) or they must be literate in IIOP to the degree that they could use it to gain access to one of the servers on the other side of the internal router.

The firewall acts as a clearing house for security alerts and as a management console for the DMZ. The organization chose Firewall-1 software based on its track record and traditional association with Sun, on whose hardware it is deployed. The Firewall software gets alerts from the two routers and provides a unified view of security on the DMZ. The firewall software also controls the configuration of the two routers, to avoid inconsistencies creeping in between the three main parts of the firewall system.

## **96.10. Consequences**

The following benefits may be expected from applying this pattern:

- Security is improved, because fewer systems are exposed to attack and multiple firewall artefacts must be breached to compromise security.
- The level and depth of protection can be varied to match the anticipated risk and the cost limitations.
- The additional security is transparent to the users of the system functionality and to the developers of such functionality.
- Fewer hosts must be hardened to withstand attack than if they were all exposed to the outside world.

The following potential liabilities may arise from applying this pattern:

- Availability may be impacted because the firewall becomes a single point of failure. The standard procedure is therefore for a firewall to ‘fail closed’—that is, in the event of failure, it will deny all connections to the protected systems.
- Manageability is impacted, because the very restrictions that limit access to internal data may make it difficult to access the application from an internal monitor.
- Cost is increased because extra elements must be procured to build the DMZ. These include not only the filtering routers, firewall software and firewall host, but also the extra network equipment, such as switches and cabling, used on the DMZ itself.
- Performance is impacted due to the overhead of network traffic filtering. Performance is also impacted as it becomes necessary physically to separate the Web servers from the application servers. If this has not already been done to improve another non-functional characteristic, it must be done to implement a DMZ, and so will add multiple extra network hops for each user transaction.

## **96.11. Variants**

Multi-homed firewall. The number of machines involved in implementing the DMZ will vary according to the level of protection required (based on anticipated risk) and the amount of money available. In the simplest case, the DMZ may be partitioned using a single firewall machine. This machine will have three network cards: one connected to the Internet, one connected to the internal network, and one connected to a dedicated LAN containing only the Web servers and any other ‘public facing’ parts of the system. The firewall software running on the machine will manage the traffic between the three networks to maintain three separate security zones. The benefits of such an ‘multi-homed host’ implementation include reduced cost and ease of maintenance. However, this system creates a single point of failure, both in terms of security and availability. It also means that any attacker is only one system away from gaining access to the sensitive internal systems.

Firewall as filter. A multi-homed firewall host may be used in place of the external or internal router. This means that all traffic must pass through the firewall (and its filtering rules) to reach the internal network or the DMZ itself.

Stealth firewall. Rather than relaying traffic, the firewall may simply be attached to the demilitarized network and act in ‘stealth’ mode, simply monitoring traffic for potential intrusion. This can make the firewall itself more difficult for an intruder to detect.

## **96.12. Known Uses**

DMZs are extremely common for almost all Internet sites and advice on the creation of DMZ configurations is offered by almost all major network hardware and software vendors, such as:

- Sun [2]
- Microsoft [3]
- Cisco (variously described as part of their SAFE Blueprint)

## **96.13. See Also**

## **96.14. References**

[1] Dyson, P., & Longshaw, A. (2004). *Architecting enterprise solutions: patterns for high-capability internet-based systems*. John Wiley & Sons.

[2] Sun. (n.d.). <http://www.sun.com/executives/force/solutions/SecuritySolnIIFinal3.pdf>

[3] Microsoft. (n.d.). [http://www.microsoft.com/windows2000/techinfo/reskit/en-us/default.asp?url=/windows2000/techinfo/reskit/en-us/deploy/dgcf\\_inc\\_icku.asp](http://www.microsoft.com/windows2000/techinfo/reskit/en-us/default.asp?url=/windows2000/techinfo/reskit/en-us/deploy/dgcf_inc_icku.asp)

## **96.15. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **97. Enterprise Partner Communication**

## **97.1. Intent**

Enterprises often partner with third parties to support their business model. These third parties may include application and managed service providers, consulting firms, vendors, outsourcing development teams, and satellite offices. As part of this relationship, access must be granted to allow data to travel between the organizations. Without attention to the protection of that data and the methods by which they are transferred, one or both organizations may be at risk.

## **97.2. Example**

The museum has received a sum of money and is expanding! It wants to expand its services in the following ways:

1. Publish an RSS news feed advertising all upcoming museum events and information.
2. Sell goods online from its Web site. The museum has created a merchant account with a popular payment processor and financial organization. The Web site application will use a programmatic API provided by the payment processor.
3. Outsource the development of a Web site to a third party. One component of the Web site will be a public, e-commerce site selling goods and promoting museum events and exhibits. The second component will be a private, intranet Web site containing an employee directory as well as confidential corporate funding and research and development data. The museum realizes that the third party will require some confidential database tables and documents in order to design and test the application.
4. Subscribe to the International Museum Consortium (IMC) service. This service will publish current and rolling exhibit information to other subscribers. Membership of this service will allow the museum to search and bid for rolling exhibits from any other subscribing museum around the world. They feel it would give them a competitive advantage over other regional and local museums and will substantially increase their patron attendance. The IMC will provide the software application, centrally manage user accounts, and facilitate a bidding and messaging process. The museum already has an infrastructure capable of operating and managing the software application, and simply needs to configure it to access the museum's inventory database.

Each of these projects involves exchanging information with other parties but vary in the degree of security requirements and in the method of data exchange. The museum clearly recognizes the value of these projects, but is concerned that its personnel, customer, and confidential exhibit information will be at risk of unauthorized access, modification, or denial of service. It would like to implement these projects but needs to protect its data, systems, and reputation.

### **97.3. Context**

An enterprise has an existing business process, or is proposing a new business process, that requires information to be exchanged with another entity across a computer network. The business factors that initiated the partnership have already been determined and a high-level service level agreement, complete with disaster recovery and business continuity planning, has been established.

### **97.4. Problem**

When an enterprise engages in a business relationship, it typically exchanges information and allows users and/or applications to access privileged resources. Not only can there be risk of theft or manipulation of data, but also risk of unauthorized access to resources by another organization. Furthermore, you may trust the partner with whom you entered into a relationship, but can you trust their contractors, application vendors, networks, or firewall configuration? A breach in their network may lead to a breach in your own.

How can an enterprise protect its systems and data while communicating with external partners?

An enterprise must resolve the following forces:

- It needs to be reasonably assured that sensitive information is protected when traveling beyond its control.
- Security procedures become difficult to manage when one entity does not share the same security requirement and considerations as the other.
- It must conform to legislation when storing and transferring financial or personal health information.
- Applications that communicate with business partners become vulnerable, not only to attack from that partner, but also from attacks from users who defeat the partner's security.
- The services that the partner may access might require special or custom network paths that are not used by regular customers or internal users.
- An enterprise may not have the time or ability to properly evaluate the security controls of the partner, and the partner may not be able to conform to the security requirements imposed by the enterprise (in time).
- Outsourcing software development efforts creates additional challenges, as the data and people may reside across the planet and beyond the immediate reach of the enterprise.
- Both parties must commit to the agreement but be flexible enough to modify the policy should the risk or business requirements change. For example, if transaction volumes dramatically increase, or if vulnerabilities are suddenly discovered in an application.
- The enterprise may require the business partners to conform to a particular interoperability scheme that the partner is not able to match.

## **97.5. Solution**

Specify enterprise partner communication in five areas: define the scope and security requirements of the information to be exchanged, audit the business partner, identify, and protect communication channels, define exchange methods and procedures, and identify service termination activities.

1. Define scope and security requirements. First, define which data or application services are to be exchanged between organizations. Then identify the security requirements for this information.
2. Audit business partner. Perform a security audit of the partner organization commensurate with the security requirements of the information and the policies of your enterprise.
3. Identify and protect communication channels. Identify and protect communication channels in the following ways:
  - Communication channels: identify the preferred channels of communication.
  - Traffic separation: separate business partner traffic from regular enterprise traffic, and from other partners, wherever possible.
  - Ports and portals: determine the required SINGLE ACCESS POINT connecting the two organizations and secure them.
  - Access controls: apply administrative, physical, and technical access controls, as appropriate, to protect the data throughout its life cycle. These controls serve to protect the data while stored either at the enterprise or business partner and as it passes from one system to another.
4. Define exchange methods and procedures. First, identify the pre- and post-processing procedures that are to be applied to the data and communication channels. Then maintain and monitor usage logs and reports. This will provide an early warning of performance and stability issues, as well as indication of malicious activity.
5. Perform service termination activities. At the completion of the partner agreement, perform the following service termination activities:
  - Access revocation: user accounts, authorization privileges and system access should be promptly removed.
  - Data sanitization: purge all sensitive information from disk drives, databases, and other files
  - Repurpose assets: network devices, servers, application resources can now be re-used for other partner communications or internal functions

## **97.6. Structure**

The structural components of this pattern are displayed in figure 198.

SINGLE ACCESS POINT provides a central, auditable entry point into the enterprise. Access controls at these access points enforce restrictions on inbound and outbound traffic. Dedicated communication channels and encryption controls protect the data throughout its transmission and storage. Partitioned storage facilities provide dedicated separation of information between enterprise and partner data.

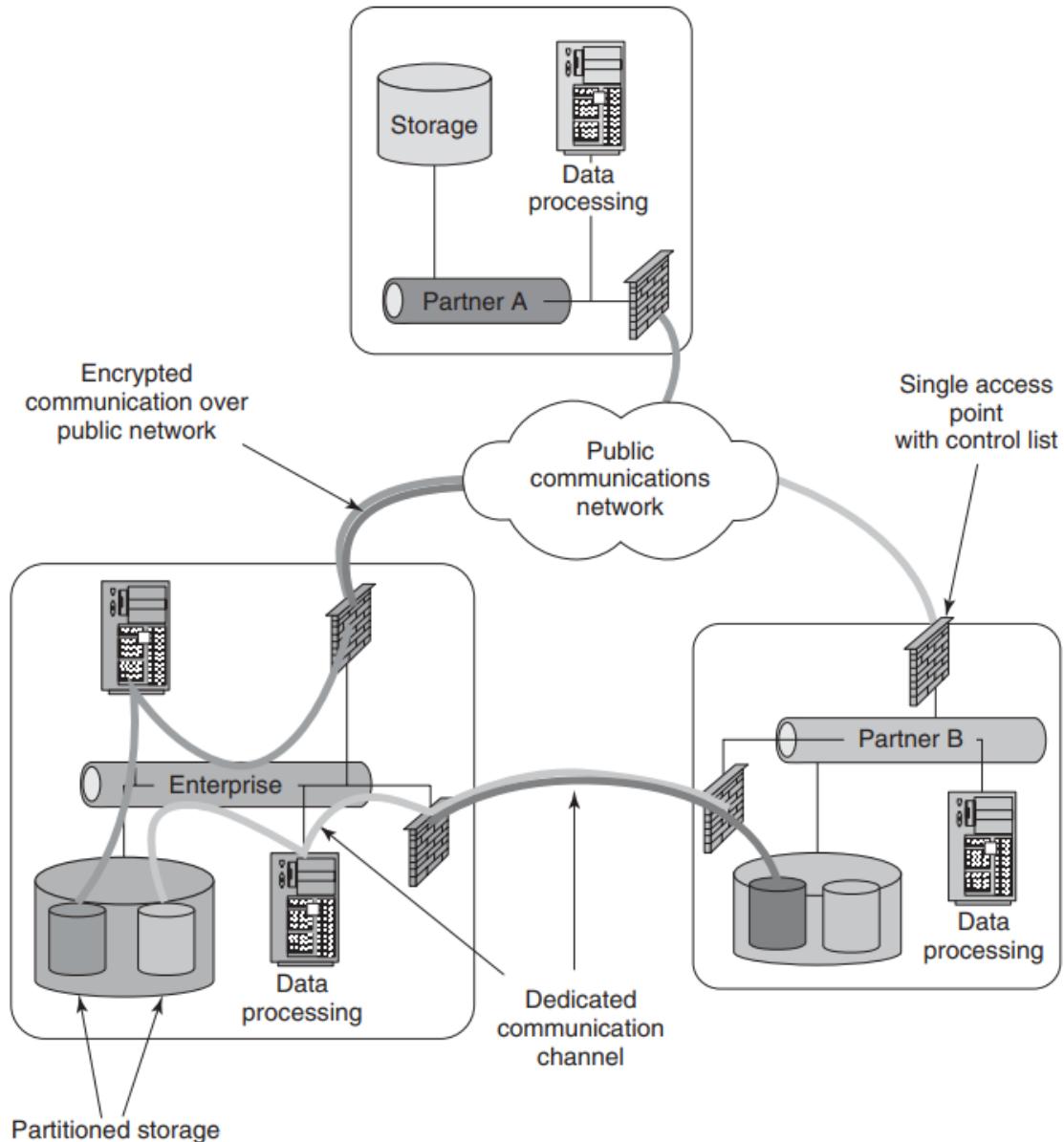


Figure 198: Enterprise partner communication structure

## 97.7. Dynamics

## 97.8. Implementation

The following steps should be considered during the pattern implementation:

1. Define scope and security requirements. Determine the minimum set of data that should be exchanged by sanitizing it as much as possible. That is, strip it of any confidential or unnecessary information. Personally identifiable financial or medical numbers, for example, can be substituted for another unique identification number. There is no need to send more information than necessary. Indeed, extra data may inflate security requirements, incurring additional infrastructure, costs, and delays.

If application services are used, provide interfaces (APIs, URLs) for only those functions that are necessary to fulfill the business requirement. This improves security by limiting the access to the enterprise and its systems.

The data owner will be able to provide the security requirements of the data. If not, SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS can be used. This enterprise pattern provides a process whereby a data owner (or other) can determine the security properties necessary for the exchange of data between organizations. The properties are expressed as follows:

- Protection against inadvertent or unauthorized disclosure: confidentiality
- Protection against inadvertent or unauthorized modification: integrity
- Making business assets available for authorized use: availability
- Attribution of responsibility for actions: accountability

2. Audit business partner. An enterprise may require a security audit of the partner before exchanging any information or entering into any business agreement. The purposes of the audit are twofold:
  - To evaluate the security policies, practices, and controls of the partner. Policies and practices can be evaluated by the enterprise, or it may prefer to acquire independent results from an external consulting firm. Security controls can be tested with a vulnerability scan and/or penetration test.
    - To compare and reconcile these results against ‘prescribed standards of performance’ [1]. Such standards may be enforced by a number of sources: federal or local legislation may define a basic (minimal) level of protection for information exchange, use and storage. The enterprise, itself, may impose much stricter restrictions.
3. Identify and protect communication channels. Communication channels include all protocols, hardware devices, communication lines (dedicated and public) and computer network segments over which data will be traveling should be identified. Both the type of data as well as its security requirements will determine (or, at least, strongly effect) the type of communication channel necessary or available. For example, many payment transactions are still sent over value added networks (VANs) using X.400 messaging. Conversely, many modern applications communicate across a public TCP/IP network using a combination of HTTP, HTTPS protocols and HTML, XML, or a proprietary message format. Other influencing factors of the type of communication channel include the available technology of the partner organization, industry conventions and budget.  
For traffic separation, dedicated communication channels are preferred, because they reduce the risk of harmful (malicious or inadvertent) events originating from one partner and affecting others, as well as eliminating single points of failure across multiple users. When possible, isolate partner traffic either physically or logically. Physical separation is achieved by using dedicated hardware (servers, firewalls, communication lines) and software. Logical separation is achieved through segmented IP addressing, virtual environments such as multiple operating systems on a mainframe, virtual Web hosts, and multiple databases on a shared installation.  
For each of these communication channels, identify and protect the SINGLE ACCESS POINT into the enterprise and its systems. Relevant questions to ask are: do additional ports need to be opened up at the firewall or edge routing device, and how will this affect the overall security posture of the environment? Does the network or application need to be modified to allow access for a particular set of users or hosts

and will this result in additional threats or vulnerabilities? Will physical access be required by the partner to the enterprise office (or its affiliates)? Will any special arrangements be necessary to support the partner, such as segregated network connectivity (VPN), analogue phone lines, and so on? How will enterprise systems be protected while external partners are on the premises?

Access controls must be applied at any point at which data is passing from one system (user, hardware device, application) to another. Types of access controls include the following:

- Technical access controls used at the network (transmission) layer in routers, firewalls, and servers to prevent access from unauthorized hosts. These can be used at the application layer as well, to grant and deny access to specific users.
- Administrative access controls used to define policies and procedures that govern the acceptable circumstances and conditions under which the data can be accessed.
- Physical access controls used to ensure physical protection of the data. For example, mechanical door locks to offices, server rooms, and cabinets.

For all implementations of access control, employ a ‘failed closed’ policy by preventing all access, then granting access for specific entities. This is most commonly used on network perimeter devices such as firewalls, routers, and internet servers.

User authorization should be enabled according to the principles of Least Privileges and Separation of Duties [2] and by applying ROLE-BASED ACCESS CONTROL. Least Privileges ensures that a particular subject (human or application user) only has the necessary privileges required to perform a given task. An example would be to grant access to write or modify files in a repository, but not delete files. ROLE BASED ACCESS CONTROL is used to assign a subject (user) to one or many roles, with each role allotted a unique collection of privileges. This abstracts the subject from their privileges, providing a business-centric approach and improving the management of access control. The roles of system administrator, sales advisor, and purchaser, for example, would all have unique collections of privileges in a corporate directory.

Finally, Separation of Duties distributes responsibility (trust) across multiple subjects and is often used in conjunction with ROLEBASED ACCESS CONTROL. For example, partitioning roles (and therefore privileges) of a bank customer with teller and auditor.

4. Define exchange methods and procedures. Any form of data exchange will require special pre- or post-processing, depending both on the method of exchange and on any requirements to comply with internal policies or legal regulations. Examples of four common exchange methods and procedures are listed below.

### 97.8.1. Method 1: On-demand Transfer

This refers to the ad-hoc exchange of information of raw data from one site to another. For example, weather data, news feeds, stock ticker data, and batch file transfer.

Security related procedures:

1. By what mechanism will data be transferred? FTP, HTTP, private line?
2. How will the data be transferred: pushed or pulled? Automated, batch, manual?
3. What will be the naming convention of the files? For example, dated, static, other?
4. Must a file always exist? Will null files be accepted?

### **97.8.2. Method 2: Real-time Information Exchange**

For example, payment processing, EDI. This is payment transaction information, either between financial organizations, or a merchant and a payment processor. The information is sent real-time and contains account (credit card) data and a monetary value. These can be both high or low volume and high or low value transactions. Generally, when a transaction is sent, an authorization or confirmation number is returned for logging.

Security-related procedures:

1. Will a custom programming API be necessary? If supplied by another entity, will it require review, modification to operate within the enterprise's infrastructure?
2. What are the possible response codes? What do they represent?
3. Can batching (near real-time) be used, or are all transactions individual?
4. Is a notification to other business processes necessary to continue a workflow?

### **97.8.3. Method 3: Large Volume Information Transfer**

For example, managed security outsourcing, and application development outsourcing relationships. This is the large volume shuffling of corporate data. Managed service providers are sent large log files for processing. Development outsourcing companies are provided corporate, and sometimes confidential customer, information used for developing or repairing an application on behalf of the enterprise. Information transfer can occur at any time although not real-time. It requires large bandwidth transmission as well as privacy and integrity controls.

Security-related procedures:

1. Is electronic transfer of the data prohibited? Instead, should the data be transferred physically?
2. What scheduling requirements are required: can the exchange wait until a predetermined time, or does it need to be transferred and processed immediately?
3. What sort of notification, if any, must be made to either humans or applications before or after the data transfer?
4. Will the outsourcing company require direct access to the enterprise's internal networks or servers? Will they require extra privileges to data or user stores?

### **97.8.4. Method 4: Interactive Application Services**

These applications can be accessed by a human or another application in order for one enterprise to provide services to another enterprise. That is, one enterprise is extending its business model to incorporate the services of another enterprise. For example, a Web-based airline reservation site that incorporates the services of a car rental and hotel reservation company, or a third party with network access into the enterprise.

Security-related procedures:

1. What communication and messaging protocols are used?
2. What authentication procedures will exist between applications or networks extensions and what auditing will be performed?

3. What sort of notification, if any, must be made to either humans or applications before or after the data transfer?
4. Who will access the services? Just employees of business partners, contractors or third or fourth parties?

### **97.8.5. Logging and monitoring**

Log files from application servers, firewalls and other application and networking devices will provide important use and access audit trails. They will identify where the access originated, potentially what was being accessed, and for how long. They can be used to monitor use for performance and quality assurance reasons, as well as to provide information for forensic investigations. Such processes may already be employed at the enterprise, but they should also be enabled by the business partner.

Implementation steps (continued):

5. Perform service termination activities.

### **97.8.6. Access revocation**

Temporary and expired user accounts are often a convenient way of gaining access to a system. Therefore, remove all user accounts and entries from access control lists from any network device (firewalls, routers, VPN concentrators), server (hosts files), applications and user stores (database, LDAP, and so on).

### **97.8.7. Data sanitization**

Proper cleaning of disk drives is a critical but often-overlooked task. Specialty tools are available to completely erase entire hard drives or particular files. Also, one or both parties possess data belonging to the other: this data should be completely erased or returned to the business partner in a secure manner.

### **97.8.8. Asset Repurposing**

Only when access controls have been reset and data sanitization has occurred is the asset ready to be reused for another business partner or internal function.

## **97.9. Example Resolved**

The museum has chosen to address its four projects as follows:

### **97.9.1. On-demand News Feed**

The museum will make its news data available to other museums or organizations through a free RSS subscription service: the museum just wants to know who's accessing their feeds. The information will be updated as necessary and can be retrieved ad hoc by the other entities. Using SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS, the museum recognizes that prior to release, the information will require medium confidentiality and integrity, while after

release only a low degree of integrity is required. The application server will sit behind a firewall in a DEMILITARIZED ZONE and both devices will undergo moderate host hardening and patching. The read-only information will be accessible via FTP or XML over HTTP, and all connections will be logged. To prevent abuse of the feed, request attempts will be limited to ten per hour.

### **97.9.2. Real-Time Transaction Processing**

Payment transactions are initiated from the museum's application and sent real-time from the museum to the payment processor. Each transaction travels across the public internet over TCP/IP and is encrypted end-to-end using the API's encryption algorithm. This ensures confidentiality and integrity of the transaction. The authorization number is stored by the museum.

The payment processor provides this service to hundreds of other merchants and is quite proficient at high availability of all of its systems. The museum is not dual homed and therefore accepts the risks due to an outage of its Internet service provider. In the event of a failure, however, the museum has obtained contact information for both the payment processor and its ISP.

### **97.9.3. Outsourcing Web Site Development**

Clearly, the transmission of confidential information to the development company, as well as their use of that data, requires protection, specifically, confidentiality and integrity. The museum has performed a security audit of the development company. From this audit, it agrees that the application will be developed and tested on servers protected behind both a corporate perimeter firewall and separate firewall, isolating their development environment from other projects. The development party is the only group with system access to the museum database: data will be incrementally backed up on a nightly basis, fully on a weekly basis, and stored in a locked cabinet to which only the immediate team members and managers have access. Finally, the development company will not share any of the enterprise's information with a third party without prior consent of the museum.

### **97.9.4. IMC service**

The museum recognizes that the exposure of its entire database containing all corporate and personnel information would pose too great a risk. Therefore, the museum chooses to install the IMC software and a new database on a separate network segment from the internal corporate systems in a DEMILITARIZED ZONE, isolating it from the corporate network. On a daily basis, only necessary portions of museum and exhibit information will be exported from the authoritative corporate source and imported into the local IMC database. Specifically, donators and funding sources are not exported, nor is financial or human resource information. The museum is provided with an administrative interface that can be used to retrieve messages or update museum information as necessary. The IMC server will marshal all messages between museums over HTTPS only and all user-level requests will be logged.

## **97.10. Consequences**

Use of this pattern provides the following benefits:

- Expectations with respect to security controls and procedures are properly managed.
- Activity and transaction logs are maintained for auditing and compliance for both parties.
- Trustworthy communications enable new business opportunities.
- Sensitive corporate data and systems are not exposed to unnecessary threats or vulnerabilities.
- The exchange procedures can be documented to create a repeatable guideline for subsequent partner agreements.

It also incurs the following liabilities:

- Complex negotiations with business partners may delay the implementation of a new project.
- The cost of a security audit or required controls may be beyond the financial capabilities of the partner entity—but necessary if they wish to do business with the enterprise.
- This pattern does not address integration issues (programmatic interfaces, process flow, messaging) between organizations or applications.

## **97.11. Variants**

[3] describes several patterns for enterprise partner communication. Each pattern builds on the previous pattern's flexibility and ability to meet the increasingly sophisticated demands of the enterprise:

- B2B Topology 1: Document Exchange
- B2B Topology 2: Exposed Application
- B2B Topology 3: Exposed Business Services
- B2B Topology 4: Managed Public Processes
- B2B Topology 5: Managed Public and Private Processes

IBM has also introduced the Trading Partner Agreement [4], an XML-based standard for defining ‘how trading partners will interact at the transport, document exchange and business protocol layers. A TPA contains the general contract terms and conditions, participant roles (buyers, sellers), communication and security protocols and business processes, (valid actions, sequencing rules, etc.).’ [5]

[6] describes security requirements that should be considered when allowing physical or logical access for on-site contractors, trading partners or support staff to enterprise data systems. Examples include the following:

- Description of each service to be made available
- Target level of service and unacceptable levels of service
- Right to monitor and revoke user activity
- Controls to ensure protection against malicious software
- Involvements of third parties with subcontractors

- Clear and specified process of change management

## **97.12. Known Uses**

These secure communication procedures are part of many enterprise policies for managing business relationships. For example, many enterprises require a provision in business contracts allowing them to perform a security audit of any third party. Since federal legislations are often the driving force behind this, audits are becoming more and more a priority for senior executives as they become personally responsible for the overall security of their organizations.

## **97.13. See Also**

## **97.14. References**

- [1] Swanson, D. (2000, October). Secure Strategies. *Information Security Magazine*.
- [2] Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278-1308.
- [3] IBM. (2004, February). Extended Enterprise Business Pattern. IBM Developer Works. <http://www-106.ibm.com/developerworks/patterns/b2bi/index.html>
- [4] Sachs, M., Dan, A., Nguyen, T., Kearney, R., Shaikh, H., & Dias, D. (2000). Executable trading-partner agreements in electronic commerce. In *IBM TJ Watson Research Center* (Vol. 10598).
- [5] Geller, M., & Peterson, J. (2000). Trading Partners Agreement Markup Language (tpaML). *Library Consortium Management: An International Journal*.
- [6] “ISO/IEC 17799:2005 Information Technology – Code of Practice for Information Security Management” International Organization for Standardization, 2000.

## **97.15. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **98. Enterprise Security Approaches**

## **98.1. Intent**

This pattern guides an enterprise in selecting security approaches, that is, prevention, detection, and response. Security approaches are driven by the security properties its assets require, such as confidentiality, integrity, and availability, and by assessed security risks. Security approaches also provide a basis for deciding what security services should be established by the enterprise.

## **98.2. Example**

A new wing of an existing museum of gemstones is to be opened. Business planning activities have provided an enterprise scope in terms of needs, concerns, and assets. Application of SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS has identified security properties applicable to each asset type. The dominant asset type for the museum is gemstones. Gems are valuable and should not be stolen or manipulated, so their required properties are availability and integrity. Another important asset type is documentation and records of gem properties, which require confidentiality, integrity, and availability. The museum needs to determine the security approaches most appropriate for achieving these required security properties, and how those approaches should be coordinated for the museum.

## **98.3. Context**

Business assets that require protection and their required security properties (confidentiality, integrity, and availability) are understood, for example from applying SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS. Enterprise or business unit security risks (not system risks) are sufficiently understood, for example, from applying RISK DETERMINATION and its closely related patterns.

## **98.4. Problem**

To integrate security into a business model, an enterprise or organization needs to determine preferred security approaches for achieving the security properties of its assets. Planning and operational diligence are security approaches that are always necessary to ensure effective security. In contrast, prevention, detection, and response are security approaches that may be applied in different proportions to each asset type and security property combination. Some business asset/property combinations will have one preferred approach. For example, critical assets that require the integrity property and that cannot be repaired or replaced will have a focus on prevention, since detection and response do not offer a solution to business impairment. On the other hand, assets that require the integrity property but are not critical, or that can be easily and cheaply repaired or replaced, will have a focus on detection of integrity problems and response (usually replacement).

How can security approaches be selected and integrated across an enterprise?

The forces applicable at the business model level of organization concerns are still abstract and are strongly intertwined with the business processes of the organization. The enterprise needs to resolve the following forces:

- The security properties identified for enterprise assets must be achieved.
- Security risks cannot be eliminated but can be significantly reduced by a combination of prevention, detection, and response approaches.
- For critical assets, prevention is preferable to recovery, that is, it is better to prevent a violation of security than to have the violation occur and then try to recover from it.
- Prevention is sometimes impossible to guarantee or is prohibitively expensive. A prevention mechanism can fail in the face of an unforeseen attack, but it can still be effective for the regular case.
- Some detection mechanisms can also facilitate prevention, especially when made obvious, such as a prominently displayed security camera, or a motion sensor that sets off a loud alarm.
- The costs of providing security must be kept to a minimum.
- Security should have minimal negative impact on business process performance and on users (for example, vendors, clients, staff).
- Continuity of operations must be maintained even in the face of security incidents, and you want to recover in a timely and satisfactory way from security incidents that cannot be prevented (for example disaster recovery).
- It should be possible to analyze security incidents to improve your approach.

## 98.5. Solution

Specify an integrated set of approaches that achieve the required security protection for each asset type. The process emphasizes two perspectives, namely, the individual perspective of each asset type, and a holistic perspective of the overall organization. For each asset type, systematically and explicitly examine a set of risk criteria to determine appropriate security approaches and their suggested business priorities. Risk criteria involve the security properties for an asset type, business risk analysis results regarding criticality of the asset, and other high-level business operations information. From a holistic perspective, ensure that the various approaches for asset types complement and reinforce each other, rather than work against each other.

The process of defining approaches is typically performed by an enterprise architect or strategic planner. The first step is to collect all the necessary information, including asset types and their security needs. Next, information on risk criteria that influence approaches is either collected or generated. Finally, approaches are selected and integrated.

## 98.6. Structure

Table 29 shows elements of the structure of this solution. Participating elements include humans involved in defining the solution for a specific situation. Participants also include primary elements of the process of defining a solution: security needs, security approaches, and selection criteria. More details of these three primary elements are also given in the table. The Implementation section gives additional common examples of selection criteria. Multiple

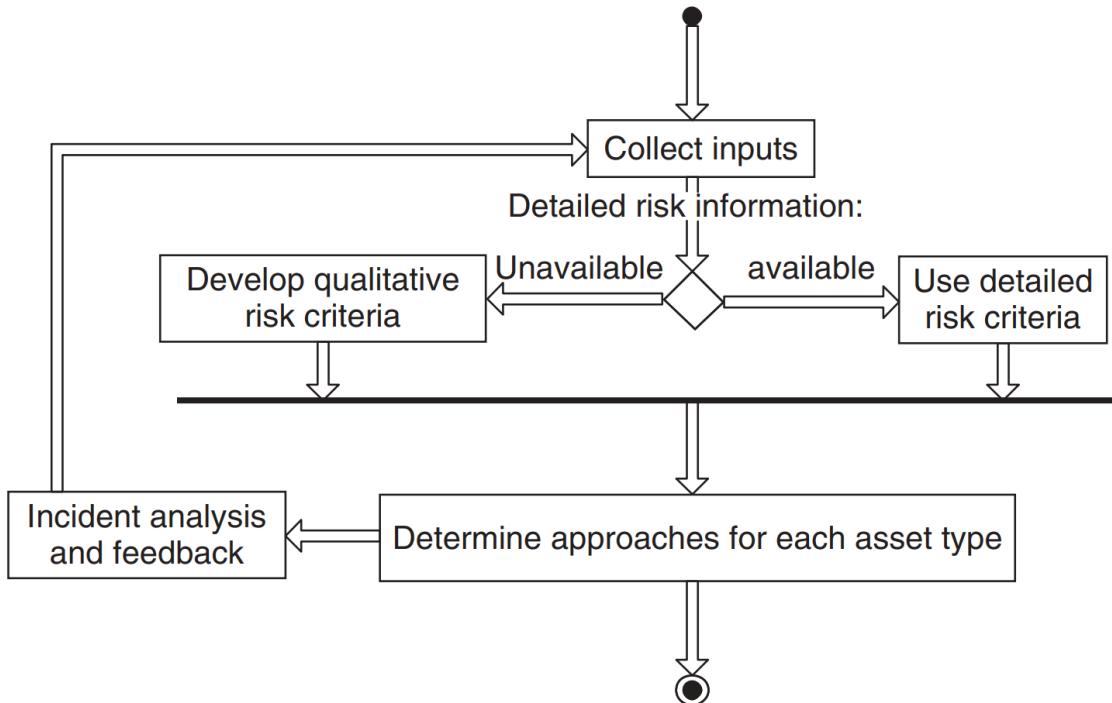
criteria apply to each security approach. More than one approach can be selected for each need.

Participating Element	Security Need	Security Approach	Selection Criterion
<ul style="list-style-type: none"> <li>• Business planner/controller</li> <li>• Enterprise architect</li> <li>• Enterprise security officer</li> <li>• Asset</li> <li>• Security need</li> <li>• Security approach</li> <li>• Selection criterion</li> </ul>	<ul style="list-style-type: none"> <li>• Confidentiality</li> <li>• Integrity</li> <li>• Availability</li> <li>• Accountability</li> </ul>	<ul style="list-style-type: none"> <li>• Prevention</li> <li>• Detection</li> <li>• Response</li> </ul>	<ul style="list-style-type: none"> <li>• Assets are irreplaceable</li> <li>• Asset loss prevents operations of critical business processes</li> <li>• Accountability is needed in case of legal ramifications</li> <li>• Assets must be repaired/restored as soon as detection occurs</li> <li>• ... (see implementation section)</li> </ul>

*Table 29: elements of selecting enterprise security approaches*

## 98.7. Dynamics

The process introduced in the Solution section is illustrated in the next figure. The process comprises three basic steps: collect information, identify security risk criteria, and determine security approaches for each asset type. The second step varies depending on whether sufficient risk information is available to understand the risk criteria that affect the security approach. If it is not available, some qualitative level of criteria must be developed.



*Figure 199: The process for selecting security approaches*

The figure also shows an analysis and feedback process. Decisions must be revisited because the world changes continuously. The figure shows feedback to the ‘collect inputs’ step, but feedback can go to any of the steps. In addition, if circumstances change sufficiently, feedback can extend beyond the scope of this pattern, to re-apply previous patterns such as RISK DETERMINATION.

## 98.8. Implementation

This section first provides further detail on the process, then presents criteria for selecting security approaches.

### 98.8.1. Process guidelines

1. Collect necessary input information:
  - a. Critical enterprise asset types
  - b. Basic security needs or properties for each asset type
  - c. Specific security risks for each asset type

Note that asset types and basic security needs might be obtained as a result of applying SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS. Similarly, specific security risk information obtained as a result of applying RISK DETERMINATION.
2. Identify security risk criteria that influence approaches:
  - a. If detailed risk information is available (for example, by applying RISK DETERMINATION), those criteria can be used here to determine which approaches to use: prevention, detection, response (also planning, operational diligence).

- b. If such detailed risk information is not available, qualitative risk criteria such as criticality, ease of replacement, cost of replacement, and harm to reputation can be defined and used here.
- 3. Determine which approaches to use for each asset type. More details about the association of types of security needed, risk criteria, and approaches are provided below.
- 4. Revisit approaches for each asset type as circumstances change.
  - a. Decisions to revisit may be time-driven, for example annually.
  - b. Decisions to revisit may be event driven. Examples are: (1) an organization makes a significant change to its business process, (2) a major law is passed that requires specific security measures, (3) an organization experiences a major security incident that calls into question its security approaches.

### **98.8.2. Approach Criteria**

For each asset type, appropriate security approaches and their suggested business priorities are determined based on desired security properties and risks. If detailed risks are available, for example, from applying the risk management pattern, they can be used to determine approaches. If such risks are not known or available, the qualitative selection criteria shown in Tables 30–33 can be used.

For example, Table 30 would be used to help determine approaches. If accountability is needed for an asset type due to legal ramifications, then detection is an indicated security approach with a high priority.

In using the above tables, it is important to understand that the information is generated from an overall organization perspective. In addition, the tables are not intended to cover all situations for a given organization. The example resolved in the next section will illustrate both of these points.

The focus on security approaches is typically documented as part of a security concept of operations. A security concept of operations presents approaches for addressing security properties and how the approaches work together to address security across the organization. The result should balance prevention, detection, and response into an appropriately layered set of defenses. Balance is needed among layered asset protections, such as entrances to museum spaces and gem display cases. Balance is also needed for the focus on approaches, such as prevention versus detection and response.

Business factors tend to present conflicting forces regarding appropriate balance. Some, such as laws and regulations, sensitivity of certain assets, and the desire to be viewed as a secure enterprise, encourage a high level of prevention. Others, such as cost constraints, the need for financial health, and a desire to be viewed as open and accessible, encourage a minimum degree of prevention with reliance on detection/ response. In cases in which the risk is sufficiently low, a ‘no action’ approach may be selected, that is, the approach is to take no measures of prevention, detection, or response. For example, theft of expensive clothes from a shop can be detected by security tags that sound an alarm when the goods are taken outside. But for very inexpensive clothes, the cost of security tags may exceed the cost of a few stolen items. The shop owner therefore may decide to make no response and just write off the loss.

The process of balancing these forces requires assets to be differentiated according to their importance to the organization. An investment in prevention is needed for critical assets, while a greater degree of risk may be accepted for non-critical assets.

- Critical assets typically are those whose loss or damage would cause significant harm to the organization, such as assets whose protection is required by law or strategic plans. Other critical assets are those that offer competitive advantage, are irreplaceable items, can impact the reputation of an organization, or whose loss would entail significant cost impact.
- Non-critical assets are those whose loss or damage would cause little or no harm to the organization, such as easily replaceable items, or information that could be divulged with little or no effect.

Obviously, there are many possible asset value gradations between non-critical and critical assets. Balancing forces and approaches can also exploit a fact that was mentioned in the discussion of forces: some detection mechanisms can also provide a measure of prevention. These are typically cases in which potential violators are made aware of detection mechanisms and possible accountability, such as prominently displayed surveillance cameras or loud alarms.

It is well known that many considerations are brought to bear at this level in determining an appropriate enterprise security strategy. Management may sometimes levy a requirement to address something specific for security that is realistically beyond what can be accounted for in this pattern. It is strongly recommended that such items be captured, so that when appropriate, they can be tracked through to implementation. In cases in which they are inappropriate, developers of the system model will be forewarned that these requirements will need to be revisited with management.

<b>Security Approach</b>	<b>Business Priority</b>	<b>Criteria Indicating Selection of Approach and Priority</b>
Detection	High Medium	<p>Accountability is needed in the case of legal ramifications.</p> <p>Validity of business communications and their signatures/sources must be ensured.</p> <p>Validity of business process flow/workflow (for example, chain of responsibility or signature) must be ensured.</p> <p>Assets are in a single or limited number of controllable/ observable locations.</p>
Response	High Low	<p>Means of unauthorized asset access must be closed immediately.</p> <p>Intrusion claims must be substantiated in order to pursue administrative or legal actions against unauthorized access to assets.</p> <p>Information asset is non-critical and does not require accountability.</p>

*Table 30: Criteria for approaches to achieve accountability*

<b>Security Approach</b>	<b>Business Priority</b>	<b>Criteria Indicating Selection of Approach and Priority</b>
Prevention	High	Asset loss prevents operations of critical business processes.

	High	Asset loss could result in irreparable harm to enterprise reputation.
	Medium	Asset loss severely impacts operations of critical business processes.
	Low	Asset loss could result in serious damage to enterprise reputation. Asset loss will impact business processes.
		Asset loss could result in ill will in client and/or customer base.
Detection	High	Total prevention of loss or alteration of assets is not possible.  Detection is cost-effective and prevention is not.  Asset can be replaced though very costly.
	Medium	Assets are in a single or limited number of controllable/ observable locations.
Response	High	Assets must be repaired/restored as soon as detection occurs.  Alterations to assets or other asset characteristics (for example functionality for software assets) must be completely identifiable for repair/replacement.  Means of unauthorized asset access must be closed immediately.  Intrusion claims must be substantiated in order to pursue administrative or legal actions against unauthorized access to assets.
	Medium	Assets are of moderate importance to enterprise functions and do not require confidentiality.
	Low	Particular enterprise assets interact only with non-critical functions.

Table 31: Criteria for approaches to achieve availability

<b>Security Approach</b>	<b>Business Priority</b>	<b>Criteria Indicating Selection of Approach and Priority</b>
Prevention	High	Asset reveals highly confidential or sensitive information.
	Medium	Asset reveals valuable information.
	Low	Asset reveals information.
Detection	Medium	Information assets can be made available in forms in which no damage can be done (for example, read-only forms, or 'sanitized' versions). Since tools to provide such forms are subject to risk, some protection is still needed.
	Low	Intrusions (that is, unauthorized attempts to read or write protected assets) denied, but awareness of them is needed.

Table 32: Criteria for approaches to achieving confidentiality

<b>Security Approach</b>	<b>Business Priority</b>	<b>Criteria Indicating Selection of Approach and Priority</b>
Prevention	High	<p>Asset critical and non-replaceable if corrupted or otherwise damaged.</p> <p>Asset extremely costly to replace or repair.</p> <p>Asset loss could result in irreparable harm to enterprise reputation.</p>
	Medium	<p>Asset very significant and requires long-lead time to replace or repair.</p> <p>Asset cost to replace very high.</p> <p>Asset loss could result in serious damage to enterprise reputation.</p>
	Low	<p>Asset significant but replaceable.</p> <p>Asset cost to replace or repair moderate.</p> <p>Asset loss could result in ill will in client and/or customer base.</p>
Detection	High	<p>Permanent asset alteration will significantly impair enterprise or operation of critical business processes.</p> <p>Total prevention of loss or alteration of assets is not possible.</p> <p>Detection is cost-effective and prevention is not.</p> <p>Asset can be replaced although very costly.</p>
	Medium	<p>Validity of business communications and their signatures/sources must be ensured.</p> <p>Validity of business process flow/workflow (for example, chain of responsibility or signature) must be ensured.</p> <p>Assets are in a single or limited number of controllable/observable locations.</p> <p>Information assets can be made available in forms in which no damage can be done (for example, read only forms or 'sanitized' versions). Since tools to provide such forms are subject to risk, some protection is still needed.</p>
	Low	<p>Enterprise information assets need to be accurate and support any/ all legal needs.</p> <p>Intrusions (that is, unauthorized attempts to read or write protected assets) denied, but awareness of them is needed.</p>

Response	High	<p>Assets must be repaired/restored as soon as detection occurs.</p> <p>Alterations to assets or other asset characteristics (for example, functionality for software assets) must be completely identifiable for repair/replacement.</p> <p>Means of unauthorized asset access must be closed immediately.</p> <p>Intrusion claims must be substantiated in order to pursue administrative or legal actions against unauthorized access to assets.</p>
	Medium	<p>Assets are repaired/replaced normally within three days of problem detection.</p> <p>Assets are of moderate importance to business functions and do not require integrity.</p>
	Low	<p>Assets should be restored within a week, but longer periods will not impair enterprise operations.</p> <p>Information asset is non-critical and does not require integrity.</p> <p>Particular enterprise assets interact only with non-critical functions.</p>

Table 33: Criteria for approaches to achieve integrity

## 98.9. Example Resolved

This section outlines portions of the result of applying the solution to a museum of gemstones. Identification of museum assets and their security properties is available from the Example Resolved section of SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS. Museum enterprise architects and planners have completed a business unit risk assessment for the new wing of the museum. The architects and planners must now work to identify security approaches for which the museum will be willing to allocate the resources necessary to achieve the security properties identified.

An example outcome is summarized in Table 34. The column for ‘Special notes’ has been included to show examples of special considerations and decisions that might be made by management while considering general security approaches.

Note that in the integration perspective, approaches are coordinated. For example, when prevention fails (thief grabs gem), detection and response act as a fallback (laser beam was interrupted, causing automatic doors to close before thief can leave the building).

Properties and Applicability	Security Approach	Business Priority for Approach	Special Notes
Protect Integrity of museum data: • Employee • Contractual	Prevent Detect Respond	High High High	Employee data should only be available to HR, staff, & management

<ul style="list-style-type: none"> <li>• Financial</li> <li>• Partner financial</li> </ul>			
Protect Integrity of all other museum data: <ul style="list-style-type: none"> <li>• Insurance</li> <li>• Business planning</li> <li>• Public data</li> </ul>	Prevent Detect Respond	Moderate High Low	While this information is very important, modifications can be detected and emended without high consequences
Protect Integrity of physical assets: <ul style="list-style-type: none"> <li>• Buildings</li> <li>• Collections/exhibits</li> </ul>	Prevent Detect Respond	High High High	This is a critical cost driver
Protect Confidentiality of museum data: <ul style="list-style-type: none"> <li>• Financial/insurance</li> <li>• Partner financial</li> <li>• Contractual</li> <li>• Exhibit plans</li> <li>• Research and its data</li> </ul>	Prevent Detect Respond	High High Moderate	These are critical to business operations. Management wants a focus on prevention and detection with high quality encryption.
Protect Confidentiality of employee data	Prevent Detect Respond	Moderate Moderate Moderate	Not as critical to business operations. Restrict access to HR, staff & management
Protect Availability of museum employee data	Prevent Detect	Moderate Moderate	HR is only user with critical availability concerns

*Table 34: Security approaches established for desired security properties*

## 98.10. Consequences

The following benefits may be expected from applying this pattern:

- The pattern fosters management level awareness: all enterprise security patterns help management better understand security as an overall issue and gives them terminology and simple understanding of the underlying concepts without relying on details of the technology used to implement them.
- It facilitates conscious and informed decision-making about security approaches to satisfy identified security needs.
- It promotes sensible resource allocation to protect assets.
- It allows feedback in the decision process, to better adjust security approaches to the situation at hand by traceability back to business factors and security needs.
- It encourages better balance among the security, cost, and usability of an asset.
- It shows that you can combine approaches to better and more cheaply protect an asset.

The following potential liabilities may result from applying this pattern:

- It requires an investment of resources to apply the pattern. In some cases, the cost of applying the pattern may exceed its benefits.
- It requires the involvement of people who have intimate knowledge of assets, and basic knowledge of asset security needs and security approaches. These people typically have high positions in the enterprise and their time is valuable.

- It is possible for an organization to assign people to this task who have a less than adequate knowledge of assets, security needs, or approaches, because they may have more available time or are less expensive. If the people applying the pattern do not have a good knowledge of enterprise assets and their value, the pattern results may be inaccurate or not useful.
- Perception of security needs can differ throughout an organization. This may make it difficult to reach agreement on priorities of approaches. On the other hand, bringing such disagreements to the surface may be a benefit, because they can then be properly discussed and resolved.

## **98.11. Known Uses**

The prevention-detection-response approaches identified in this pattern, and the process of associating them with risk criteria, are well-established functions in the security community. [1] refers to ‘the commonly mentioned prevention-detection response philosophy...’ In a security course description, [2] states that ‘general security practitioners, system administrators, and security architects will benefit by understanding how to design, build, and operate their systems to prevent, detect, and respond to attacks.’ Sometimes these approaches are included in a broader list of security functions or safeguards. [3] identifies these categories: planning, prevention, detection, diligence, and response. [4] states on page 44, ‘In general, safeguards may provide one or more of the following types of protection: prevention, deterrence, detection, reduction, recovery, correction, monitoring, and awareness.’ Criteria details in the Implementation section of this pattern are based on extensive MITRE Corporation experience with our customers.

## **98.12. See Also**

After applying this solution, the next step typically is to apply ENTERPRISE SECURITY SERVICES to select security services that support the approaches selected in this pattern.

## **98.13. References**

- [1] Chuvakin, A. (2002, April). Intrusion Detection Response.  
[http://www.linuxsecurity.com/feature\\_stories/ids-active-response.html](http://www.linuxsecurity.com/feature_stories/ids-active-response.html)
- [2] SANS Institute. (n.d.). Track 4: Hacker Techniques, Exploits and Incident Handling.  
<http://www.sans.org/kansascity04/description.php?track=t4>
- [3] DiDuro, J., Crosslin, R., Dennie, D., Jung, P., & Louden, C. (2002). *A Practical Approach to Integrating Information Security into Federal Enterprise Architecture*. LOGISTICS MANAGEMENT INST MCLEAN VA.
- [4] “Technical Report ISO13335-3 Part 3: Techniques for the Management of IT Security” International Organization for Standardization, American National Standards Institute, 2002

## **98.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security Patterns: Integrating Security and Systems Engineering* (1st ed.). Wiley.

# **99. Enterprise Security Services**

## **99.1. Intent**

This pattern guides an enterprise in selecting security services for protecting its assets, after the required security approaches—prevention, detection, response—have been identified. It helps to establish the level of strength or confidence each security service should offer, based on priorities. Primary examples of such services are identification and authentication, accounting/auditing, access control/authorization, and security management.

## **99.2. Example**

A new wing of an existing museum of gemstones is to be opened. The museum's management has already identified security as an enterprise concern and determined appropriate security properties and approaches to be supported. Now the management needs to identify what security services will be used. A specific asset group is used in this simple example problem.

The museum has identified three specific gems as irreplaceable due to their financial value. They can only be insured for approximately two-thirds of their actual monetary value. The museum wants to provide integrity and availability for physical protection of the gems, but also confidentiality for the real value of the assets. The museum has determined that prevention will be the primary approach to providing integrity and availability of the gems. Prevention will also provide confidentiality for information that stipulates real monetary values. Detection and response will provide secondary approaches to protecting these gems and resources will be allocated to prevention first. The museum now needs to determine what abstract security services will support the desired properties and approaches.

## **99.3. Context**

Business strategies, plans, and operations are understood. These include disaster recovery and continuity of operations strategies, a semantic data model, high-level business process and workflows, business locations, organizational units, and business cycle models. Security approaches (prevention, detection, response) and their priorities have been selected to satisfy the identified security needs of enterprise assets. The approaches might have been selected by applying ENTERPRISE SECURITY APPROACHES. The pattern user has a basic awareness of potential security services.

## **99.4. Problem**

To fully integrate security into the business model, business planners need to identify the security services needed to protect each category of enterprise asset. Selection of security services will need to balance the resources the museum is willing to allocate in order to address security approaches appropriately. At the business level, planners provide direction about how much emphasis to focus on preventing security incidents, detecting incidents after the fact, and the level of focus for responding to security incidents. Some services, such as access control, emphasize a prevention approach. Other services, such as accounting,

emphasize detection and response. Still others, such as identification and authentication, support both prevention and detection.

How do you select and integrate security services across the organization to support security properties using preferred security approaches?

The forces applicable at the business model level of concerns are still abstract and are strongly intertwined with business processes. The enterprise needs to resolve the following forces:

- Customers and clients expect suitable protection of their assets
- Unauthorized access to critical assets that require prevention as the primary protection must be prevented
- A strong ability to discover security incidents provides protection for assets that require detection as a primary approach
- It is necessary to be able to recover from, or actively respond to, incidents for assets where prevention is not suitable or where prevention fails
- Accurate actor identification provides more protection when actors access critical assets
- Strong security services provide greater asset protection, but tend to be harder to use
- Weak security services tend to be easier to use, but provide less asset protection

## 99.5. Solution

Specify an integrated set of security services to address identified security approaches and security properties for each asset type. This process emphasizes two perspectives, namely, the individual perspective of each asset type, and a holistic perspective of the overall organization. Assets can vary greatly. This pattern therefore focuses on associations of security approaches and security services to assist the user in understanding relationships that can then be applied to asset categories. The Implementation section below provides examples. The examples are to help the pattern user to establish a particular set of security services to address all asset security needs for a given organization. From a holistic perspective, it ensures that the various approaches for asset types complement and reinforce each other, rather than work against each other.

The process of defining security services is typically performed by an enterprise architect and systems engineer. The first step is to collect all necessary information, including the asset types and security approaches that have been defined—for example, by applying ENTERPRISE SECURITY APPROACHES. Next, services are selected for each asset type and integrated. Finally, a ‘human touch’ is involved in applying an enterprise level pattern such as ENTERPRISE SECURITY SERVICES. Its application helps to shape thoughts about security, but it never can be a one-shot solution. You need feedback and conscious re-visiting of your decisions because the world and organization change continually. Any of the earlier steps in this process might be revisited. In addition, if circumstances change sufficiently, feedback can extend to the beginning of the reasoning chain, to re-apply previous patterns such as ENTERPRISE SECURITY APPROACHES. More details on the process are provided in the Implementation section below.

After applying this solution, the next step typically is to specify requirements for the selected security services—for example, by applying one of these patterns: I&A REQUIREMENTS, ACCESS CONTROL REQUIREMENTS, or SECURITY ACCOUNTING REQUIREMENTS. It is important

to note that ENTERPRISE SECURITY SERVICES is organization-wide, while the scope of each service requirements pattern is a system or security domain within the organization.

## 99.6. Structure

Table 35 shows elements of the structure of this solution. Participating elements include humans involved in defining the solution for a specific situation. Participants also include primary elements of the process of defining a solution: security approaches, selection criteria, and security services. More details of these three primary elements are also given in the table. The implementation section below gives additional common examples of selection criteria. Multiple criteria apply to each security approach and to each security service. More than one service can be selected for each approach.

Participating Element	Security Approach	Selection Criterion	Security Service
<ul style="list-style-type: none"> <li>• Business planners/controllers</li> <li>• Enterprise architect</li> <li>• Enterprise security officer</li> <li>• Asset</li> <li>• Security approach</li> <li>• Selection criteria</li> <li>• Security service</li> </ul>	<ul style="list-style-type: none"> <li>• Prevention</li> <li>• Detection</li> <li>• Response</li> </ul>	<ul style="list-style-type: none"> <li>• Assets are irreplaceable</li> <li>• Continuous record of asset protection is required</li> <li>• Need for daily asset access accounting</li> <li>• System cannot be down more than 8 hours</li> <li>• Financial data could harm partnerships</li> <li>• ... (see implementation section)</li> </ul>	<ul style="list-style-type: none"> <li>• I&amp;A</li> <li>• Access control</li> <li>• Accounting</li> <li>• Security management</li> <li>• ...</li> </ul>

Table 35: Elements of enterprise services solution

## 99.7. Dynamics

## 99.8. Implementation

This section first provides further detail on the process that was summarized in the Solution section, then presents criteria for selecting security services.

### 99.8.1. Process Guidelines

1. Collect necessary input information:
  - a. Critical enterprise asset types.

- b. Basic security needs or properties for each asset type. Asset types and basic security needs might be obtained as a result of applying SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS.
  - c. Specific security approaches for each asset type, including prevention, detection, and response, and the business priority for the approach in each case. Specific approaches and priorities might be obtained as a result of applying ENTERPRISE SECURITY APPROACHES.
2. Determine which security services to use for each asset type and approach:
- a. Determine the factors that apply to your organization
  - b. Identify services that support the approaches, based on applicable factors
- Note that one possible response is to take no action, that is, to accept the risk or ignore the incident, in which case no security service is designated.
- More details on relating security approaches to security services are provided below.
3. Revisit security services for each asset type as circumstances change:
- a. Decisions to revisit may be time-driven, for example annually.
  - b. Decisions to revisit may be event driven. Examples are: (1) an organization makes a significant change to its business process, (2) a major law is passed that requires specific security measures, (3) an organization experiences a major security incident that calls into question its security services.

### 99.8.2. Approach criteria

Tables 36–38 correlate security approaches with security services and a business priority. The criteria indicating selection provide typical examples of instances when the organization has set a business priority at a certain level. Example mechanisms that may be employed to offer the service are also provided. Note that these tables could not possibly address all possible security services—instead they focus on fundamental services that will provide a basis for security.

Rows of the tables may be interpreted as follows, using as an example Table 36, which addresses prevention as the approach. An organization has identified a need for prevention. I&A is a security service selected as a means of supporting prevention of unauthorized operations on assets. The organization has established prevention as a high business priority for a given asset category, such as irreplaceable assets. In this circumstance a strong I&A service is needed. It may be implemented through the use of both biometrics and passwords structured as multiple authentication layers. Suppose another asset category has a moderate need for prevention, such as assets replaceable but at significant cost. In this case a moderately strong I&A service is needed. It may be implemented through use of biometrics by itself, or use of a token generator. Finally, if the prevention priority for an asset category is low, then a weaker I&A service is needed. This may be implemented through randomly generated passwords.

The example implementations are not decided in this pattern. The first three columns—service, priority, and criteria—represent organization-wide decisions made in the scope of this pattern. The fourth column, example mechanisms, represents system decisions made in the scope of each system security architecture.

Security Service	Business Priority	Criteria Indicating Selection	Example Security Mechanisms
------------------	-------------------	-------------------------------	-----------------------------

Access control	High	Enterprise has irreplaceable assets	Categorize access to assets according to roles and responsibilities, and restrict access to individuals via their roles/responsibilities
	Moderate	Assets can be damaged deliberately or inadvertently	Encapsulate assets (for example, envelope, encrypt, vacuum)
	Low	Assets require basic level of protection for insurance purposes	Provide physical protection controls
Accounting	High	Continuous record of asset protection is required (for example by a contract)	Real-time audit trail for information assets or sensors for physical assets
	Moderate	Asset access limited and must be accounted for	Pre-defined job functions in organization associated with user roles
	Low	Asset access physically limited but videotapes of area-access required	Videotape of assets, predefined job locations
I&A	High	Enterprise has irreplaceable assets (for example, extremely costly, value lost if modified, not insurable, one of a kind)	Use multiple authentication layers (for example biometrics and passwords)  Store identities on smart card with biometric authenticator
	Moderate	Assets replaceable at significant cost	Use token generator for identity authenticator  Restrict access to I&A information
	Low	Assets can be replaced as long as problems are detected	Use unguessable authenticator (for example randomly generated passwords)
Security management	High	User I&A information alterable by single identified person	Only security officer can alter I&A information  All I&A information is encrypted in storage and transfer
	Moderate	Only select roles may alter I&A information	SSO and System Administrator can alter I&A information  All I&A information is encrypted in transfer
	Low	I&A information should not be easily modified	Access to server where I&A information can be altered is restricted

Table 36: Correlating prevention with security services and business priorities

Security Service	Business Priority	Criteria Indicating Selection	Example Security Mechanisms
------------------	-------------------	-------------------------------	-----------------------------

Access control	High	All events needing immediate attention can be specifically identified	Accounting service mechanisms will need extreme granularity Access controls will relay to real time audit trail
	Moderate	Normal/abnormal functionality is identified and controlled	Accounting service mechanisms provide daily audit trails for all information system functionality
	Low	Prevention is highest priority for organization	Select access control activities are reported to accounting service mechanisms for documenting
Accounting	High	Business records need to be accurate and support any/all legal needs	Document all initial business records, any changes to them, and actor involved, in non-repudiable manner
	Moderate	Inability to recover from incident could weaken reputation	Ensure audit trails are reviewed for early detection of incidents
	Low	Need to recover from environmental disruptions	Maintain a history of all business records so that emergencies can be recovered from
I&A	High	Critical interactions are only authorized for specific staff	Use intrusion detection to detect any unauthorized interactions
	Moderate	Sensitive information restricted	Keep complete audit trails for all access to sensitive information
	Low	Need for daily asset access accounting	Assets need identifiers for differentiation
Security management	High	All security management information is company sensitive	Access control is enforced continuously for all this information  This information for accounting cannot be altered
	Moderate	All security management information is selectively accessible	Access control with roles ensures the information is current and valid
	Low	Security management information must be periodically reviewed, and changes documented	Audit trails must include changes by security officer and system administrators

Table 37: Correlating detection with security services and business priorities

Security Service	Business Priority	Criteria Indicating Selection	Example Security Mechanisms
Access control	High	Assets are nationally sensitive (for example nuclear plants)	Access requires specific permissions and is restricted by time of day, location, and so on
	Moderate	Financial data could harm partnerships	Access is restricted to a specific community of interest

	Low	Only HR employees should access corporate personnel data	Access authorizations are established by department functions
Accounting	High	Location/condition of specific assets must not be altered	Accounting records must provide continuous monitoring (for example videotape) with immediate alert locking asset location on any change
	Moderate	Any unauthorized changes to asset must initiate notification	If detection indicates unauthorized asset change, accounting service mechanism must send notification to system administrator
	Low	Only physical disasters need immediate responses	Accounting service mechanism only notifies select staff if physical catastrophe occurs
I&A	High	Unknown users must be immediately locked out permanently	Identification used with biometrics secured on a token  I&A service mechanism does not have high false positive or negative ratios  I&A part of layered defense
	Moderate	When user I&A provides warning, additional means are used to reduce possibility of false positive	Front door human guard, badging system with photo, and identifier and password on automated system
	Low	Users are not allowed to repeatedly provide invalid log-in information	Computer system locks down after preset number of invalid attempts
Security management	High	System cannot be down more than eight hours	Security management plans and procedures for contingency operation are in place and assure response in 8 hours
	Moderate	System cannot be down more than twenty-four hours	Security management plans and procedures for backup and recovery will restore a functioning system in twenty-four hours
	Low	System information must be accessible online in two weeks	System backups with all security management information must be run every two days and recovery plans are in place

Table 38: Correlating response with security services and business priorities

## 99.9. Example Resolved

This example expounds on the problem example provided earlier. As noted, the museum has gems that are irreplaceable and only partially insurable. They have a business priority for ensuring their integrity and availability by preventing their theft or any damage. The museum will therefore need to have strong I&A, access control, accounting, and security management services to protect the gems. Detection and response security approaches will also be provided.

as backups for the prevention approach. To provide integrity and availability for the detection approach, both I&A and accounting security services will be needed. For the response approach the security management service will also need to be dependable.

The museum also indicated a real need to protect confidentiality of the real value of these gems by preventing that information from being easily obtained. In addition, the museum will need to ensure integrity of that information. This additional consideration for integrity of gem values to have a high business priority will need to be fed back into the earlier work to ensure it is captured. There is a high business priority for prevention of any lapses of confidentiality and integrity of gem data on insurance contracts, attributes (carats), purchase amounts, and appraisal values. To achieve the required prevention approach, stringent I&A, access control, and security management services will be needed. To achieve the required prevention as well as detection and response for preventing integrity violations of gem data, strong mechanisms for all four identified services will be needed.

The museum has now reached a point at which they can begin to determine refinements for security services appropriate to support abstract selected services. Table 39 captures the museum's resolution of abstract security services to be used.

Museum Asset	Security Property	Security Approach	Business Priority	Selected Service
High value gems	Integrity availability	Prevention	High	<ul style="list-style-type: none"> <li>• I&amp;A</li> <li>• Access control, e.g., locked glass display</li> <li>• Accounting</li> <li>• Security management</li> </ul>
	Integrity availability	Detection	Medium	<ul style="list-style-type: none"> <li>• I&amp;A</li> <li>• Accounting, e.g., surveillance camera</li> </ul>
	Integrity availability	Response	Medium	<ul style="list-style-type: none"> <li>• I&amp;A</li> <li>• Accounting</li> <li>• Security management</li> </ul>
Gem insurance contracts, attribute data (i.e., carats), purchase data, and appraisal data	Confidentiality	Prevention	High	<ul style="list-style-type: none"> <li>• I&amp;A</li> <li>• Access control, e.g., a safe</li> <li>• Security management</li> </ul>
	Integrity	Prevention, Detection, Response	High	<ul style="list-style-type: none"> <li>• I&amp;A</li> <li>• Access control</li> <li>• Accounting</li> <li>• Security management</li> </ul>

Table 39: Protecting Museum assets

## **99.10. Consequences**

The following benefits may be expected from applying this pattern:

- The pattern fosters management level awareness: all enterprise security patterns help management to better understand security as an overall issue and gives them terminology and simple understanding of the underlying concepts without relying on details of the technology used to implement them.
- It facilitates conscious and informed decision making about security services to support identified security approaches.
- It promotes sensible resource allocation to protect assets.
- It allows feedback in the decision process to better adjust security services to the situation at hand by traceability back to business factors and security needs.
- It encourages better balance among security, cost, and usability of an asset.
- It shows that you can combine services to better and more cheaply protect an asset.

The following potential liabilities may result from applying this pattern:

- It requires an investment of resources to apply the pattern, including time to analyze enterprise assets and security approaches. In some cases, the cost of applying the pattern may exceed its benefits.
- It requires the involvement of people who have intimate knowledge of assets, and basic knowledge of asset security needs and security approaches. These people typically have high positions in the enterprise and their time is valuable. On the other hand, the pattern allows more people to be aware of the issues, so that after the initial investment of time, other people can be in a position to maintain and evolve the service selection.
- It is possible for an organization to assign people to this task who have less than adequate knowledge of assets, approaches, or services, because they may have more available time or are less expensive. If the people applying the pattern do not have good knowledge of enterprise assets and their value, the pattern results may be inaccurate or not useful.
- Perception of security needs can differ throughout an organization. This may make it difficult to reach agreement on priorities of services. On the other hand, bringing such disagreements to the surface may be a benefit, because then they can be properly discussed and resolved.

## **99.11. Known Uses**

The prevention-detection-response approaches identified in this pattern are well established functions in the security community. Likewise, security services identified in this pattern are well-established, although there is lack of consensus on names for some of them, notably accounting. To a significant degree, criteria details in the Implementation section of this pattern are based on extensive MITRE Corporation experience with our customers. There are also some standards that include related information. For example, [1] discusses services and mechanisms—under the name ‘safeguards’—such as I&A, access control, audit, and security management, and associates these with security properties such as confidentiality and integrity. [2] describes a security services model that includes identification, authentication,

access control, audit, non-repudiation, and security administration services. The latter also maps services to a set of primary purposes or approaches: prevent, recover, and support.

A specific example of how a prevention approach leads to use of the access control service is the Cisco use of Access Control Lists to protect networks, described in [3]. Examples of how accounting in the form of audit software supports detection of fraud are described in [4].

## **99.12. See Also**

## **99.13. References**

- [1] "Technical Report 13335-4:2000 Information Technology - Guidelines for the Management of IT Security - Part 4: Selection of Safeguards" International Organization for Standardization, 2000-03-01
- [2] Stoneburner, G. (2001, December). Underlying Technical Models for Information Technology Security: Recommendations of the National Institute of Standards and Technology {USA}. NIST Special Publication SP800-33. National Institute of Standards and Technology (NIST)
- [3] JANET-CERT. (n.d.). Cisco ACL Example. [http://www.ja.net/CERT/JANET-CERT/prevention/cisco/cisco\\_acls.html](http://www.ja.net/CERT/JANET-CERT/prevention/cisco/cisco_acls.html)
- [4] Coderre, D. (1999). Computer-assisted techniques for fraud detection. *The CPA Journal*, 69(8), 57.

## **99.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.

# 100.I&A Requirements

## 100.1. Intent

An identification and authentication (I&A) service must satisfy a set of requirements for both the service and the quality of service. The function of I&A is to recognize an individual and validate the individual's identity. While each situation that calls for I&A is unique, there are common generic requirements that apply to all I&A situations. This pattern provides a common generic set of I&A requirements. The pattern also helps you to apply the general requirements to your specific situation and helps you to determine the relative importance of conflicting requirements.

## 100.2. Example

The museum gemstones wing will build on the parent museum's intranet, with workstations distributed throughout multiple departments. Based on applying ENTERPRISE SECURITY SERVICES, the museum recognizes the need for specific security functions. Among these are access control and security accounting. Both of these functions rely on an I&A service to establish identity. What kind of I&A service does the museum need to support these functions? What other situations call for a I&A service in the museum? After some analysis, Samuel, the museum's system engineer, and Edward, the enterprise architect, come up with these possible situations that require I&A services:

- Establishing physical access to the museum during business hours
- Establishing physical access to the museum by staff during outside business hours
- On-line access to the intranet from within the local area network of the museum wing
- Remote on-line access to the museum's intranet
- Access to highly sensitive museum physical assets, especially gemstones
- On-line access to highly sensitive museum information assets
- Tracking who is downloading information from the publicly available museum Web site
- Support non-repudiation of business transactions on the part of customers or partners
- Employee accountability of computer and network resource use within the museum
- Accountability to support identification of the source of computer viruses or a network denial of service attack

The engineers feel that these situations differ in their I&A requirements but are not sure how to capture the differences. For example, a single mechanism for Web site I&A doesn't work, because Vic the visitor, downloading publicly available information, has I&A requirements that differ from Manuela the museum manager, who is working from home and retrieving sensitive accounting data. Furthermore, for each of these situations, the museum wants to properly balance conflicting objectives, such as a service that detects would-be hackers versus a service that is easy for employees to use. Typically, a strong I&A mechanism that detects most imposters, that is, people who falsely claim to be legitimate, are hard to use, while I&A mechanisms that are easy to use tend to give weaker protection, in too many cases concluding that an imposter is legitimate.

Samuel's initial thinking had been that he could simply select some I&A mechanism such as a password-based log-on or an employee badge. This has given way to the realization that more thought and consideration is needed to ensure that multiple I&A needs are properly addressed. Samuel and Edward recognize that they need to specify a clear and balanced set of requirements for each situation the requires I&A. How can they accomplish this?

### **100.3. Context**

An organization or project understands its planned uses of I&A, for example, from applying ENTERPRISE SECURITY SERVICES, or from applying one or more of the pattern systems that use I&A, such as the patterns for access control or accounting patterns.

The scope is known to be situations in which both identification and authentication are needed. Other situations exist in which only identification is needed without authentication, but those situations are not addressed in this pattern.

### **100.4. Problem**

Requirements for I&A often conflict with each other, and trade-offs among them are often necessary. The conflict stated in the Example section is that strength of protection with I&A tends to conflict with ease of use.

I&A comprises both associating an identifier with an actor (identification) and verifying that the association is correct (authentication). I&A is a security service whose results are often used by other security services, including access control and accounting. A basic set of generic I&A requirements exists for all types of use and circumstances. However, these generic requirements need to be specialized for a given I&A domain. In addition, the relative importance of the requirements will vary based on the circumstances. What is needed is (1) to capture the specific set of requirements, and (2) to understand how to differentiate the relative importance of the requirements in specific circumstances to balance or resolve the conflicts.

How can you determine specific requirements for an I&A service, and their relative importance?

Determination of I&A requirements needs to resolve the following forces:

- Owners of I&A services want the services to perform their expected function, that is, correctly to determine whether an actor is associated with an identifier
- Incorrectly confirming the false claim of an imposter can lead to extensive disclosure of or damage to assets
- Incorrectly denying the true claim of a legitimate actor can lead to loss of productivity through denial of service or denial of access to authorized assets
- Users want I&A services to offer good quality of service: rapid response, proper functioning, easy to understand, safe, appropriate for category of user, and supportive of handicapped users
- The enterprise wants its I&A services to be cost effective and provide a good return on investment
- There are often reasons for making identifiers public—for example, e-mail identifiers need to be known so that others can send e-mail

- The I&A service will need to protect against the potential for stolen identities and the impacts of stolen identities and authenticators

## 100.5. Solution

Specify a set of I&A requirements for a specific I&A domain and determine the relative importance of each requirement. The solution has two aspects: a requirements process and a common set of generic requirements.

### 100.5.1. Requirements Specification and Prioritization Process

A system requirements engineer, in conjunction with an enterprise architect, typically perform the requirements process. An important first step is explicitly to define the domain for which you are specifying I&A requirements, such as a specific system or facility. Factors such as enterprise constraints that affect specialization and importance of requirements also need to be defined. The I&A requirements for the target I&A domain are then specified, using the generic requirements provided below. The final activity is to define the relative importance of the specified requirements.

### 100.5.2. Generic Requirements Description

The following set of generic requirements responds to the problem and forces described above. The first two represent I&A functional requirements. The remaining requirements represent I&A non-functional requirements, including requirements for security of the I&A service.

The following analysis is presented to help you understand and apply the first two generic requirements. For the I&A service properly to execute its function, it must be able to deny the identity claims of imposters and confirm the claims of legitimate actors. In any given I&A episode, any of four outcomes is possible, illustrated in Table 40.

Actual Situation	I&A Service Conclusion	
	Confirmation of actor claim (You are Actor A)	Denial of actor claim (You are not Actor A)
Actor A claims to be Actor A	True positive	False negative
Actor B claims to be Actor A	False positive	True negative

Table 40: Outcome of I&A situations

The table shows that the four outcomes result from two variables, namely, the actual situation and the I&A service conclusion. One function of the I&A service is to confirm the identity of legitimate actors, that is, actors who are who they claim to be. This result is a true positive or true acceptance. The second function of the I&A service is to deny the identity of imposters, that is, actors who are not who they claim to be. This result is a true negative or true rejection. A perfect I&A service would result in 100% true positives in situations where the actor is legitimate, and 100% true negatives in situations where the actor is an impostor. But no, I&A service is perfect, and two types of errors are possible. One type of error, called a ‘false positive’ or ‘false acceptance,’ is confirmation that an imposter is who he claims to be. In Table 40, the false positive is erroneous confirmation that Actor B is Actor A. The second type of

error, called a false negative or false rejection, is denial that an actor is who he claims to be. In Table 40, the false negative is erroneous denial that Actor A is Actor A. If an error occurs, it is then propagated to the function that relies on the I&A service. For example, an access-control service may use a false positive from the I&A service to permit an imposter access to sensitive assets, which can then be damaged or destroyed.

The generic requirements are as follows.

#### **100.5.3. Accurately Detect Imposters**

In the context of Table 40, this requirement addresses the imposter situation, that is, Actor B claims to be Actor A. The requirement says that the I&A service must recognize that this actor is not Actor A and deny the claim. The service must result in a true negative and not make the false positive error. Note that this requirement does not ask the I&A service to recognize Actor B as Actor B, but only to recognize that this actor is not the claimed Actor A.

#### **100.5.4. Accurately Recognize Legitimate Actors**

In the context of Table 40, this requirement addresses the legitimate actor situation, that is, Actor A claims to be Actor A. The requirement says that the I&A service must recognize that this actor is Actor A and confirm the claim. The service must result in a true positive and not make the false negative error.

A trade-off exists between this requirement and the previous one. A stringent I&A service that provides a very high probability of detecting imposters also tends to have a higher probability of denying the claims of legitimate actors. Conversely, a more accommodating I&A service that provides a very high probability of confirming legitimate actors also tends to have a higher probability of confirming the claims of impostors. When you apply this pattern to your specific system or organization, you need to determine which type of error is more important to avoid.

#### **100.5.5. Minimize Mismatch with user Characteristics**

A I&A service is typically used by different categories of users, such as level of experience. Both inexperienced users or novices, and experienced or sophisticated users, want the I&A services to interact with them at their own level. Some I&A techniques require more sophistication than others. Additional characteristics to be considered include fixed versus mobile location of users, and remote versus local users.

#### **100.5.6. Minimize Time and Effort to Use**

Performing I&A almost always costs users some time and effort in the process of acquiring access to an enterprise asset. For example, remembering and typing a password, or standing in line to be approved by a security guard, or assigning and maintaining certificates associated with a software module, is not as easy as not typing the password, not needing to wait for the guard, or not assigning the certificate. User effort and time delays associated with I&A adds to the bottom-line cost of enterprise operations, so that in general it is desirable to minimize the effort and time involved in performing I&A. Single sign-on (SSO) is one common approach to minimizing time and effort in the context of an enterprise network, by means of a single

authentication that is performed when users initially access the network. This requirement is often in conflict with accuracy requirements, however.

#### **100.5.7. Minimize Risks to User Safety**

Issues of safety, such as requiring use of iris-scanning if users could be wearing gas masks, or damage done by retinal scanning, can preclude use of an authentication technique. This requirement is sometimes in conflict with accuracy requirements.

#### **100.5.8. Minimize Costs of Per-user Setup**

Establishing a new user or actor in an I&A domain involves generating an identifier for the actor, establishing grounds for authenticating the actor, delivering to the actor any data, tokens, or hardware the actor needs, and training users or software maintainers in the use of the selected technique. Each of these procedures has associated costs that add to the bottom-line costs of establishing or maintaining supported functions. Cost should in general be minimized. This requirement is often in conflict with enterprise accuracy requirements.

#### **100.5.9. Minimize Changes Needed to Existing System Infrastructure**

System infrastructure includes equipment, facilities, people, and procedures. System infrastructure support for I&A includes both system-wide support and support at each connection point where actors interact with I&A services.

Changes to existing infrastructure or addition of new infrastructure have associated costs. For example, new equipment costs money to acquire, absorbs employee time to install, and carries maintenance costs. All these costs add to the bottom-line costs of establishing or maintaining supported functions, and in general should be minimized. This requirement is often in conflict with minimizing enterprise accuracy requirements.

#### **100.5.10. Minimize Costs of Maintenance, Management, and Overhead**

I&A is a business procedure that can require very substantial time and effort to maintain and manage. All these costs add to the bottom-line costs of running the business and in general should be minimized. This requirement is often in conflict with accuracy requirements.

#### **100.5.11. Protect I&A Service and Assets**

I&A assets, especially authenticators and related data, are vulnerable to theft or disclosure. The I&A service itself needs protection, including confidentiality and integrity of I&A data, availability of the I&A process, and accountability for I&A service-related actions. This requirement is supportive of accuracy requirements but is often in conflict with ease of use.

### **100.5.12. Variations Across Sets of Requirements**

The specific values of requirements, and the relative importance of each requirement, vary in different use situations. The use situations given in the Problem section illustrate some of these differences. For example:

- I&A results used in granting on-line access to highly sensitive enterprise information assets would be likely to place high importance on avoiding false rejections and protecting I&A assets, and lesser importance on minimizing cost and effort to use.
- I&A results used in tracking who is downloading information or products from publicly available enterprise Web site would be likely to place high importance on minimizing cost and effort to use, and lesser importance on avoiding false rejections.

## **100.6. Structure**

## **100.7. Dynamics**

## **100.8. Implementation**

This section first provides further detail on the process that was summarized in the Solution section, then discusses factors for determining the relative importance of requirements.

### **100.8.1. Process Guidelines**

The requirements process typically includes these steps:

1. Establish the domain for which the I&A service is needed. Ensure that the domain has been identified and scoped. Typical I&A domains include an information system, physical facility, network, portal, or entire enterprise. Other constraints may bound the domain—for example, the I&A requirements for entering a designated facility during normal work hours may differ from the requirements outside business hours, such as night-time and weekends: these would represent two domains.
2. Specify a set of factors that affect specialization and importance of requirements. The factors include uses of I&A, I&A needs, enterprise constraints, and priorities. You can find a general candidate set of factors in Table 41.
3. Specify I&A requirements for the target I&A domain. To do this, specialize the set of generic requirements given in the Solution section.
4. Define the relative importance of specific requirements. The association of factors and requirements is discussed below.

## 100.8.2. Factors in Determining Relative Importance

Table 41 presents factors for judging the relative importance to the enterprise of the generic I&A requirements that were identified in the Solution section. For each requirement, the table describes how the factors affect the relative priority of the requirement.

<b>Generic Requirement</b>	<b>Factor</b>	<b>Impact on Priority</b>
Accurately detect imposters	Potential cost to the enterprise if a link is made with an identifier to which the actor is not entitled (for example, could be used to give access to assets)	This requirement should have increased priority if inability to detect imposters could cause significant damage to the enterprise or system.
Accurately recognize legitimate actors	Existence of time-critical functions where access is controlled based on actor identifier, potential cost to the enterprise if controlled critical functions are not performed in a timely manner.	This requirement should have increased priority if rejection of legitimate actors for time-critical functions could cause significant damage to the enterprise or system.
	User base sensitivity to temporary denial of service. potential cost in dollars or goodwill to the enterprise if users become annoyed	This requirement should have increased priority if rejection of legitimate actors could occur to the point of significant denial of service and user annoyance.
Minimize mismatch with user characteristics	User experience	This requirement should have increased priority if the I&A service could cause significant user frustration by not accommodating user experience level, whether novice or sophisticated.
	User base membership (employees, partners, public, software)	This requirement should have increased priority if the I&A service could cause security risks or significant user frustration by not supporting all user categories, such as employees versus partners.
	User location (local, remote)	This requirement should have increased priority if the I&A service could cause security risks or significant user frustration by not supporting all user locations, such as local versus remote.
	User mobility (fixed or mobile locations, fixed or variable devices)	This requirement should have increased priority if the I&A service could cause security risks or significant user frustration by not supporting user mobility, such as fixed versus mobile locations.

Minimize time and effort to use	Frequency of use	This requirement should have increased priority if the I&A service has heavy use.
	User base characteristics	Inability of some users, such as handicapped users, to perform I&A may require changes to the business model. A I&A service that is difficult to use and requires more time may increase the potential costs in money or good-will if users become annoyed. Both should increase the priority of this requirement.
Minimize risks to user safety	Relevant statutes and enterprise policy	Statutes or policy may mandate this requirement, in which case it would in effect have top priority.
	Potential liability of enterprise for injury (for example damage to eye in retinal scan)	This requirement should have increased priority if the I&A service poses significant risk of incurred costs, and negative publicity, from users injured performing I&A.
Minimize costs of per-user setup	Number of users in general terms (hundreds, thousands, millions)	The existence or projection of a large number of users should increase the priority of this requirement.
	Volatility of user base	The existence or projection of a large turnover rate among users should increase the priority of this requirement.
	Existing user knowledge and skills	The existence or projection of a large proportion of novice users should increase the priority of this requirement, while a large proportion of experienced users should decrease its priority. However, for I&A, this factor is usually a minor one in either case.
Minimize changes needed to existing infrastructure	Number of connection points	A large number of connection points for the I&A service should increase the priority of this requirement, because each connection point may need an associated change.
	Predicted restructuring of existing infrastructure	If the infrastructure is already scheduled to be changed for other reasons, this requirement will have reduced priority.
Minimize costs of maintenance, management, and overhead	Ability to rely on users properly to protect data or hardware entrusted to them	This requirement should have decreased priority if the users are knowledgeable and trustworthy. However, this assumption has some risk.
	Volatility of user base	The existence or projection of a large turnover rate among users should increase the priority of this requirement.
Protect I&A assets	Cost and risk of authenticator theft	This requirement should have increased priority if the cost and risk of theft of an authenticator, such as a password, is relatively high.

	Cost and risk of I&A service being unavailable	This requirement should have increased priority if the cost and risk of I&A being unavailable is relatively high.
--	--	---

Table 41: Factors affecting relative importance of I&A requirements

## 100.9. Example Resolved

Samuel the systems engineer and Edward the enterprise architect identifies each situation from the museum example above as a separate domain, with a separate set of requirements. The first domain for which they specify I&A requirements is that of the museum employees who access the museum information systems. Table 42 shows the requirements they specified for this domain. The first column contains two sets of information: the generic requirement, followed by the specific requirement for the museum. The second column presents the relevant factor for the requirement in this domain, and the third column discusses the resulting importance of each requirement.

Samuel and Edward determine that the most important I&A requirements for the museum are:

1. Accurately detect imposters
2. Minimize risks to user safety
3. Minimize changes needed to existing infrastructure
4. Protect I&A assets

Generic/Specific Requirement	Factor	Importance for Museum
Accurately detect imposters.  The I&A service shall have a minimum certainty of 0.9999 (shall have no more than 1 false acceptance out of 10000 I&A claims of imposters).	Potential costs of not detecting an imposter	All I&A services for workstations in physical asset display and research work areas must satisfy this requirement. The museum considers this to be extremely important.
Accurately recognize legitimate actors.  The I&A service shall have a maximum false rejection rate of 0.02 (shall deny no more than 1 actor out of 50 I&A claims of entitled actors).	Existence of time-critical functions	This is a moderate level of concern for the museum wing, as they prefer to incur the costs of hampered staff than to falsely assert the identity of an actor.
	User base sensitivity to temporary denial of service	N/A
Minimize mismatch with users.  The I&A service shall support information system users with these characteristics: users are employees interacting with museum information systems, there are local and remote user	User base membership (employees, partners, public, software)	The museum considers this a moderate concern. Only identified and authenticated actors will be able to log on to the information system. No anonymous users.

locations, and the user locations are fixed.		
	User location	All user locations are known.
	User mobility	All I&A services will have fixed locations.
Minimize time and effort to use.  The I&A service shall be easy to use.	Frequency of use	Museum users will not be required to perform multiple log-ons. Training will be provided to ensure workstations are logged off. The museum does not consider this a significant requirement.
	User base characteristics	The museum considers this a moderate concern related to staffing.
Minimize risks to user safety.  The I&A service shall provide adequate safety.	Relevant statutes and enterprise policy	Statutes and policy do mandate this requirement for the museum.
	Vulnerability of enterprise to negative publicity	N/A
Minimize costs of per-user setup.  The I&A service set-up cost per person shall be as small as possible, and in any case shall be less than \$50 per person.	Number of users in general terms	The museum's user base will be restricted to identified and authenticated users. Costs for I&A will be per workstation.
	Volatility of user base	The museum considers this a moderate concern as the rate of staff turnover is not high.
	Existing user knowledge and skills	As noted, the museum intends to provide user training to reduce costs.
Minimize changes needed to existing infrastructure.  The I&A service shall be able to interface with existing components from the parent enterprise.	Existing support contracts	The museum considers this requirement extremely important. The I&A for this museum wing must be able to interface with existing components from the parent enterprise.
	Number of connection points	As above, museum costs will be per workstation, the same as the parent enterprise.
	Predicted restructuring of existing infrastructure	Any future infrastructure changes will occur under the parent enterprise funding profile.
Minimize costs of maintenance, management, and overhead.	Ability to rely on users	User training will be provided.

The I&A service shall be cost effective with respect to maintenance, management, and overhead.		
	Volatility of user base	The museum considers this a moderate concern, as the rate of staff turnover is not high.
Protect I&A assets.  The I&A service shall protect its security assets, such as passwords.	Cost of authenticator theft	The museum considers this a very important requirement, since it could put physical assets at risk.
	Cost of I&A service being unavailable	The museum will need to address multiple back-up plans for loss of I&A service.

Table 42: Resolving requirements for museum information system I&A

## 100.10. Consequences

You may expect the following benefits from applying this pattern:

- The pattern fosters explicit definition of I&A domains and a clear connection of requirements to I&A domains. This increases understanding of the full set of domains that are involved in I&A and understanding of the scope of each set of requirements.
- It facilitates conscious selection of I&A requirements, so that decisions about selecting I&A mechanisms have a clear basis, rather than occurring in a vacuum.
- It promotes explicit analysis of trade-offs that encourages balancing and prioritizing of conflicting requirements. It helps avoid stronger than necessary I&A, which makes it difficult for valid users, and at the same time it helps to avoid weaker than necessary I&A, which makes it easy for imposters to defeat and therefore provide inadequate protection.
- It results in documentation of I&A requirements that communicates to all interested parties, and also provides information for security audits.

The potential liabilities of applying this pattern are:

- It requires an investment of resources to apply the pattern, including time to analyze domains and I&A needs. In some cases, the cost of applying the pattern may exceed its benefits.
- It poses a danger of over-engineering and complexity creep if stakeholders are offered too many options. You can mitigate this by using the requirements only as guidelines for analysis, or by selecting parts of the pattern that give the most help.
- The formal selection process may be too long and costly and produce too much overhead. You can mitigate this in the same way as noted above.
- Specific circumstances might not be covered by generic I&A requirements. You can mitigate this by adding specific requirements and including them in the trade-offs.
- Documentation of requirements implies that they must be maintained as they change over time. You can mitigate this by keeping the requirements in a form that is easy to update, integrated with other system documentation.
- Perception of I&A requirements can differ throughout an organization. This may make it difficult to reach agreement on priorities between requirements. On the other hand,

bringing such disagreements to the surface may be a benefit of the pattern, because then they can be properly discussed and resolved.

## 100.11. Known Uses

The general I&A requirements and the process of specifying I&A requirements described in this pattern represent a consolidation of MITRE Corporation's experience in working with multiple customers over several decades. The approach is generally used informally by those customers, as opposed to being codified or published. However, some discussions of I&A requirements exist. Examples include:

- [1] is a US government policy for electronic authentication of individuals participating in on-line transactions. It discusses some of the non-functional requirements identified in this pattern, such as cost and user burden. [2] provides technical guidance for this policy.
- [3] is an international standard that defines evaluation criteria for information technology security. It includes a class or family of criteria that address the requirements for functions to establish and verify a claimed user identity.
- [4] is a risk-based technique to elicit authentication requirements for electronic transactions. It includes the process of defining context, scope, and nonfunctional I&A requirements.
- [5] describes functional I&A requirements (false positives and false negatives) and discusses I&A domains in terms of requirements scope.

## 100.12. See Also

After applying this solution, the next step is typically to decide what type of I&A to use. If you have decided to use only automated I&A, you can apply AUTOMATED I&A DESIGN ALTERNATIVES.

## 100.13. References

[1] Office of Management and Budget. (2003, December 16). E-Authentication Guidance for Federal Agencies. Memorandum M-04-04. Washington, DC, USA.

<http://www.whitehouse.gov/omb/memoranda/fy04/m04-04.pdf>

[2] National Institute of Standards and Technology (NIST). (2004, September). Electronic Authentication Guideline. NIST Special Publication 800-63. Version 1.0.1.

[http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6\\_3\\_3.pdf](http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63v6_3_3.pdf)

[3] International Organization for Standardization. (1999). Common Criteria for Information Technology Security Evaluation (Version 2.1). CCIMB-99- 031. ISO/IEC JTC 1 adopted CC 2.0 with minor modifications in June 1999 as ISO/IEC 15408, Version 2.1

[4] Software Engineering Institute. (2004, May). e-Authentication Risk and Requirements Assessment: e-RA Tool Activity Guide. Pittsburgh, PA, USA.

<http://www.cio.gov/eauthentication/era.htm>

[5] Firesmith, D. (2003). OPEN Process Framework (OPF) Authentication Requirements. OPEN Process Framework Repository Organization.

## **100.14. Source**

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). Security Patterns: Integrating Security and Systems Engineering (1st ed.). Wiley.