

Software Systems Architecture

Working with Stakeholders Using Viewpoints and Perspectives

Second Edition

**Nick Rozanski
Eoin Woods**

Upper Saddle River, NJ • Boston • Indianapolis • San
Francisco
New York • Toronto • Montreal • London • Munich • Paris •
Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

25. The Security Perspective

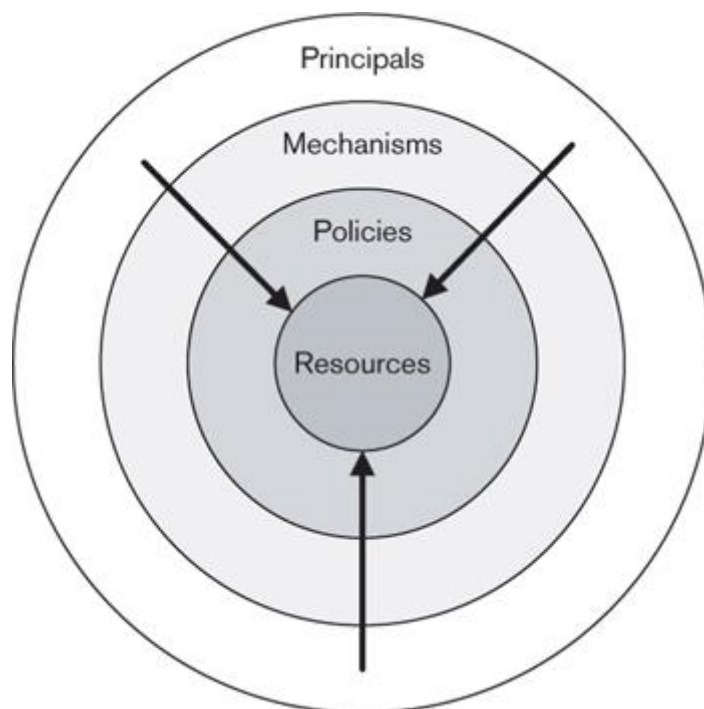
Desired Quality	The ability of the system to reliably control, monitor, and audit who can perform what actions on which resources and the ability to detect and recover from security breaches
Applicability	Any systems with accessible interfaces, with multiple users where the identity of the user is significant, or where access to operations or information needs to be controlled
Concerns	Resources, principals, policies, threats, confidentiality, integrity, availability, accountability, detection and recovery, and security mechanisms
Activities	Identify sensitive resources, define the security policy, identify threats to the system, design the security implementation, and assess the security risks
Architectural Tactics	Apply recognized security principles, authenticate the principals, authorize access, ensure information secrecy, ensure information integrity, ensure accountability, protect availability, integrate security technologies, provide security administration, and use third-party security infrastructure
Problems and Pitfalls	Complex security policies, unproven security technologies, system not designed for failure, lack of administration facilities, technology-driven approach, failure to consider time sources, overreliance on technology, no clear requirements or models, security as an afterthought, ignoring the insider threat, assuming the client is secure, security embedded in the application code, piecemeal security, and ad hoc security technology

Many factors drive today's need for information systems security, including the increasing trend to distribute systems, the use of public networks (particularly the Internet) as part of system infrastructure, the everyday use of interorganizational computing (such as Web services), and the increasing interest the media and the public have shown in privacy and security. All of these factors point to the fact that today your system's stakeholders are likely to be more interested in the security of the system than they would have been only a couple of years ago.

We define *security* as the set of processes and technologies that allow the owners of resources in the system to reliably control who can access which resources. The *who* refers to the people, pieces of software, and so on that form the set of actors in the system who have a security identity; security specialists normally refer to such actors as *principals*. The *resources* are the parts of the system considered sensitive (i.e., those to which access must be controlled) such as subsystems, data elements, and operations. The *access* to the resources refers to the operations that the principals in the system will want to legitimately perform on the resources (e.g., read them, change them, execute them, and so on), and the fact that access must be limited to principals known to the system.

Resources are at the core of the system's security. *Policies* define the legitimate access allowed to them, which is enforced by *security mechanisms*, which are used by the principals of the system to gain access to the resources that they need. We illustrate the interrelationship of these concepts in [Figure 25-1](#) and discuss them in more detail in the Concerns section of the chapter.

Figure 25-1. Principals, Actions, and Resources



The resources, principals, and policies that need to be considered are often very specific to the system. An Internet service provider is likely to have a totally different set of security concerns from those of a military intelligence organization, which will be different again from those of an enterprise implementing an internal information system that allows remote access to its employees. However, in all of these cases, security is still the business of allowing the right levels of access to the right resources to the right people.

It is also important to recognize that security is not a simple process of “being secure” or not. Rather than being a binary state, security is really a process of risk management that balances likely security risks against the costs of guarding against them. Bear this in mind to help you set realistic expectations in the minds of your stakeholders and to make intelligent tradeoffs that address the real security risks your system faces

Applicability to Views

[Table 25-1](#) shows how the Security perspective affects each of the views we discussed in [Part III](#).

Table 25-1. Applicability of the Security Perspective to the Seven Views

View	Applicability
Context	The Context view allows you to clearly identify the system's external connections and consider how they could become system vulnerabilities and how they will need to be protected from malicious use. It may be the case that considering the security of the system will lead you to change the nature of some of these external connections. The Context view may also reveal possible threats to the system from elements in its immediate environment.
Functional	The Functional view allows you to clearly see which of the system's functional elements need to be protected. Conversely, the functional structure of the system may be impacted by the need to implement your security policies.
Information	The Information view also helps you see what needs to be protected—in this case, the sensitive data in the system. Information models are often modified as a result of security design (e.g., partitioning information by sensitivity).
Concurrency	The Concurrency view defines how functional elements are packaged into runtime elements like processes. Security design may indicate the need to isolate different pieces of the system into different runtime elements, and if so, this will affect the system's concurrency structure.
Development	You may identify guidelines or constraints that the software developers will need to be aware of in order to ensure that the security policy is enforced. You need to include these guidelines or constraints in (or reference them from) the Development view.
Deployment	The security design may have a major impact on the system's deployment environment. For example, you may need security-oriented hardware or software, or you may need to change previously assumed deployment arrangements in order to address security risks.
Operational	Enforcing security policy is not just a matter of adding advanced technological features to a system. How the system is operated once it is in production will have a major effect on its security. The Operational view needs to make the security assumptions and responsibilities extremely clear, so that these aspects of the security implementation can be reflected in operational processes.

Concerns

Resources

The reason that we need security in our systems is that they contain valuable information and sensitive operations, and we want to be sure these are accessed or executed only by certain people. The items in the system that we are trying to protect are known in security jargon as *resources*, and computer security is the business of designing processes and mechanisms to provide this protection.

Principals

In the security domain, the entities that our system needs to identify for security purposes are known as *principals*. A principal can be a person, a role, a piece of physical equipment, or another computer system.

Our security system needs to be able to reliably identify principals (that is, *authenticate* them) in order to allow decisions about their legitimate access rights to be made (*authorization* decisions).

Policies

The security policy for a system defines the system's security needs. It defines the controls and guarantees that the system requires for its resources and identifies which principals (or groups of principals) should be granted which types of access to each resource (or type of resource) within the system. A security policy can be considered to be the security specification for a system, as it defines the set of security-related constraints that the system should be able to enforce.

A typical security policy defines the information access policy in terms of the different types of principals the system contains (e.g., clerks, managers, administrators) and, for each type of information (e.g., payroll records, customer details, invoice information), what sort of access each principal group requires (e.g., whether they can view the information, change it, delete it, share it). A security policy also needs to define how the execution of certain sensitive system operations (e.g., dispatch of payments or system shutdown) will be controlled. The policy also should define information integrity constraints that must be enforced, such as the integrity rules and checking required in data stores and protection of messages from unauthorized changes.

Threats

While the policy defines the security constraints the system requires, the threats the system faces are the possible ways the security constraints might be breached by an attacker who wishes to avoid them. Explicitly considering the security threats the system faces allows you to identify security enforcement mechanisms that can then counter these threats. Adding security mechanisms to a system is never free and often adds significantly to the complexity and cost of the system, while often reducing its usability and making it more difficult to operate in production. Therefore, it is important to select mechanisms appropriate to the threats faced, to ensure that all of the chosen mechanisms counter realistic, credible security threats to the system.

The threats that a system faces are related to the environment into which the system is deployed and how the system is built. Certain threats (such as attackers using "phishing" attacks to gain user names and passwords) are much more likely for an Internet-facing system; others (such as users who hold different security roles colluding to bypass a system control) are much more likely for an enterprise system; and threats relating to the introduction of malicious code into the system are more likely where it is easy to introduce new software into the system (such as those that must support extension through plug-ins).

Common threats that most information systems face include password cracking, network attacks that exploit software or configuration vulnerabilities, and denial-of-service attacks as well as nontechnical, social-engineering attacks that try to trick authorized users of the system to perform operations on behalf of the attacker.

Confidentiality

Confidentiality is normally defined as limiting the disclosure of secrets to those who are legitimately allowed to access them. In practice this usually means keeping information in a system from being disclosed to anyone who has not been granted the right to view it. Confidentiality can be achieved using access control provided that the information can be retained within the confines of the system. When the information cannot be reliably retained within the control of the system (e.g., when it needs to be transmitted to another system), cryptography is usually used to maintain confidentiality when the information passes out of the control of the system.

In classical information security theory, confidentiality is usually considered along with integrity and availability, forming the well-known “CIA” trio of concerns. Nowadays, accountability is often added as a fourth general concern for the security engineer.

Integrity

In the context of information security, integrity is a guarantee that information cannot be changed undetectably, and so we can be sure that information has not been tampered with since it was created or changed by an authorized principal. Ensuring information integrity normally involves “signing” data cryptographically, allowing the data and the cryptographic signature to be compared in order to prove that an unauthorized party has not changed the data.

Availability

Availability is often thought of as a purely operational consideration, but in fact it also has a relationship to security. Ensuring that potential attackers of your system cannot block its availability with denial-of-service attacks is an important part of your security design.

Designing a system for availability involves thinking about planned and unplanned outages that can occur due to the nature of the system itself; designing a system to avoid outages as a result of an external attack involves thinking about risks that can occur as a result of the environment in which the system resides (such as who can access the system and how). Availability security is an area that has rapidly gained importance as systems are routinely connected to public networks like the Internet.

Accountability

Accountability is the means of ensuring that every action can be unambiguously traced back to the principal who performed it.

In centralized server-based systems, the notion of auditing is a common mechanism used to ensure accountability. In distributed systems, cryptographic message signing is typically used to prove that a message originated from a particular principal (a form of accountability also known as *nonrepudiation*).

Detection and Recovery

System security is never perfect, and if a system is attractive enough to an attacker, it is almost inevitable that some sort of security breach will occur eventually. This means that an important security concern is the ability of the system to detect security breaches and recover from them. Addressing this concern is unlikely

to be a purely technical matter but is more likely to involve people and processes as well as the technology required to spot a security breach and react to it appropriately.

Security Mechanisms

Security mechanisms are the technologies, configuration settings, and procedures required to enforce the rules established by the security policy and provide the confidentiality, integrity, accountability, and availability guarantees required by the system. Information systems security is a relatively mature field, and many proven security technologies already exist to act as mechanisms in an information system. Examples of commonly used security technologies include user name and password authentication, single-sign-on systems, virtual private networks to secure network links, database access control systems, and SSL/TLS encryption for client/server connections. The broad groups of security mechanisms that are typically found in a modern information system are as follows:

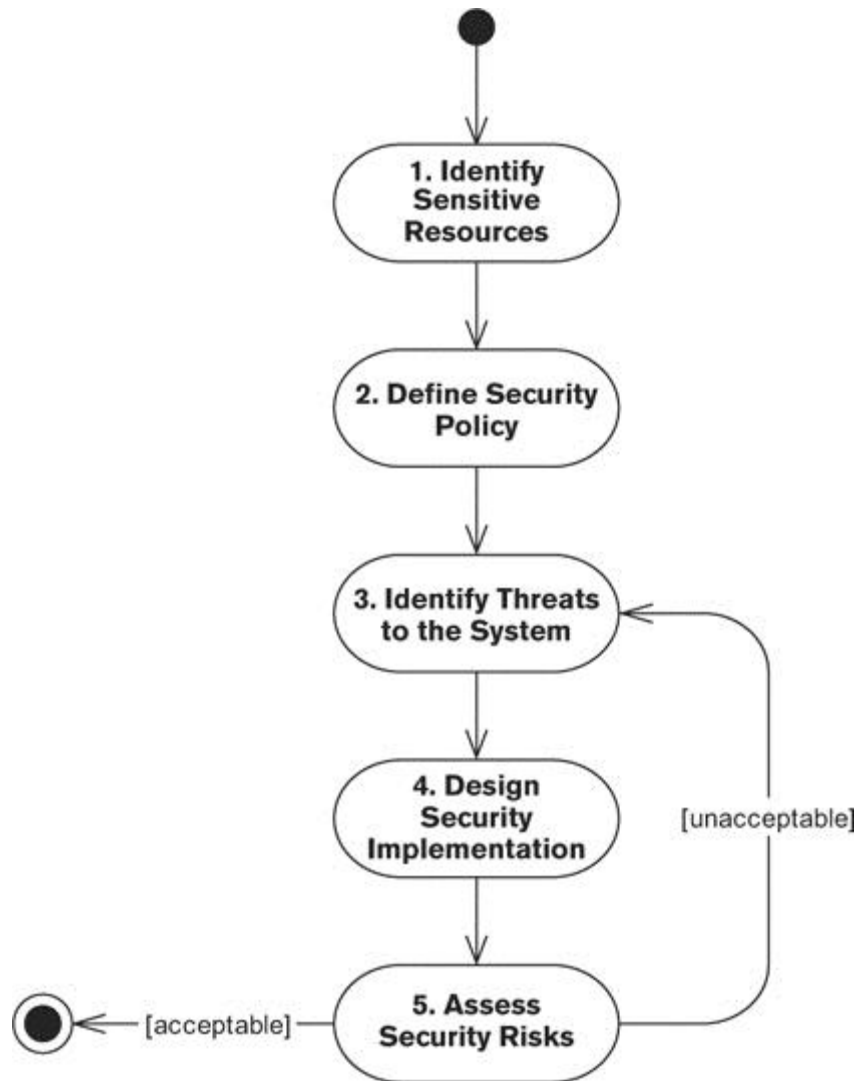
- *Authentication, authorization, and auditing*, which are used to identify principals, grant them rights and privileges, and monitor the use of those rights and privileges, in order to implement effective access control
- *Information privacy and integrity* mechanisms, which use cryptography to prevent information from being accessed by unauthorized parties and allow detection of information tampering
- *Nonrepudiation* mechanisms, such as message signing, which again use cryptography, in this case to prove the provenance of information, such as the identity of a message sender
- *System availability* mechanisms, which aim to keep the system available in the face of malicious or accidental threats (we discuss these technologies primarily in [Chapter 27](#) on the Availability and Resilience perspective)
- *Security monitoring* mechanisms, such as intrusion detection systems and security log monitors, that are used to detect breaches in system security and respond to them

Your challenge as an architect is to select the right set of technologies from the wide array available and to apply them appropriately to the particular system you are building. In particular, it can be difficult to safely combine a number of these different technologies, along with system-wide security configuration settings and a set of secure operational processes, to create a truly secure end-to-end system. Except in the simplest cases, it is usually necessary to involve security specialists in the system design process in order to achieve this. Such specialists will usually also be able to advise you on standards or common practices for your organization or industry.

Activities: Applying the Security Perspective

The activity diagram in [Figure 25-2](#) illustrates a simple process for applying the Security perspective. In this section, we describe the activities in this process.

Figure 25-2. Applying the Security Perspective



Identify Sensitive Resources

Before considering *how* to secure your system, you need to establish *what* needs to be secured. All of the security for your system needs to be driven from this key concern, which you should consider as early in the system's lifecycle as possible.

The information you gather about the security-sensitive resources feeds into all of the other security-related decisions you make.

Notation

A simple text-and-tables approach is normally sufficient, although you may choose to annotate one of the diagrammatic models of the architecture (typically in the Functional or Information views).

Activities

Classify Sensitive Resources. Using the Functional and Information views as primary inputs, along with any security requirements information available, decide what the sensitive resources in the system are—typically, functional operations and data items. For each type of sensitive resource, define the reasons the resource is sensitive, who should be considered its owner, and the type of access controls it requires.



Example

[Table 25-2](#) shows a simple example of documenting some of the sensitive resources that might appear in an e-commerce ordering system.

Table 25-2. Example of Sensitive Resource Identification

Resource	Sensitivity	Owner	Access Control
Customer account records	Personal information of value for identity theft or invasion of privacy	Customer Care Group	No direct data access
Descriptive product catalog entries	Defines what is for sale and its description; if maliciously changed, could harm the business	Stock Management Group	No direct data access
Pricing product catalog entries	Defines pricing for catalog items; if maliciously or accidentally modified, could harm the business or allow fraud	Pricing Team in Stock Management Group	No direct data access
Business operations on customer account records	Needs to be controlled to protect data access and integrity	Customer Care Group	Access to individual record or all records by authenticated principal
Descriptive catalog operations	Needs to be controlled to protect data access and integrity	Stock Management Group	Access to catalog modification operations by authenticated principal
Pricing catalog modification operations	Needs to be controlled to protect data access and integrity	Pricing Team	Access to price modification operations by authenticated principal, with accountability of changes
...

If you're fortunate enough to have comprehensive security requirements defined for you, the sensitive resources may have already been identified. In many cases, though, you will need to do this yourself, in conjunction with interested stakeholders (such as auditors and managers of the groups owning the information). Useful inputs into this exercise include organizational security policies and relevant external regulations, as well as the opinions of interested stakeholders.

When performing this exercise, be sure that everyone (including yourself) understands exactly what is included in each resource type as well as the identity of the owner and the type of security the resource type requires.

Define the Security Policy

Having identified the system's sensitive resources and the threats against them, you should be able to define a security policy (also sometimes called a *trust model*) for your system. The security policy is the basis for the security implementation in the system. This model identifies *who* will be trusted with *what* access to *which* system resources (and any constraints on this access such as limiting it to certain times or days of the week), the *integrity* guarantees required within the system, and the *accountability* required when sensitive resources are accessed.

Ideally, you will have a complete, unambiguous, straightforward security policy defined for you by the system's main stakeholders that you can just check and review before proceeding. Alternatively, if it's like every system we've worked on, you'll have nothing of the sort, in which case you'll need to create it and get agreement on it.

The policy should normally be defined in terms of groups of resources and principals (often based on organizational unit and role) rather than enumerating lots of specific cases. Also remember that this is a policy, not a design, so it needs to define what access will be provided to whom rather than defining how this will be achieved.

Work to make the security policy as simple and general as you can, with as few special cases as possible. A straightforward policy model allows the simplest possible implementation of enforcement mechanisms and makes it more likely that the model will be enforced correctly.

The other vital quality of a security policy model is precision. This model is an important deliverable for your system, and it should be approved by all the important stakeholders. If the model isn't precise, each group of stakeholders will interpret it differently, according to their own interests, assumptions, and desires. This is bound to lead to problems when implementing the policy.

Notation

A security policy is normally defined by using a simple text-and-tables approach to create a structured document. Tables are particularly valuable when defining groups of resources and principals and showing the types of access allowed.

Activities

Identify the Principal Classes. In order to make defining the security policy manageable, start by grouping your principals into classes that you can treat as groups for security policy purposes. Partition the principals into sets based on their role and the types of access they require to different types of sensitive resources.

Identify the Resource Classes. Partition the system’s sensitive resource types into groups that you can treat uniformly for access control purposes. (The resource types are derived from the resources, as described earlier.)

Identify the Access Control Sets. For each resource class, define the operations that can be performed on members of that class and the principal classes that should be allowed to access each operation on the class.

Identify the Sensitive System Operations. Consider any system-level operations that are independent of the system’s managed resources (such as administrative operations), and define which principal classes should be allowed to access these operations.

Identify the Integrity Requirements. Consider any situations in the system where information is or could be changed or sensitive operations performed, and identify the set of integrity guarantees required for them (such as auditing or “four eyes” approval).



Example

[Table 25–3](#) shows part of the result of defining the security policy for an e-commerce system.

Table 25–3. Example of an Access Control Policy

	User Account Records	Product Catalog Records	Pricing Records	User Account Operations	Product Catalog Operations	Price Change Operations
Data administrator	Full with audit	Full with audit	Full with audit	All with approval and audit	All with audit	All with approval from a product price administrator
Catalog clerk	None	None	None	All	Read-only operations	None
Catalog manager	None	None	None	Read-only operations with audit	All	All with audit
Product price administrator	None	None	None	None	Read-only operations	All with audit
Customer care clerk	None	None	None	All with audit	Read-only operations	None
Registered customer	None	None	None	All on own record	Read-only operations	None
Unknown Web-site user	None	None	None	None	Read-only operations	None

Seven different classes of principals have been identified (including unauthenticated Web-site users) and six classes of sensitive resources. Only data administrators are allowed access to data directly; all other users can only execute business operations. Data administrators can also execute business operations (on the basis that they can simulate their occurrence anyway by changing data), but all of their activity is audited and some actions need approval. The other classes of principals have been allocated just enough privileges in order to perform their roles in the system.

Identify Threats to the System

In a perfect world, we could now stop and simply publish the list of sensitive resources and the security policy, and everyone would voluntarily follow the policy. However, the reason we need security is that we can't trust everyone to do this, so we need to identify the possible threats to the security policy. Identifying threats provides a clear definition of what needs to be protected and what it needs to be protected from. This makes it clear which threats you are aware of and can try to guard against and also (implicitly) the threats you have not considered.

The result of this process is termed a *threat model*, which builds on the initial list of sensitive resources and should provide a thorough analysis of the threats to which you think the system is subject, the impact of the threat being realized, and the likelihood of the threat occurring.

To create a threat model, ask a number of key questions about your proposed system.

- Who is likely to try to circumvent the security policy?
- What is the attacker's motivation for attacking the system?
- How will an attacker try to circumvent the security policy?
- What are the attacker's main characteristics (such as sophistication, commitment, resources, and so on)?
- What are the consequences of the policy being breached in this way?

When considering the threats your system faces, be sure to consider threats from inside the organization as well as the more obvious "bad guys" who are outside it. Many large organizations have suffered attacks from current or former employees. Employees are often in a trusted position and have a lot of knowledge about the organization's systems, which makes it easier for them than for outsiders to circumvent information systems controls.

It is also important to consider how the environment into which your system is deployed affects the threats it faces. If your system is hosted outside your organization (such as at a hosting provider or in a cloud computing environment), you need to consider the threats it faces from the insiders who run the infrastructure on which it is hosted, as well as the insiders in your own organization. On the other hand, such external providers may be able to provide a level of security technology, expertise, monitoring, and support that is not available within your organization.

Explicitly identifying threats allows specialists from outside your project to provide assistance by reviewing the model and advising you about potential threats that have not been considered at all or threats that may have been incorrectly characterized in the model. Be sure to have it widely reviewed so that you can be confident you have considered all likely threats. The threat model also allows you to systematically consider the set of security facilities your system needs.

Notation

The threat model is normally presented by using a text-and-tables approach to create a structured document. A graphical alternative is to use an attack tree, which is a tree structure used to categorize and illustrate the threats a system faces and the likely probability of each occurring.



Example

An attack tree is a useful approach to representing the threat model for your system. The technique is based on the Fault Trees technique used in safety-critical systems design to analyze possible failure modes. In the security domain, you can take a similar approach to represent possible attacks on your system.

An attack tree represents the possible attacks your system may face in order for an attacker to achieve a particular goal. The root of the tree is the goal the attacker is trying to achieve, and the branches of the tree classify the different types of attacks the intruder could attempt in order to achieve the goal.

Attack trees can be represented graphically (as a tree structure with nodes and links) or textually as a “dotted decimal” hierarchy (like nested subheadings in a technical document). The latter form is often more practical because attack trees can get very large as all possible attacks are considered, which can make the graphical form difficult to draw and comprehend.

Here is a possible attack tree for the goal of extracting customer credit card details from an e-commerce Web site.

Goal: Obtain customer credit card details.

1. Extract details from the system database.

1.1. Access the database directly.

1.1.1. Crack/guess database passwords.

1.1.2. Crack/guess operating system passwords that allow database security to be bypassed.

1.1.3. Exploit a known vulnerability in the database software.

1.2. Access the details via a member of the database administration staff.

1.2.1. Bribe a database administrator (DBA).

1.2.2. Conduct social engineering by phone/e-mail to trick the DBA into revealing details.

2. Extract details from the Web interface.

- 2.1. Set up a dummy Web site and e-mail users the URL to trick them into entering credit card details.
- 2.2. Crack/guess passwords for user accounts and extract details from the user Web interface.
- 2.3. Send users a Trojan horse program by e-mail to record keystrokes/intercept Web traffic.
- 2.4. Attack the domain name server to hijack the domain name and use the dummy site attack from 2.1.
- 2.5. Attack the site server software directly to try to find loopholes in its security or configuration or to exploit a known vulnerability in the software.

3. Find details outside the system.

- 3.1. Conduct social engineering by phone/e-mail to get customer services staff to reveal card details.
- 3.2. Direct a social-engineering attack on users by using public details from the site to make contact (see also 2.1).

An attack tree should be created for each of the possible goals that an attacker may have for breaching your system's security. Once you have an attack tree, you can analyze each threat it contains to establish whether the system's security neutralizes the threat.

Activities

Identify the Threats. Consider the security threats your system faces from the perspective of the sensitive resources within it, the possible access to these resources that potential attackers might wish to gain, the main characteristics of the potential attackers, and the types of attacks they are likely to carry out.

Characterize the Threats. Characterize each threat in terms of the resources that would be compromised if the attack were successful, the result of this compromise, and the likelihood of the attack occurring.

Design the Security Implementation

Once you understand the sensitive resources and threats, you can consider the technical security design for the system. The goal of this step is to design a system-wide security infrastructure that can enforce the system's security policy in the face of the risks identified in the threat model. In this step, you consider using specific security technologies such as single-sign-on systems, network firewalls, SSL communication link security, cryptographic technology, policy management systems, and so on.

This design process results in a number of design decisions you should incorporate in the architecture. These decisions affect a number of architectural structures, including those likely to be described in the Functional, Information, Deployment, and Operational views.

Notation

The outputs of this step are a set of design decisions to be reflected in the architectural views, so the notation used for the technical security design depends on the notation used in each view. You may also

produce some form of overall security design model, which is typically captured by using a software design notation such as UML (similar to its use in the Functional viewpoint).



Example

Based on the attack tree shown in the previous example, some of the security measures you might consider for the e-commerce system include the following:

- Isolating the database machines from the public network by using network firewall technology
- Isolating the security-sensitive parts of your system from the public network by using network firewall technology
- Analyzing the paths into your system to check them for possible vulnerabilities
- Arranging penetration testing to see if experts can find ways into your system
- Identifying an intrusion detection strategy that would allow security breaches to be recognized
- Training administration and customer service staff (in fact, probably all staff) to avoid social-engineering attacks and to abide by strict privacy protection procedures for customer information
- Designing your site so that a minimal amount of user information (ideally, none) is publicly viewable
- Designing your site so that sensitive information (e.g., credit card numbers) is never shown in full (e.g., display just the last four digits to allow legitimate users to identify their cards in lists)
- Routinely applying security-related software updates to all third-party software used in the system
- Reviewing the system's code for security vulnerabilities using analysis tools and expert inspection
- Constantly reminding users of security precautions they should take (e.g., not revealing passwords to anyone, including your staff; checking URLs before entering information; and so on)

It is interesting to note how much of a typical security implementation does not involve security technology directly but is about making sure that people act in a secure manner.

Activities

Design a Way to Mitigate the Threats. For each of the previously identified threats, design a security mechanism to address the threat. This may include modifying existing architectural decisions, applying one or more security technologies, and designing procedures and processes for the system's operation and use.

Design a Detection and Recovery Approach. Bearing in mind that the security mechanisms you identify are unlikely to be foolproof, design a system-wide approach to detecting violations of the system security policy and recovering from them. This will typically include technical intrusion detection solutions, internal system checks and balances to reveal unexpected inconsistencies, and a set of processes for regularly

checking the system for intrusions and reacting to any discovered. Good solutions for intrusion detection are available as proven commercial and open source software packages; however, deciding how to use the packages and design the processes around them are still system-specific activities.

Assess the Technology. One way to address a threat is to use a piece of security technology to provide a security mechanism. This activity involves assessing which candidate security technologies are suitable for addressing a particular threat in a particular context. This includes generic assessment (e.g., checking reliability, ease of use, and so on) as well as context-specific assessment (e.g., checking that the proposed technology can be operated by the system’s administrators and is efficient enough to meet the system’s performance and scalability goals).

Integrate the Technology. Once you have decided which security technologies to apply to your system, the other important activity is deciding how you will integrate them with the primary system structure and with other security technologies. It is important to spend some time designing the integration approach carefully, to avoid possible security loopholes creeping in due to the unexpected side effects of an ad hoc integration approach.


Assess the Security Risks

No security system is perfect. The process of implementing system security is a balancing act: You balance the risks you believe you face against the cost of implementing mitigations for those risks and the costs you may face if the risks occur.

Having designed a security infrastructure for your system, you now need to reevaluate the risks to consider whether your proposed security infrastructure has achieved an acceptable cost/risk balance. If so, your Security perspective is complete. If not, you need to return to a consideration of the threat model and refine the security infrastructure in order to achieve an acceptable cost/risk balance.

Notation

The risk assessment delivers a record of the set of risks, the estimated likelihood of each risk occurring given the design of the system, and the notional cost (i.e., the estimated cost adjusted for the probability of occurrence) that each risk implies. This information is usually best presented in a simple tabular form.



Example

[Table 25-4](#) shows an example, using a tabular presentation style, for a couple of the risks facing our e-commerce system.

Table 25-4. Example of Risk Assessment

Risk	Estimated Cost	Estimated Likelihood	Notional Cost
Attacker gains direct database access	\$8,000,000	0.2%	\$16,000
Web-site flaw allows free orders to be placed and fulfilled	\$800,000	4.0%	\$32,000
Social-engineering attack on a customer service representative results in hijacking of customer accounts	\$4,000,000	1.5%	\$60,000
...

A risk presentation in this form allows us to focus on the risks with the highest notional costs (i.e., those that have the highest risk of loss and greatest likelihood of occurrence).

Activities

Assess the Risks. The single activity in this step is a process of risk assessment. For each risk in your threat model, reevaluate its likelihood of occurrence and its likely impact given the planned security infrastructure, and compare this against the security needs you established earlier in the process. Assess whether this is an acceptable level of security risk for your particular system's situation. It is worth noting that this can be a lengthy process because it can be difficult and time-consuming to produce reliable estimates for the impact and likelihood of the occurrence of each risk.

Architectural Tactics

Apply Recognized Security Principles

The field of computer security is relatively mature for a rapidly evolving, high-technology discipline. There is an established body of knowledge, developed and applied by an identifiable professional security community with its own principles, standards, norms, and culture.

This community of researchers and practitioners has established a number of widely accepted and commonly applied principles that are considered important to establishing security within a system. Some of the more important principles include the following.

- *Grant the least amount of privilege possible:* Always grant security principals the smallest set of privileges they require in order to perform their tasks. Consider varying the set of privileges a principal has over time if certain sensitive tasks are executed only intermittently.
- *Secure the weakest link:* The security of a system is only as strong as its weakest element, so understanding the weakest link in your security is an important step toward understanding how secure your system really is. The weakest link could be technological (an unsecured network link), procedural (allowing easy access to a data center), or human (people who write down their passwords). Identify and secure the weakest links in your system's security until you achieve an acceptable level of security risk.

- *Defend in depth*: If you examine physical security systems, you'll find that they rarely rely on just one security measure. Just as medieval castles had moats, drawbridges, and strong walls, banks have alarms, vaults, security guards, surveillance systems, and multiple locks on important doors. These are examples of the principle of defense in depth, where a series of defenses provides a greater level of security than a single one could. Defense in depth is particularly relevant to computer systems, given that many of the security technologies we use may themselves have hidden flaws and that we are also susceptible to human and procedural failures. Rather than relying on one security measure to counter each threat to your system, consider possibilities for layering defenses to provide greater protection.
- *Separate and compartmentalize*: Attempt to clearly separate different responsibilities so that authority for each can be assigned to different principals if required, and compartmentalize responsibilities for different parts of the system so that they can be controlled individually. This makes it easier to control access securely and means that a successful attack on one part of the system does not immediately compromise it entirely. Good examples are separating the "security override" privilege from the "alter audit trail" privilege (with the audit trail being used to record use of the "security override" privilege) and implementing separate security configuration and mechanisms for each major subdivision of a system.
- *Keep security designs simple*: Security engineers often say that complexity is the enemy of security. Complexity in a system is difficult to deal with and makes it very difficult to analyze the system to assess its security. This makes it difficult to know whether the system will be secure or not, and it increases the likelihood of vulnerabilities creeping into the system. Systems with stringent security requirements need to be simple enough to make it possible for them to be secured and verified.
- *Don't rely on obscurity*: Some years ago, it was common for the details of secure systems to be kept secret in an attempt to make them more difficult to attack. An example of this was hiding security secrets, such as cryptographic keys, in obfuscated (and undocumented) computer code for which the source was kept secret. The weaknesses in this approach are that it assumes the attackers aren't smart enough to work past the obscurity and that it may prevent external experts from assessing the real level of security provided. The conventional wisdom in today's security community is total disclosure, where the system is designed assuming that potential attackers know it as well as its implementers do. This principle makes the system's security less reliant on hiding secrets, which is extremely difficult to do successfully.
- *Use secure defaults*: Many systems we have encountered include inherently insecure default settings and behaviors, including empty default passwords, permissive default access control lists, network ports open by default, and so on. Such behavior is very likely to result in real security threats, particularly when exhibited by packaged software products, which are often installed by users who are not familiar with them. Ensuring secure default behavior makes a real contribution to practical systems security.

- *Fail securely*: Another common problem found in many real systems is insecure failure mode behavior, where the system is reasonably secure during normal operation but becomes insecure when things go wrong. Examples include error logs that record sensitive information, audit trails that suspend auditing if the audit logs run out of space, systems that drop back to an insecure mode if security negotiation fails, unprotected recovery consoles that appear after crashes, and so on. System failures are inevitable occasionally, so make sure that if your system fails, it does so securely.
- *Assume external entities are untrusted*: Within your system you should be able to exert a great deal of control over the security environment and the principals within it. This is not the case for external entities who access the system. Ensure that all external entities are totally untrusted until proven otherwise to avoid accidental security breaches in unexpected cases.
- *Audit sensitive events*: Most systems include a number of key security-related events that, if abused, could compromise the security system. Common examples include resetting passwords, assigning powerful roles, and manipulating audit trails. These sensitive events need to be securely audited so that their use can be monitored. This principle often places a number of constraints on your deployment structure to ensure that a reliable audit trail can be implemented.

Make sure that you understand these commonly accepted principles and apply them to your system's security. In the Further Reading section at the end of this chapter, we recommend a number of background references that can provide much more detail on these topics.

Authenticate the Principals

Authentication is the reliable identification of each of the principals who can use the system. These principals could be people, computers, pieces of software, or anything else in the system that has a security identity. Nearly all system security depends on being able to identify the various principals in the system reliably.

Implementing authentication involves reliably binding a unique name of some sort to each principal who can use the system. A wide variety of authentication technologies exist to allow you to identify the principals in your system, including user names with passwords, public/private key systems (such as X.509 digital certificates), and hardware token technologies (such as smart cards). Off-the-shelf products known as single-sign-on systems allow you to delegate the entire system-wide authentication process to them (which is particularly useful in mixed technology environments such as those with both UNIX and Windows authentication). Each of these technologies has its place, but each tends to be suited to different environments and security needs, so you need to select the right ones for your system.

The key decision to be made is how every principal in your system is uniquely identified, via a mechanism that is secure enough to address the particular risks your system faces. You should bear in mind that you may need to use different principal authentication technologies for different principal types, and in some cases, principals may need to be identified by more than one authentication technology (e.g., users in a corporation may have separate digital certificates, operating system logins, database logins, and application logins). When multiple logins are necessary, we strongly recommend some form of single-sign-on technology

as a unifying layer on top of the different underlying systems. People simply can't remember more than a couple of passwords, and forcing them to remember more will result in passwords written on notes stuck to terminals!

What is critical at the architectural level is that every principal can be reliably identified when required and that the system you use is simple and usable enough that the system's users will not try to work around it.

Authorize Access

Once you have identified the principals, authorization involves restricting and enforcing what those principals are permitted to do within the system. Conceptually, you define lists of principals and their permitted actions for each controlled resource in the system. This usually involves assigning principals to roles and groups so you can treat them as homogeneous populations. Then you can define permitted actions on resources for the roles and groups in the system.

Most enterprise software technologies already include some form of access control that allows you to limit access to resources based on user identity and credentials (such as group membership). In addition, software developers can code explicit security checks into their software when they implement the system.

In many systems, your key problem as the architect is how to create a coherent access control system from this patchwork of separate, often incompatible authentication and authorization technologies. For system security to be manageable and effective, the system architecture needs to include a clearly defined approach to enforcing principal authorizations, particularly if multiple candidate authorization systems are available in the system's environment.

One solution to this problem is the use of purpose-built access control products. These allow access to many sorts of resources to be controlled through a single product that interfaces to a variety of authentication systems. Another common approach is to standardize all authorization to be the checking of roles against a central enterprise directory (such as an LDAP directory). A simpler approach still would be a simple homegrown access control manager in your system architecture, which may be sufficient for small systems.

Ensure Information Secrecy

Secrecy is the means of ensuring that only owners of information and those with whom they choose to share it can read that information.

Secrecy can be partially achieved by using authorization appropriately, to prevent access to stored information that a principal is not authorized to access. However, access control is rarely sufficient, and it usually needs to be combined with other techniques in order to achieve the secrecy required.

Traditional information systems were highly centralized, with huge information vaults held on central computers. The information was tightly controlled by access control systems (typically as part of database management systems) and was accessible only via custom applications running on terminals directly connected to the central computer.

In contrast, today's information systems are highly distributed, with many computers communicating in order to implement the functions of the system. This means that information may become much less secure than with a traditional centralized system—particularly if public networks are used for communication and the different parts of the system are within different organizations. The ability of most end users to easily access the Internet and its many storage services and the widespread use of very large-capacity personal storage devices (such as USB sticks) make the situation more complicated still.

One of the implications of this shift is that sensitive information needs to be protected once it moves outside the authorization controls of the database servers where it resides. Ensuring secrecy of information outside databases normally involves the use of cryptography, to encrypt information in such a way that only principals who have access to a particular key can decrypt it and read it. In reality, the inherent complexity of cryptography means that it is often implemented by securing network links through the use of communications protocols that include cryptographic protection, like SSL/TLS, rather than by encrypting individual pieces of sensitive information. However, this protects the information only during transmission and leaves it easily accessible at both ends of the network link.

From the architect's point of view, the snag with this neat picture is that extensive use of cryptography in your system is not without cost. It will add significant complexity to many aspects of the system (cryptographic key management alone is a complex problem), and it also tends to have high computational costs, burning up precious computing resources that could be dedicated to useful functional processing. It can be difficult to apply cryptography correctly so that the system is actually secure. It is significantly more complex than just using server-certificate-based SSL to identify your servers because this leaves your system open to any number of alternative attacks (such as intercepting information elsewhere in the system once it has been decrypted).

In general, use your threat model to identify where information needs to be protected, and use the minimum amount of cryptography that leaves your system acceptably secure.

Ensure Information Integrity

Integrity is the name given to the concern of ensuring that information is protected from unauthorized change (particularly during message transmission).

Implementing integrity in most information systems also involves the application of cryptography. We would like the recipients of messages sent over a network to be able to satisfy themselves that they have received each message unchanged and that no one has tampered with it. Ensuring integrity of information usually involves the application of cryptographic hash functions. Hash functions are cryptographic operations that use a secret key to compute a large numeric value for a block of information. They allow the recipient of that block to ensure that the information has not been changed since the hash value was computed. However, useful as this is, the same caveats mentioned for secrecy also apply to integrity, in particular that it adds both complexity and runtime overhead to the system and so needs to be used judiciously.

Ensure Accountability

Many systems require some or all of their users to be accountable for their actions. In some systems (such as certain financial and medical systems), legal requirements mandate user accountability for key operations. There tend to be two distinct forms of accountability required in information systems: *auditing* and *nonrepudiation* of messages.

Auditing is usually used in situations where secure central servers execute the system's primary operations. These servers can record logs of operations performed by the system's users, which can later be used to establish how a particular situation occurred.

In distributed systems without centralized servers, effective auditing is more difficult (due to the lack of a secure store for the audit trail). The analogous concept in these systems is nonrepudiation, the ability to definitively identify a message's creator in a way that she cannot plausibly deny. A common solution to this problem is to use digital signatures on messages that can be generated only by someone with access to the corresponding digital certificate's private key.

Both of these approaches come with their own costs. Auditing has a runtime performance penalty associated with it, as well as management and storage overhead for the records created. Digital signatures are computationally expensive to create and verify, and they can be complex to administer because they require all of the principals in the system to have a unique digital certificate. We are also ignoring other important complexities such as certificate management and revocation and the difficult problem of ensuring that technically sophisticated users do not compromise other users' digital certificates.

Once again, your role as an architect is to balance the cost of these accountability mechanisms against the risks you think your system faces, applying these techniques where the threat model indicates they are really needed.

Protect Availability

When thinking about system availability, it is natural to focus on hardware reliability, software replication, failover, and so on. Such approaches are an important piece of the availability puzzle, as we discuss in [Chapter 27](#) on the Availability and Resilience perspective. However, there is an easily overlooked security aspect related to availability: protecting your system from hostile attacks that aim to reduce its availability.

Such attacks, known as *denial-of-service (DoS) attacks*, have been the subject of increasing interest as systems have started to expose their network interfaces to public networks like the Internet and a number of high-profile DoS attacks have taken place on well-known sites such as the Internet's DNS backbone servers. DoS attacks can range from something as simple as forcing a user to be locked out by repeatedly using an incorrect password with her login,¹ to sophisticated distributed network attacks (so-called DDoS attacks) that use custom software running on many Internet-connected computers to overload systems. DoS attacks can also be physical attacks such as attempting to interrupt a system's power supply.

¹. This doesn't sound like a terribly serious attack until you consider time-sensitive systems such as online auctions. This attack has allegedly been used on a number of occasions at Internet auction sites to

prevent users from bidding on (and so raising the prices of) items that the attacker wishes to buy. Other systems can also be vulnerable to similar DoS attacks—for example, a battlefield system isn't much use if an attacker can lock out all of its users simply by bombarding it with credible user names and random passwords.

Work to understand the possible DoS threats your system faces and the impact of any of the threats occurring. This will probably require working with security specialists who understand this area in detail. Once you understand the threats, their costs, their likelihood, and the potential approaches to protect against them, you need to decide the right level of protection for your system by balancing these factors. Be aware, however, that protecting your system from network DoS attacks can be tremendously difficult, and for most systems you may simply have to accept some level of risk of such an attack occurring.

Integrate Security Technologies

Almost inevitably, when you design the security implementation, you will need a number of different technologies in order to fulfill all of the requirements. Similarly, security normally needs to be implemented across a number of distinct parts of your system. Given this situation, you need to make an early design decision about how to achieve end-to-end security in the system. Implementing a large number of unrelated security technologies is highly undesirable because it introduces complexity and the likelihood of creating security vulnerabilities at the boundaries of each technology.

Part of your role in the security design of the system is to ensure that security is implemented consistently and that the different pieces of the technology are put together to form a complete, integrated security system. This will be particularly important if you're using security consultants from a number of sources (such as different product vendors) who will focus on their own parts of the security puzzle.

Provide Security Administration

Administration is a weakness of many existing security systems, particularly when complex policies need to be enforced or when there is a need to cope with the sort of large user populations found in today's global organizations. This problem can become even more awkward when you need to combine a number of security technologies in order to implement the security policy.

As part of architecture definition, you need to ensure that the planned security implementation can be administered effectively (this will form part of the Operational view). Without good administration, security policy is likely to be ignored simply to "get the job done." Complex security administration facilities can also lead to security loopholes caused by administrators who are unaware of the full ramifications of their actions.

Remember to involve key stakeholders when considering security administration. Your system administrators will be able to provide a lot of feedback about the acceptability of the administration facilities available—testers too often have a perspective on this because they need to reproduce large security configurations automatically to allow reliable, repeatable testing of the security facilities.

Use Third-Party Security Infrastructure

Many of today's infrastructure technologies (such as J2EE and .NET servers, enterprise directories, and e-mail servers) provide a number of standard security functions. Also, specific security infrastructure technologies (such as enterprise access control products) are available that can provide security services to applications. These technologies offer an alternative to the traditional approach to implementing application security—namely, embedding policy enforcement in the application code.

In general, using external security infrastructure is simpler than coding it directly; after all, someone else has done a lot of the work already. This approach is often much more flexible, too. Rather than needing to change program code when the security policy changes, it is often possible to change the configuration of the infrastructure element—a process that can be achieved by the administrators without involving software developers and rereleasing software. Most important, a security infrastructure product is likely to provide more reliable security than application-specific code because it is more widely used and has been written by software developers with specialized interest and training in system security.

We strongly recommend adopting a security approach of pushing as much of your policy enforcement into the underlying infrastructure of your system as possible. We suggest this for three reasons: simplicity, flexibility, and reliability. Security infrastructure products hide much of the complexity of integrating different security systems. They should also provide some degree of abstraction over individual mechanisms, allowing the mechanisms to be changed more easily. Finally, widely deployed infrastructure products have been widely tested, analyzed, and understood, increasing their reliability when compared with custom solutions.

Problems and Pitfalls

Complex Security Policies

Security policies have a habit of starting out simple and well defined but becoming complex, full of special cases, and poorly understood as they are reviewed by more and more stakeholders. Implementing system security is difficult during the best of times. If the security policy is not a simple and regular rule set, this job becomes much more difficult during both development and system operation, and the likelihood of flaws in the security implementation is much higher.

Risk Reduction

- Make sure that the security policy is as simple as it can be—make functional, performance, and other tradeoffs if appropriate in order to achieve this goal.
- If policies appear to be very complex, consider partitioning the security-sensitive resources or the principals in different ways to make the policy simpler.

Unproven Security Technologies

There are many security technologies available for implementing your security infrastructure. They range from simple, very well-understood technologies such as user names and passwords to large-scale, sophisticated systems in their own right such as many public/private key cryptography implementations.

However, whatever technology you use, your system's security is only as strong as its weakest link. A single weak element in your security infrastructure has the potential to make the rest of the infrastructure irrelevant. The weakness in an element can come from an inherent design or implementation flaw in the underlying product, a mistake made in its application, or indeed a mistake made in operating the technology.

Risk Reduction

- Err on the side of caution when selecting your security technology—such technology needs to be well understood and proven in operation before you can consider it a sound choice for a generic piece of infrastructure.
- If you don't have practical experience with a particular technology, find someone who does have such experience to guide you. (And don't assume that everything the vendor says is true!)

System Not Designed for Failure

As any architect knows, things go wrong in the process of designing, building, and operating information systems. Therefore, system elements are often created to fail in certain ways in order to minimize the impact of a particular failure. This is often a sound approach. It allows the system to continue being used (possibly with reduced function) while one or more pieces of it are being fixed. However, when considering security, this is not always the best approach.

To illustrate this, consider the failure of a security element that controls access to a sensitive resource. If the element is unexpectedly unavailable, the high-availability approach to design might suggest that the default behavior in such cases is to simply allow access until the security element is fixed. However, this obviously isn't the right thing to do from a security perspective. Similar but much more subtle examples of this problem can happen when unexpected errors occur and security has not been considered while designing the relevant error-handling code.

Risk Reduction

- Design your security infrastructure to cope with failure in a safe way from the beginning so that unexpected element failures do not open security loopholes in the system.
- Ensure that your security infrastructure is configured to fail securely if unexpected situations occur.
- When reviewing your architecture in failure scenarios, be sure to check what impact the failures have on the system's security.

Lack of Administration Facilities

An important but often neglected part of designing security infrastructure is to ensure that it can be effectively administered when the system is in production. Your testers may also have administration needs to ensure that they can reproduce large-scale test scenarios effectively.

Two common problems in this area include administration tools that can be used only for simple test cases (rather than for the thousands of users who can be found in real systems) and the use of a patchwork of

administration tools that need to be carefully combined in order to manage the system. Such a situation will likely result in loopholes in security policy enforcement as administrators struggle to use inadequate tools to control access.

Risk Reduction

- Review your architecture to ensure that its administration facilities are adequate for the expected size of the user population and the complexity of the security policy.
- Make sure that the administration facilities you are planning to provide are acceptable to the administrators and operators who will have to use them.
- If possible, make sure that administrative security operations can be performed by using one task-driven interface, rather than requiring the use of a number of tools to perform a single task. If many steps are required, the likelihood of omission or error is much higher.

Technology-Driven Approach

A problem we have seen with the security design of some systems occurs when the available technology drives the security design process. You can tell that you have this sort of problem when you inquire whether the designers have considered security yet and you get the reply, "Oh, yes—we're secure; we use SSL." In such situations, although security technology is deployed, you often have no idea whether it addresses the system's security needs.

Risk Reduction

- Drive your security design by the resources that need to be protected, the security model that needs to be implemented, and the security threats that the system faces.
- Avoid designing your security architecture around specific pieces of security technology. Don't incorporate security technology that is not justified by the security needs of your system.

Failure to Consider Time Sources

A number of security mechanisms rely on checking the passage of time (e.g., product license timeouts and password expirations). These mechanisms assume that a reliable source of time is readily available. Although the specifics of the mechanisms and the time source they need differ, they all share the characteristic that if the time source is compromised, the mechanism is ineffective.

Risk Reduction

- Identify the security mechanisms that require accurate time and the characteristics of the time they need (such as accuracy, absolute or relative, universal or time zone, and so on).
- Incorporate enough secure time sources in your system to meet the needs of your security mechanisms.
- Make sure you understand what will happen if a secure time source is unavailable or is compromised.

- Protect secure time sources against likely threats (using mechanisms such as calling back to secure servers, using external time servers or dedicated hardware devices, using operating system access controls to protect the time source, and so on).

Overreliance on Technology

It is often said by security luminaries like Bruce Schneier and others that security is a process, not a product. Yet you can often find system designers placing great reliance on particular security products to keep their systems secure. Although you certainly do need to use good third-party security products to secure your system, you need to use them intelligently as part of an overall security design that encompasses all of the different aspects of your system.

At a security conference some time ago, an executive of a security product company pointed out that “there is no way you can buy anything, subscribe to anything, and say you are 100% secure.” This is sound advice, coming from someone actually in the IT security industry.

Risk Reduction

- Use your threat model to drive your security design. Addressing these tangible threats will help you design a system that is secure, rather than a system that just uses security technology.
- Design a sound set of operational procedures to avoid human error exposing the system to security threats.

No Clear Requirements or Models

It is common for systems to have no clear, well-defined security requirements and no formal security models at all. The problem with these systems is that you simply don’t know if they are secure because you don’t know what “secure” means. Security is an area where it usually isn’t possible to be confident that you have met the needs of the stakeholders without firm requirements and formal models. Security can’t be seen, and most stakeholders won’t test it. You’ll find out that you have problems only when the security is breached—at which point you can be pretty certain that the stakeholders won’t think their needs are being met.

Risk Reduction

- Drive the security design process by using threat and security policy models. Developing these models will help define requirements because they focus the stakeholders’ minds on what is valuable, what is likely to be attacked, and what the impact of such attacks would be.
- Use plenty of concrete examples when discussing security with stakeholders. Security is an abstract area that requires some lateral and imaginative thinking in order to be effective. Examples will help your stakeholders think clearly about what they need.
- When identifying your architectural scenarios, consider those related to security as well as to functionality, performance, evolution, and so on. Thinking through security-related scenarios can help identify and clarify important security requirements.

Security as an Afterthought

Because stakeholders often don't think about security requirements explicitly, these requirements don't always get mentioned in the initial requirements analysis. This can lead to the problem where security has to be added to the system at some point during (or even after) development. At best, this is likely to be an expensive and painful process involving a lot of rework. At worst, it won't be possible to introduce the required security without changing the system in some way that upsets a stakeholder group.

Risk Reduction

- Start considering the system's security as soon as you start developing its functional and information structure. This will allow you to understand the security needs early in the lifecycle and make sure that the system you design can be secured.

Ignoring the Insider Threat

As noted previously, while it is tempting to treat the system's "insiders" as trusted principals, you need to consider threats from people inside the organization at least as thoroughly as those from principals outside it. Insiders are often trusted and are often privy to knowledge about the organization, its systems, and the security controls that make circumventing a security control rather easier for them than for an outsider. They may also have grievances against their employer for all kinds of reasons, which could motivate them to attempt to damage the organization (e.g., by leaking confidential data). Perhaps it's no surprise, then, that in a 2010 survey² nearly 60% of British businesses confirmed that they had dealt with one or more incidents of unauthorized insider access to systems and data.

². The "Information Security Breaches Survey 2010," available from www.pwc.co.uk/eng/publications/isbs_survey_2010.html.

Risk Reduction

- Consider threats to the system that are specific to insiders who have partial or privileged access to it or just thorough knowledge of the system and its controls.
- Work with the enterprise security group, if there is one, to ensure that people do not retain authorizations that they no longer need in the event that their job changes.
- Consider possibilities for collusion between different insiders. Such situations are very difficult to guard against, but it may be worth introducing some form of monitoring for unusual situations that suggest some form of collusion.
- If your system is hosted outside your organization, consider whether insiders in the hosting organization can pose a threat to the system and its resources. Consider protecting externally hosted systems in a way that prevents the hosting company from gaining access to the system (e.g., by encrypting data to prevent its being read by employees of the hosting company who will have administrative access to the hosting environment).

Assuming the Client Is Secure

Years ago, when mainframes and proprietary mini-computers ruled the roost, controlling who could connect to your system was relatively straightforward. Users used proprietary “green screen” terminals to access the system, the terminals were all registered and authorized to connect to the application, and you could be pretty sure that they hadn’t been tampered with. Things couldn’t be more different today!

Nearly all modern information systems are created in a networked environment, with the main services of the system being accessed by clients of various sorts across a network. Internet-connected systems are obviously the most extreme example of this, with all of their public services being accessed through Web browsers and Web services, by a range of application clients largely outside the architect’s control. While most users may be using a market-leading browser on a desktop computer, some may be using mobile phones to access the system, and in a few cases people may even have written their own Web browsers! Even in the corporate environment, it is becoming more common for end users to demand access to applications on a variety of devices including smartphones, tablet computers, and their own computers at home.

In this dynamic and accessible environment we can’t reliably assume anything about the client devices that will be used to access our system, and we have to accept that from a security perspective, we have lost control of that part of the system’s deployment environment. The only option open to us in many cases is to embrace this diverse client mix as well as we can and to ensure that the system’s security does not rely on trusting the client’s behavior.

Risk Reduction

- Remember that almost anything can be changed on a compromised client device, and reflect this assumption in the system threat model.
- Even if security checks are performed on client devices (e.g., for performance or usability reasons), make sure that the definitive security check is performed in the system’s servers that are under your control.
- If it is feasible, consider limiting the devices that can connect to your system by checking their identity and accepting connections only from devices that have been deployed with a verifiable identity.
- If necessary, consider working with your enterprise security group in order to employ mechanisms that prevent data from your application from being saved on client computers (e.g., by preventing users from writing data to local removable storage).

Security Embedded in the Application Code

A problem we have observed in a number of systems occurs when the code that enforces the security model is found sprinkled throughout the application itself. The problems with this approach include reduced reliability, the difficulty of changing the security model being enforced, and the likelihood of introducing security errors in your system.

Risk Reduction

- Push as much of the security technology as possible into your underlying infrastructure elements.
- If security does need to be in application code, apply good software engineering judgment to encapsulate as much of it as possible in a single place. Also consider using software technologies (such as aspect-oriented programming or code generation) that would allow the security code to be applied automatically as part of the build process.

Piecemeal Security

To be effective, security needs to be considered on a holistic basis, throughout the system. A problem we have seen in many systems is security being applied to parts of the system but not to others. For example, highly sensitive data is encrypted during transmission but not when stored. This may or may not be a problem depending on the sensitivity of the data and the threats the system is likely to have to withstand. Your role is to make sure that security is implemented everywhere it is needed as revealed by the threat model—not just in the places that immediately spring to mind.

Risk Reduction

- The use of an architecture-driven development process will help address this risk. Make sure that you keep considering the architecture and particularly its security as a whole, rather than as a set of separate parts.

Ad Hoc Security Technology

Computer security is a specialized field with its own culture, standards, processes, and background. Security engineers (the people who build security technology and secure systems) tend to have a lot of specialized training and experience. Similarly, cryptographers (the people who study and create ways to encrypt and decrypt data) tend to have advanced degrees in cryptography. Despite this, many software developers without this specialized background fancy themselves as amateur security engineers or cryptographers and decide to create some or all of the security technology in their systems.

In general, we suggest that this is a bad idea—creating truly secure technology is harder than it looks, and without specialized training, most of us can't do it reliably. Given the integrated nature of security, the consequences of one piece of weak security technology can be pretty catastrophic.

Risk Reduction

- Use proven, widely accepted security technology from established providers when possible, and get expert help with its use and deployment.
- Make sure you find out what previous users of the possible technologies thought of it and how the security community as a whole rates it.
- If you have to create your own technology, engage expert assistance to help you systematically develop it.

Checklists

Checklist for Requirements Capture

- Have you identified the sensitive resources contained in the system?
- Have you identified the sets of principals who need access to the resources?
- Have you identified the system's needs for information integrity guarantees?
- Have you identified the system's availability needs?
- Have you established a security policy to define the security needs for the system, including which principals are allowed to perform which operations on which resources and where information integrity needs to be enforced?
- Is the security policy as simple as possible?
- Have you worked through a formal threat model to identify the security risks your system faces?
- Have you considered insider as well as outsider threats to the system?
- Have you considered how the system's deployment environment will alter the threats to the system?
- Have you worked through example scenarios with your stakeholders so that they understand the planned security policy and the security risks the system runs?
- Have you reviewed your security requirements with external experts?

Checklist for Architecture Definition

- Have you addressed each threat identified in the threat model to the extent required?
- Have you used as much third-party security technology as possible?
- Have you produced an integrated overall design for the security solution?
- Have you considered all standard security principles when designing your security infrastructure?
- Is your security infrastructure as simple as possible?
- Have you defined how security breaches will be identified and how to recover from them?
- Have you applied the results of the Security perspective to all of the affected views?
- Have external experts reviewed your security design?

Further Reading

You can find a short but thorough and very readable introduction to the main concepts of information systems security in an IEEE magazine article [\[LAMP04\]](#). A much deeper but still approachable introduction to the process of building secure systems and processes appears in Anderson [\[ANDE08\]](#); this book provides a

comprehensive introduction to many important security topics, as well as being an entertaining read, full of interesting stories from the security field.

A well-known, generally respected, and always colorful figure in the security field is Bruce Schneier. Two books that he authored or coauthored that offer useful background are [\[FERG10\]](#), which provides a good nuts-and-bolts introduction to cryptographic technology for software engineers, and [\[SCHN01\]](#), which moves beyond a technology-focused approach and explains how technology will never provide a complete solution to security problems. This second book is worth reading to understand just how complex the security field is and how broad effective security solutions need to be.

Two good, practical security books aimed at software developers are Viega and McGraw [\[VIEG02\]](#) and Howard and LeBlanc [\[HOWA04\]](#). Both books explain how to construct software that is secure by design rather than by accident or buzzword compliance. [Chapter 5](#) in the former book also presents and explains a practical and simple set of security principles. Gary McGraw's later book [\[MCGR06\]](#) takes the theme further, explaining how security concerns should be woven into the design process for secure systems. The Fault Trees approach, which forms the basis of the attack trees we introduced in this perspective, is explained in Leveson [\[LEVE95\]](#), and a simple guide to threat modeling can be found in [\[SWID04\]](#).

