 **Advanced Review**

Computer security

Dieter Gollmann∗

Computer security encompasses concepts and methods for protecting sensitive resources in computer systems. Computer security starts from the policies that regulate access to protected resources. In technology, the focus is on mechanisms for enforcing these policies. We will put various enforcement mechanisms into context with the policies and the IT architectures they were originally designed for. We will also briefly touch on network security and conclude with remarks on security evaluation .  2010 John Wiley & Sons, Inc. *WIREs Comp Stat* 2010 2 544–554 DOI:

10.1002/wics.106

**Keywords:** computer security; security engineering

# INTRODUCTION

S

ecurity comes into play once computer systems process sensitive data or run sensitive services. Computer security can be divided into four parts: controlling access to a computer system itself, controlling access to resources managed by the system, protecting data in transit between systems,

and securing applications against malign inputs.

A survey of access control has to consider the policies to be enforced and the enforcement mechanisms. In computer security, one can trace a development from stand-alone systems enforcing user-centric policies at the operating system level to distributed systems enforcing code centric policies in web browsers or even in web pages. These trends are driven by changes in the applications that make use of IT and go in hand with changes in the IT architectures. Concomitant issues are digital rights management (DRM), trusted computing, the move from policies for a single organization to policies in federated systems, and from the security evaluation of stand- alone systems to the challenges posed by a security assessment of today’s extensible software systems.

# FOUNDATIONS

New security challenges arise when new—or old—technologies are put to new use. The first elec- tronic computers were built in the 1940s (Colossus, EDVAC, ENIAC) finding applications in academia (Ferranti Mark I, University of Manchester), com- mercial organizations (LEO, J. Lyons & Co.), and government agencies (Univac I, U.S. Census Bureau) in the early 1950s. *Computer security* can trace its ori- gins back to the 1960s. For example, protection rings1

∗Correspondence to: [diego@tu-harburg.de](mailto:diego@tu-harburg.de)

Hamburg University of Technology, 21071 Hamburg, Germany DOI: 10.1002/wics.106

were introduced to solve the problem of protecting user and system data during process executions in the Multics operating system,2 an issue that had to be addressed in the emerging multiuser systems.

The RAND report by W. Ware3 summarized the technical foundations computer security had acquired by the end of this decade, such as supervisor and user mode, and protection rings. The report also provided a detailed analysis of the policy requirements of one particular application area, the protection of classified information in the U.S. defense sector. The RAND report was followed shortly after by the Anderson report4 that laid out a research program for the design of secure computer systems, again dominated by the requirement of protecting classified information.

In the further evolution of computer security, two strands of development are intertwined, the *security policies* a computer system is asked to enforce, and the *architectural layer* of the computer system where the policy is enforced. This article will show how computer security ranges from control mechanism in the system core related to simple, generic policies, to controls in application software dealing with a much wider range of security policies.

# Access Control

In computer security, *access control* preventing unau- thorized access to systems and resources is the main security service for achieving *confidentiality*, i.e., pre- venting unauthorized disclosure of information, and *integrity*, i.e., preventing unauthorized modification of information. Access control mechanisms define how a *subject* (process) may access a *resource* (object). Access control can be split into three main tasks:

The setting of the security policy; *principals* are authorized to access certain resources. Traditionally, principals were closely related to

•

**544**  2010 John Wiley & Sons, Inc. Volume 2 , September/October 2010

WIREs Computational Statistics Computer security

human users5 but this view is too restrictive for contemporary evidence-based security policies.

The *authentication* of evidence supplied with an access request. Traditionally, the principal making the request was authenticated.

•

The evaluation of the request with respect to the given policy. The early literature on access control refers to this task as *authorization* (of the request), see e.g., Ref 6.

•

The reader will observe an unfortunate over- loading of the term *authorization* that is applied to both principals and requests. A possible solution to this problem would be a convention that refers to ‘authorized principals’ and ‘approved requests’. At this point of time, the student of access control has to decide on the meaning of authorization—and also of authentication7—on a case-by-case basis. Figure 1 shows the view of access control traditionally adopted in operating systems research.

The *reference monitor* was originally defined as the abstract machine mediating all accesses to objects by subjects. The *security kernel* was the implementation of the reference monitor, required to mediate all accesses, be protected from modification, and be verifiable as correct. The *trusted computing base* (TCB) was the totality of protection mechanisms within a computer system responsible for enforcing a security policy. (Definitions following Ref 8.) Since, there has been some interpretation creep: the reference monitor component in current operating systems would strictly be the security kernel from above, and TCB is today sometimes used in a limited sense to stand only for the security kernel.

# OPERATING SYSTEM SECURITY

**Discretionary and Mandatory Access Control**

The two policies relevant for the early phases of computer security are known as *discretionary access control* and *mandatory access control*. In hindsight, the terms *user-centric access control* and *multilevel*

*Authorization*

*Authentication*

**....**.

Request

ACL

*security* would be more appropriate. In an operating system, access control decides whether a process (sub- ject) running under a given user identity (principal) is permitted access to a given resource (object).

The policy specifies what principals are allowed to do.

•

Authentication creates a process that runs under the *user identity* of the user authenticated during log on. The standard is password-based user authentication. Kerberos [RFC 4120] is often used for authenticating remote users.

•

A *policy decision point* checks whether access requests comply with the policy.

•

Discretionary access control policies assign the right to access protected resources to individual users. In the traditional formulation, a policy is given as a matrix recording permitted operations directly for each (*subject*, *object*) pair.9 (This presupposes that there has already been a mapping from current subjects to principals.) In actual implementations, the policy is typically stored in the form of *access control lists* for each object, stating the principals allowed to access the object, together with the permitted mode of access. *Capabilities* are a design alternative where each subject is associated with a data structure recording the objects the subject is permitted to access. *Groups* are an elementary layer of indirection in user-centric access control.

In mandatory access control, subjects and objects are labeled with *security levels*. Security levels form a (partially) ordered set. Traditional security levels are unclassified, confidential, secret, and top secret; more sophisticated policies refer to lattices of security levels. Security levels constitute a layer of indirection between subjects and objects. The policies for protecting the confidentiality of classified data demand that a subject can only read objects at its own or at a lower level (no read up). To prevent unauthorized declassification of information, a subject may only write to objects at its own or higher levels ( - property, no write down). The Bell-LaPadula model10 is a formal state machine model for discretionary and mandatory access control that had great influence on computer security research well into the 1980s. The Biba model gives dual data integrity policies based on

∗

11

**..**

..

**..**

Principal

Reference

monitor

Object

integrity levels.

A seminal result by Harrison, Ruzo, and Ullman12 shows that a core question in policy

**FIGURE 1** | Access control = authentication + authorization.

management is undecidable: in an authorization system where new subjects and objects can be created

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Volume 2 , September/October 2010  2010 John Wiley & Sons, Inc. **545**

**Advanced Review** wires.wiley.com/compstats

dynamically, can a given subject through some sequence of operations acquire a specific access right?

# Design Principles

Two core elements in the design of access control mechanisms are *status information* and *controlled invocation* (also called *restricted privilege*). To main- tain a distinction between actions on behalf of the system and actions on behalf of users, a system runs either in supervisor (kernel) or user mode (status infor- mation). When a user requests execution of a priv- ileged instruction, the system temporarily switches to supervisor mode and should execute the user request in a fashion that provides certain security safeguards (controlled invocation). Protection rings extend the distinction between user mode and supervisor mode to a more fine grained hierarchy of integrity levels.

Status information and controlled invocation can be supported already at the processor level. Most current processor architectures provide two status bits and machine instructions supporting the secure tran- sition to a higher status. Familiar operating systems like Linux, Unix, or Windows use two status modes only; the reference monitor is implemented in kernel space. Examples for controlled invocation are Unix *setuid* programs that run with the rights of the owner of the program rather than with the caller’s rights.

The design of security systems can be further guided by the protection principles formulated in Ref 13: economy of mechanisms, fail-safe defaults, complete mediation, open design, separation of privilege, least privilege, least common mechanism, and psychological acceptability.

Access control enforces a given policy. For any reasonable large system, it is unlikely that policy rules will be defined for all objects individually. Rather, default rules are defined that will be applied to newly created objects. Default rules may be associated with principals or with containers (directories). In the first case, the creator implies the default policy for a new object, in the second case the default policy depends on the container in which the object is placed. Inheritance rules can be used to specify more complex default policies, see e.g., the discussion on Active Directory in Ref 14.

# COMMERCIAL SECURITY

Multilevel security is definitely *military* security and, by default, not well suited when processing ‘unclassi- fied but sensitive’ data. Data can be sensitive because they have commercial value to a company; they can also be sensitive when they are personal data about

private individuals. Commercial security policies often put a high value on integrity (consistency). The commercial world has developed *well-formed transactions* for ensuring *internal consistency* prop- erties that relate to the internal state of a system, and *separation of duties* and *auditing* for ensuring *external consistency* properties that relate the internal state of a system to the real world. The Clark–Wilson model15 and the Chinese Wall model16 are two influential contributions to commercial security. The latter captures the dynamic conflict of interest policies that apply in financial consultancy businesses.

# Database Security

The objects in commercial security are more likely to be structured business records than the monolithic files of operating system access control. Business records stored in relational databases raise new challenges for access control, i.e., the consistent definition of policies for databases, tables, records, and fields. In addition, policies may have to take into account integrity relations between fields in a table, and between fields in different tables. *Foreign keys* are a prominent example for the latter case.

Statistical databases are characterized by the fea- ture that individual data may be sensitive and must not be disclosed, while statistical queries like count, sum, or average on those data are permitted. To avoid disclosure of information about individual entries, the size of query sets and of their complements has to be checked. However, this precaution is insufficient. Even if individual queries are guaranteed to be innocuous, combinations of ‘safe’ statistical queries could be used in so-called *tracker attacks* to infer information about a single entry. For more information on aggregation and inference in database security, and on counter- measures like randomization of query data, see Ref 17. Similar privacy challenges arise today in data mining.

# Role-based Access Control

Access rights are often assigned to job functions on a need-to-know basis. This is captured by *role-based access control* (RBAC). *Roles* are collections of pro- cedures; procedures can be well-formed transactions with built-in integrity checks. Users are assigned to roles and are permitted to execute the procedures linked to the role. The NIST model for RBAC distinguishes between18:

*Flat RBAC*: users are assigned to roles and roles to procedures; users get permissions to execute procedures via role membership; support for user-role reviews.

•

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

**546**  2010 John Wiley & Sons, Inc. Volume 2 , September/October 2010

WIREs Computational Statistics Computer security

*Hierarchical RBAC*: adds support for role hierarchies.

•

* *Constrained RBAC*: adds separation of duties. *Symmetric RBAC*: support for permission-role reviews (can be difficult to provide in large distributed systems).

•

Few operating systems support RBAC directly, but native access control features might be adapted for that purpose. RBAC is more often found in database management systems and workflow management systems. On a conceptual level, roles and procedures are layers of indirection between subjects and objects. *Administrative RBAC* applies the principles of RBAC when defining policies for the administration of policies.

# FEDERATED ACCESS CONTROL

When organizations join to form a federated security domain, the import of identities, credentials, policy rules, and decisions from different contexts (name spaces) becomes an important security issue. In a federation we may find several *Policy Administration Points* where policies are defined, *Policy Decision Points* where access requests are evaluated against a given policy, *Policy Enforcement Points* where the decisions are enforced, and *Policy Information Points* where additional evidence that may be needed for evaluating an access request can be obtained from.

*Trust management* as originally introduced in PolicyMaker19 refers to access control systems for such scenarios. *Federated identity management* deals with the management of digital identities in a federation, and in particular with single sign-on in a federation. In web services, related standards for authentication (SAML) and access control (XACML) have been defined.

# INTERNET SECURITY

Opening the Internet to commercial use in the early 1990s raised the importance of security policies for remote transactions. Sensitive data had to be protected in transit, and nodes on the Internet had to be shielded from unwelcome access. Worms and viruses, first

communications in an insecure network. The Secure Socket Layer protocol (SSL) developed by Netscape delivers this service at the transport layer. SSL has been adopted by the IETF as Transport Layer Security (TLS). The most recent version, TLS v1.1, is specified in RFC 4346. SSL/TLS is designed for a client/server scenario.

Typically, the server has been issued with a digital certificate; the client needs suitable root verification keys to be able to verify the certificate and authenticate the server. Client authentication is optional. For this approach to be effective, an infrastructure that issues server certificates and equips clients with verification keys is needed. Digital certificates can be obtained from so-called *certification authorities* (CAs). The verification keys of popular CAs are pre-installed in web browsers.

The Hypertext Transfer Protocol (HTTPs) for creating secure browser sessions is based on SSL/TLS. When connecting to a server, the client browser matches the URL of the connection with the domain name from the server certificate. A secure session thus created has the client browser and the server as its endpoints. Within this secure transport layer session, user authentication may be performed. After successful authentication a secure application layer session can be created based on a *user authenticator*, e.g., an HTTP cookie.

Deploying *https* as a security service in an application like e-banking, where the transaction endpoints are customer and bank, relies on the user’s ability and willingness to check that the URL in the browser bar is correct. Conversely, one may conclude that for those applications SSL/TLS creates secure sessions at a wrong layer. Cookies authenticating the user at the application layer sent in supposedly secure SSL/TLS sessions with a wrong endpoint can be disclosed to an attacker (Figure 2). Therefore, authen- tication has to be lifted to the transaction endpoints. This issue is addressed, e.g., in Ref 22; cookie values depending on the current SSL/TLS session can alert the participants to a mismatch between the session endpoints at transport and application layer.

**.**

|  |  |  |
| --- | --- | --- |
| Applicati | on laye | r session |
|  |  | Transport layer session |

Cookie

Cookie

**..**.

discussed in research papers20,21 became real security challenges.

# Secure Channels

**...**.

.**....**

User/ browser

Transport layer session

Man in the middle

|

Server

Cryptography provides means for building secure channels between end points, thereby protecting

**FIGURE 2** Man-in-the-middle attack on secure application layer sessions.

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Volume 2 , September/October 2010  2010 John Wiley & Sons, Inc. **547**

**Advanced Review** wires.wiley.com/compstats

IPsec (RFC 4301) provides cryptographic services similar to SSL/TLS at the network layer. IPsec is more generally an access control system at the network layer. IPsec policies at a node specify whether packets are passed through, dropped, or cryptographically encapsulated. These policies refer to source and destination IP addresses, and Trans- mission Control Protocol (TCP) and User Datagram Protocol (UDP) port numbers.

# Firewalls

Similar access control services in the network are provided by firewalls. *Firewalls* are devices controlling the flow between two networks, mostly an internal network (intranet) and the Internet. Packet-filtering firewalls make decisions based on IP addresses and port numbers. *Stateful* packet-filters handle situations where an internal node opening a connection to an outer network is permitted to receive data via this connection. *Application-level proxies* intercept traffic and reassemble application-level messages, perform checks at that layer, and may forward a sanitized message to the receiver.

# Intrusion Detection

An *intrusion detection system* (IDS) monitors traffic (network-based intrusion detection systems, NIDS) or log files (host-based intrusion detection systems, HIDS) and issues alerts when a suspected attack is detected. *Knowledge-based* intrusion detection relies on databases of known attack patterns. The challenges here are keeping this database up-to-date and facili- tating fast search. *Anomaly-based* intrusion detection is based on the premise that intrusions are observable deviations from normal behavior. First, a baseline of normal behavior is established in a safe environment. Later, during operation the system is monitored and the deviation from the baseline behavior is measured. If it exceeds a threshold, an alarm is raised. This approach would have the advantage that it does not have to know about specific attack characteristics and could detect so-called zero-day exploits. The challenge here is substantiating the premise: is it really the case that deviation from normality indicates an attack?

In practice, knowledge-based methods are being deployed. The number of alerts is considerable and beyond the capacities of direct human analysis. *Con- text aware* IDSs suppress alarms that are not relevant for the given network and software configuration. The use of data mining techniques for extracting high priority alerts is being explored today. *Honeypots* are systems deployed for the sole reason of monitoring

attacker behavior, with the specific aim of detecting novel attacks.

# SOFTWARE SECURITY

Software processing external inputs can be attacked by supplying intentionally malformed inputs. Software is secure if it can resist such attacks. There are two fundamental classes of countermeasures. Either software has to be executed in a way that malformed inputs have no malign effects, or the inputs have to be preprocessed so that only well formed and benign inputs are eventually being processed.

# Memory Management

The attack that created general awareness of software security is the *buffer overrun* (overflow) on the stack described in ‘Smashing the Stack for Fun and Profit’.23 The attack targets software that writes input data to a buffer without properly checking the amount of data being written. The attacker provides malformed input that exceeds the size of the buffer. When processing this input, user-supplied data is written beyond the buffer overwriting the return address on the stack. Execution will then resume at a location specified by the attacker, allowing the attacker to run so-called *shell code*.

As defenses, programs can be rewritten to include checks on input lengths; check values (*canaries*) can be placed in memory adjacent to the return address for detecting that the return address may have been overwritten.24 A nonexecutable stack blocks attempts to pass the shellcode as an input to the attacked software component (when inputs are stored on the stack). With a control stack separate from the data stack, user data cannot affect control data such as the return address.25

Another attack pattern exploits errors in the control flow of a program. For example, when an uninitialized object is returned to the memory manage- ment system, it is often the case that control data used in this step has not been defined but memory manage- ment will nevertheless go to the locations where these data are expected and read whatever it finds there. If the attacker is able to predict these locations and load them with malformed values in advance, the memory management function will process the attacker’s data. Returning uninitialized objects is a flaw that typically arises when a conditional branch jumps to the end of the program where the object is returned, but the object is assigned a value only in another branch. *Double-free* and *uninitialized memory corruption* attacks fall into this attack category.

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

**548**  2010 John Wiley & Sons, Inc. Volume 2 , September/October 2010

WIREs Computational Statistics Computer security

# Race Conditions

A *race condition* occurs when two executions access the same object, and the final result depends on the order of the two access operations. A security relevant race condition in Unix systems is described in Ref 26. A *setuid* to root program checks whether its caller may access a file using the access() system call that utilizes the effective uid of the current process when making the check. When access is permitted, the file is opened with the open() system call. When the filename refers to a file owned by the attacker when access() is called and is changed to refer to some other file when open() is called, the program can be misused to manipulate files of the attacker’s choice, e.g., /etc/passwd. This situation is known in the security literature as the TOCTTOU (time of check to time of use) problem.

# Code Injection

Software security problems have been moving from operating systems to applications software for a num- ber of years. *SQL injection* attacks target software architectures where an application script assembles a database query from user inputs and predefined code fragments. When the query is constructed as a string to be passed to the database management system, malformed inputs may inject additional commands that are then executed within the application. As defenses, one can either filter the user inputs—script- ing languages like PHP provide sanitizing functions for this purpose—or change the way database queries are being performed. With *prepared statements* the application script is precompiled with place holders for the user inputs. During execution, the user inputs are inserted for the placeholders.

# Language-based Security

Type-safe languages like Java and C# are often recommended as a generic defense against software security problems. Type-safe languages guarantee the absence of untrapped errors. It is indeed the case that automatic garbage collection avoids many of the memory management problems faced by C programmers. However, type safety does not solve all problems in software security.

The design of a type-safe language and runtime system is not straightforward; several flaws in type systems have been identified in the past.27

•

Sensitive data like passwords need special atten- tion to ensure that they are deleted completely

•

and immediately once they are no longer needed;

*cache pinning* serves this purpose.

A type system deals with the errors it is defined for, but not with all possible causes of software vulnerabilities. The Java type system, e.g., does not eliminate race conditions; Java programmers have to deal with race conditions explicitly with *synchronized* methods or statements.

•

When a type system traps an error, it throws an exception; it is then up to the exception handler to actually deal with the problem.

•

Errors in native code or at the hardware level may unhinge type safety.28

•

*Wrappers* are a method for implementing input—and output—checks outside the program to be protected. In object-oriented systems that support reflection, security checks can similarly be performed in the meta-object while the original object remains unchanged. *In-line reference monitors* rewrite code to add the checks required to enforce the given security policy.29,30

*Taint analysis* marks inputs from untrusted sources and traces how these inputs are further pro- cessed. An alarm is flagged when an untrusted input reaches an exploitable statement, a so-called trust sink, without having been checked by a *sanitizing* function. Static taint analysis is applied to source code; dynamic taint analysis is applied at run time. In *model checking* a program is modeled as a finite state machine; in this model it is then checked whether all executions satisfy a given security, see e.g., Ref 31.

# CODE-BASED ACCESS CONTROL

User-centric access control fits closed organizations with well-defined user populations. Open applications may by their very nature give access to unknown users, or user identities, even if known, may be of little help for security when users cannot be held accountable. Access decisions may furthermore relate to programs that request to be executed, e.g., a Java applet, rather than to low level read and write operations. In such a situation it becomes natural to use security policies that refer to some attribute of the code that should be executed.

Code-based access control can be found in the Java security model32 and in Microsoft’s .NET architecture33. Privileges are assigned depending on code origin and not according to the identity of the user running a program. Origin can subsume the domain the code was obtained from, the identity of the code signer, a specific name space (strong name),

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Volume 2 , September/October 2010  2010 John Wiley & Sons, Inc. **549**

**Advanced Review** wires.wiley.com/compstats

and more. The .NET framework provides a range of mechanisms for managing code-based security policies. *Declarative* security policies are defined in meta-data, *imperative* policies in the code itself.

The reference monitor enforcing these policies is found in the Web browser (sandbox). Typically there is a single generic access decision function that can sup- port application defined policy attributes (evidence). A *stack walk* is performed to check that all callers have been granted the required access rights. The stack walk deals with the *confused deputy* problem,34 where an unprivileged attacker manipulates a system via calls to privileged code. Controlled invocation is implemented through *assert* statements. A stack walk for an access right will stop at a caller that asserts this right.

# Trusted Computing

To tackle the twin challenges of software security and the protection of intellectual property rights the Trusted Computing Platform Alliance (now Trusted Computing Group) was founded in 1999 to ‘make the web a safer place to surf’. The goals of trusted computing have changed since. The Trusted Platform Module (TPM) is an advanced cryptographic copro- cessor that supports remote attestation as a security service providing trustworthy information about the software configuration on a machine.

*Direct anonymous attestation* implements this service in a way that protects user privacy.35 Remote attestation can be used with security policies that are predicated on the software running on a remote machine. For example, a document owner could check the software configuration at a destination before releasing a document. Or, the provider of a cloud computing service could use attestation to demonstrate to customers that the platform provided has indeed the characteristics advertized.

# Digital Rights Management

The security policy in DRM deviates from traditional access control as it is not the goal to protect the system owner from external parties, but to impose the security policy of an external party on the system owner. DRM needs a tamper resistant enforcement mechanism and was one of the driving forces of trusted computing. A conceptually interesting alternative is offered on mobile phones when the manufacturer controls the operating system running on the phone and the access rights granted to applications.

Content owners need not rely on technology only but can also take recourse to the legal system. For example, EMI announced in 2007 that they would

provide content not protected by DRM mechanisms.*a* The similarities between legal code and software code as alternative strategies for enforcing desired behavior were noted by Lessig.36

# WEB SECURITY: ORIGIN-BASED ACCESS CONTROL

In web applications, client and server communicate via the HTTP protocol; the client browser sends HTTP requests; the server returns result pages. The browser represents the page internally in the document object in the Domain Object Model (DOM). Security policies regulate what scripts in a web page are allowed to do. Policies can specify which host may be accessed, or which connections may be opened. In the terminology of access control, web applications are principals, and have to be named. The current convention uses host names from the Domain Name System (DNS) for naming services; the policy decision point at the client side is located in the Web browser.

# Code Origin Policies

The prototype policy for Web applications is the *same origin policy*, stating that an applet may only connect back to the domain it came from or that a cookie is only included in requests to the domain that had placed the cookie. Two pages have the same origin if they share protocol, host name, and port number (Table 1). Certain actions may be exempted from the same origin policy. For example, a web page may contain links to images from other domains, reflecting a view that images are innocuous data without malign side effects.

# Cross-site Scripting

Web applications are targeted by SQL injection (see above) and cross-site scripting (XSS) attacks. XSS is an *elevation of privilege* attack exploiting the client’s ‘trust’ in a web server; i.e., a web page from the trusted server is processed with more access rights than a page from the attacker’s domain. Malign code is passed to the client via the trusted server, violating the client’s origin-based security policy.

In a *reflected* XSS attack (Figure 3), malign code is placed within a link to the trusted server in a page in the attacker’s domain. When the victim visits this page, the client’s browser renders the page and processes the link to the trusted server, sending a request containing the malign code to the server. When the server echoes back client input in its response page, the malign code

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

**550**  2010 John Wiley & Sons, Inc. Volume 2 , September/October 2010

WIREs Computational Statistics Computer security

**TABLE 1**

Same Origin Policy

URL Result Reason

<http://www.my.org/dir1/some.html> Success <http://www.my.org/dir2/sub/more.html> Success [https://www.my.org/dir2/some](http://www.my.org/dir2/some.html).html Failure Different protocol http://www.my.org:81/dir2/some.html Failure Different port <http://host.my.org/dir2/some.html> Failure Different host

Evaluating same origin for [http://www.my.org/dir1/hello.html.](http://www.my.org/dir1/hello.html)

may be established at the transport layer by SSL, or at the application layer by password-based HTTP authentication, or by any other means. Requests within this session are executed with security per- missions attributed to the client. XSRF thus evades the target’s origin-based security policy. The target fails to enforce its code origin policy because it can

.**...**

**...**.

.**.**.**.**

Page click

Bad code

Request

|  |  |  |  |
| --- | --- | --- | --- |
| Page | Link to server | Bad code |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Page | Bad code |  | Cookie |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Request | Cookie |  |  |

Server

User/browser

Attacker

only authenticate the last stepping stone of a request, which is not necessarily its true origin.

**FIGURE 3** | Reflected cross-site scripting with cookie stealing.

is executed at the client within the server’s page. Typi- cal examples for applications that echo client input are search engines or custom 404 (page not found) pages. In *stored* XSS, the attacker places a page contain-

ing malign code directly at the server. Bulletin board applications are candidates for stored XSS. When the victim visits the attacker’s entry in the bulletin board, the code in the entry is executed at the client.

In *DOM-based* XSS the attack vector split into two parts. Malign code is embedded in the URL of the attacker’s web page; this page also contains a link to a page on the trusted server that references the URL in the DOM when being loaded by the client’s browser. When the victim visits the attacker’s page, the client’s browser will store the bad URL in document.URL and request the web page from the trusted server. When this page is loaded, document.URL will be referenced and the attacker’s code is executed.

XSS can be employed for *cookie stealing*. A cookie set by the server in the client browser will be accessible to scripts loaded from the attacker’s page. Such a script can then forward the value of the cookie to the attacker, e.g., as a parameter in an HTTP GET request (see Figure 3). The attacker can then use the cookie to directly impersonate the client to the server.

# Cross-site Request Forgery

A cross-site request forgery attack (XSRF, also cross-site reference forgery, session riding) executes malicious code at a target website with the privileges of a ‘trusted’ client. Here, trust is an authenticated session between client and web server. The session

In a *reflected* XSRF attack the attacker’s webpage contains hidden actions at the target site, e.g., in a form tag. When the user visits the attacker’s page, the browser automatically submits the form data to the target. If the client has established an active session to the target at the same time, the target will interpret the request as coming from the client, and the form data is accepted by the server as coming from an authenticated user.

In a *stored* XSRF attack the malign code is stored at the server. When a client requests an application page that contains the attacker’s code, the page will direct the client’s browser back to the application and actions inserted by the attacker are executed as coming from the client.

**Countermeasures against XSS and XSRF** Countermeasures fall into three general categories. First, one could change the execution model of web applications and block the execution of scripts from untrusted web pages in the browser. Today, this would severely restrict the user in the way that resources in the Web can be used.

Second, we could treat these attacks as instances of *code injection* attacks. Servers may sanitize their outputs by appropriate encoding or escaping; clients may filter their inputs. For this strategy to work, all dangerous inputs, all accepted encodings of dangerous inputs, all avenues inputs can enter our system, and even all ways internal components may translate innocuous inputs into dangerous inputs passed on to some other component have to be known.

Third, authentication can be improved. Authen- tication could be based on a temporary secret shared

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Volume 2 , September/October 2010  2010 John Wiley & Sons, Inc. **551**

**Advanced Review** wires.wiley.com/compstats

by client and server, established when a session is being started. It must be stored in a location that is not accessible to scripts executing in the browser. For authentication, standard cryptographic techniques such as message authentication codes can be used. This approach requires the cooperation of client and server. Alternatively, clients can provide a defense for application layer (but not for SSL) sessions on their own, by *recognizing their own* requests and checking whether a URL is *the same as before*. A proxy placed between browser and network marks all URLs in incoming web pages with an unpredictable token and keeps a database associating tokens with domains. It also checks outgoing requests for the presence of a token. Requests without a token are local requests and are passed on. Requests with a token did not originate in the client; if the origin of the request matches the domain it is being sent to, the request complies with the same origin policy and is passed on. Otherwise, all

user authenticators are removed from the request.37

# SECURITY EVALUATION

Customers purchasing and deploying an IT system may desire to get some assurance about its security. *Security evaluation* is meant to address this need, analyzing the level of security provided by a product given a set of generic security requirements. The results are of potential interest for a larger number of customers. *Certification* analyses a product with respect to the requirements of a specific customer. *Accreditation* is the customer’s decision to deploy specific products. (Definitions from Ref 8.)

The Trusted Computer Systems Evaluation Criteria (Orange Book) was a first influential contri- bution to security evaluation.8 It was driven by the requirements of the defense applications of the 1970s and in essence written for evaluation of operating sys- tems. The demands of commercial security triggered developments in security evaluation that led to the Common Criteria.38 The Common Criteria have been applied to security controls in the core of IT systems and to add-on security components.

Smart cards are a success story of security evaluation. High assurance evaluation has revealed flaws in products. Factors contributing to this success are market demand, functionality that is by and large fixed so evaluation results do not immediately get out of date, and the fact that smart cards are primarily security devices. The security evaluation of operating systems has not been a similar success. There is a wide variety of user requirements. Functionality is complex and evolving so evaluations are almost by default out of date. High assurance evaluation is hardly feasible

and lower assurance evaluation hardly finds flaws. Security mechanisms are increasingly placed outside the operating system, e.g., in browsers. Hence, the use of an evaluated operating system may not deliver the desired security guarantees.

Evaluation of application software is even more challenging. Applications are heavily customized so rapid and adaptable evaluation methodologies are needed. Security at the application layer is moving closer to the end user; hence, unsophisticated users have to be considered when assessing the usability of security features.

# CONCLUSIONS AND OUTLOOK

Computer security enforces security policies. Security policies have to be written in some language. Lists are commonly used in operating systems and firewalls for specifying policies. Policy languages are intimately linked to an algorithm for evaluating a given policy. A list is usually processed from the start until the first matching entry is found, which limits the sophistication of policies that can be expressed. In Turing complete policy languages, questions about policies may be undecidable. Layers of indirection are often introduced to make policies more manageable, employing a well tried general method for managing complexity. Research on policy languages is striving for a good balance between the expressiveness of the language and the strength of its formal foundations.

Today’s PC is a single-user multi-application

machine that needs separation between applications, and not so much between users. *Virtualization* can provide a separate virtual machine for each of the applications so that an attack compromising one application still cannot affect other applications. *Noninterference* provides theoretical foundations for studying separation properties in the presence of possible interactions between components.39

Within a software architecture, there are several options for implementing a reference monitor (Figure 4). Programs in user space can submit access requests to a reference monitor in kernel space.

1. **(b) (c)**

RM

Program

|  |  |
| --- | --- |
| RM | |
|  | Program |

|  |  |
| --- | --- |
| Program | |
|  | RM |

User space

Kernel

**FIGURE 4** Placing the reference monitor. (a) RM in kernel, (b) interpreter, and (c) In-line RM.

|

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

**552**  2010 John Wiley & Sons, Inc. Volume 2 , September/October 2010

WIREs Computational Statistics Computer security

So-called *execution monitors* consider individual requests only when making their decision. Access control in operating systems follows this approach. A classification of the policies enforceable by execu- tion monitors is given in Ref 40. The browser sandbox puts code (Java applets) into the reference monitor and enforces the security policy when interpreting the pro- gram. There are indications that browser vendors are moving to access control solutions similar to those found in operating systems. In-line reference monitors place the reference monitor into the code.

In software security, awareness and tools have to be brought to application writers. Anyone writing code that accepts external input is potentially writing security relevant code. Note that external input may have been left by the attacker in an internal database, so also code that does not communicate with external entities may be vulnerable.

Technology on its own is rarely sufficient for solving security problems. Organizational procedures tend to be equally important. *Compliance* emphasizes best practices in security and in management. Rele- vant documents are the BSI baseline protection, Cobit, the ISO 27000 series, ITIL, HIPAA, and the Sarbanes- Oxley act. It should be noted that in the early days, computer security was managed by professionals. Today, end users are increasingly called up to par- ticipate in security management. They have to install security software, patch their software, and make pol- icy decisions on the fly. This puts a new dimension into the design of computer security solutions.

# NOTES

*a*Press release, 2 April 2007, [http://www.](http://www/) emigroup.com/Press/2007/press18.htm

# REFERENCES

1. Graham RM. Protection in an information processing utility. *Commun ACM* 1968, 11:365– 369.
2. Corbato` FJ, Saltzer JH, Clingen CT. Multics–The first seven years. *Proceedings of the 1972 Joint Spring Con- ference*: AFIPS Press; 1972.
3. Ware WH. Security controls for computer systems. Technical Report R-609, The RAND Corporation, Santa Monica, CA, January 1970.
4. Anderson J. Computer security technology planning study. Technical Report 73-51, U.S. Air Force Elec- tronic Systems Technical Report, October 1972.
5. Gasser M. The role of naming in secure distributed systems. *Proceedings of the CS’90 Symposium on Com- puter Security*. Rome, Italy; 1990, 97– 109.
6. Lampson B, Abadi M, Burrows M, Wobber E. Authen- tication in distributed systems: Theory and practice. *ACM Trans Comput Syst* 1992, 10:265– 310.
7. Gollmann D. Authentication by correspondence. *IEEE J Sel Area Comm* 2003, 21:88– 95.
8. U.S. Department of Defense. *DoD Trusted Computer System Evaluation Criteria*. 1985. DOD 5200.28-STD.
9. Lampson B. Protection. *ACM Oper Syst Rev* 1974, 8:18– 24.
10. Bell D, LaPadula LJ. Secure computer systems: math- ematical foundations and model. Technical Report M74-244, The MITRE Corporation, Bedford, MA, May 1973.
11. Biba KJ. Integrity consideration for secure computer systems. Technical Report ESD-TR-76-372, MTR- 3153, The MITRE Corporation, Bedford, MA, April 1977.
12. Harrison MA, Ruzzo WL, Ullman JD. Protection in operating systems. *Commun ACM* 1976, 19:461– 471.
13. Saltzer JH Schroeder MD. The protection of infor- mation in computer systems. *Proc IEEE* 1975, 63: 1278– 1308.
14. Swift MM, Hopkins A, Brundrett P, Dyke CV, Garg P, Chan S, Goertzel M, Jensenworth G. Improving the granularity of access control for Windows 2000. *ACM Trans Inf Syst Secur* 2002, 5:398– 437.
15. Clark DR, Wilson DR. A comparison of commercial and military computer security policies. *Proceedings of the 1987 IEEE Symposium on Security and Privacy*; 1987, 184– 194.
16. Brewer DFC, Nash MJ. The Chinese Wall security policy. *Proceedings of the 1989 IEEE Symposium on Security and Privacy*; 1989, 206– 214.
17. Denning DE. *Cryptography and Security*. Reading, MA: Addison-Wesley; 1982.
18. Sandhu RS, Ferraiolo David, Kuhn R. The NIST model for role based access control: Toward a unified stan- dard. *Proceedings of the 5th ACM Workshop on Role Based Access Control*; 2000, 47– 63.
19. Blaze M, Feigenbaum J, Lacy J. Decentralized trust management. *Proceedings of the 1996 IEEE Sympo- sium on Security and Privacy*; 1996, 164– 173.

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

Volume 2 , September/October 2010  2010 John Wiley & Sons, Inc. **553**

**Advanced Review** wires.wiley.com/compstats

1. Shoch JF, Hupp JA. The ‘‘worm’’ programs—early experience with a distributed computation. *Commun ACM* 1982, 25:172– 180.
2. Cohen F. *Computer Viruses*. PhD thesis, University of Southern California, 1985.
3. Oppliger R, Hauser R, Basin DA. SSL/TLS session- aware user authentication. *IEEE Comput* 2008, 41: 59– 65.
4. One Aleph. Smashing the stack for fun and profit.

*Phrack Magazine* 1996, 49.

1. Cowan Crispan, Pu Calton, Maier D, Walpole J, Bakke P, Beattie S, Grier A, Wagle P, Zhang Q, Hinton Heather. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. *Proceed- ings of the 7th USENIX Security Symposium*. 1998, 63– 78.
2. Lee RB, Karig DK, McGregor JP, Shi Z. Enlisting hard- ware architecture to thwart malicious code injection. In: *Proceedings of the International Conference on Secu- rity in Pervasive Computing (SPC-2003), LNCS 2802*. New York: Springer Verlag; 2003, 237– 252.
3. Bishop M, Dilger MM. Checking for race conditions in file accesses. *Comput Syst* 1996, 9:131– 152.
4. Paul N, Evans D. NET security: Lessons learned and missed from Java. *Proceedings of ACSAC 2004, Tucson, AZ*. 2004, 272– 281.
5. Govindavajhala S, Appel AW. Using memory errors to attack a virtual machine. *Proceedings of the 2003 IEEE Symposium on Security and Privacy*; 2003, 154– 165.
6. Erlingsson U´ , Schneider FB. IRM enforcement of Java stack inspection. *Proceedings of the 2000 IEEE Sym- posium on Security and Privacy*; 2000, 246– 255.
7. Erlingsson U´ , Abadi M, Vrable M, Budiu M, Necula GC. XFI: software guards for system address spaces.

*Proceedings of the 7th Symposium on Operating Sys- tems Design and Implementation*, Seattle, WA; 2006, 75– 88.

1. Chen H, Wagner D. MOPS: an infrastructure for examining security properties. *9th ACM Conference on Computer and Communications Security*: Springer Ver- lag; 2002, 235– 244.
2. Gong L, Dageforde M, Ellison GW. *Inside Java 2 Plat- form Security*. 2nd ed. Reading, MA: Addison-Wesley; 2003.
3. La Macchia BA, Lange S, Lyons M, Martin R, Price KT. *NET Framework Security*. Boston, MA: Addison- Wesley Professional; 2002.
4. Hardy N. The confused deputy. *Oper Syst Rev*

1988, 22:36– 38.

1. Brickell E, Camenisch Jan, Chen Liqun. Direct anony- mous authentication. In: Pfitzmann B, Atluri V, McDaniel PD, eds. *Proceedings of the 11th ACM con- ference on Computer and Communications Security*. New York: ACM Press; 2004, 132– 145.
2. Lessig L. *Code and Other Laws of Cyberspace*. New York: Basic Books, 1999.
3. Johns Martin, Winter J. RequestRodeo: Client side protection against session riding. In: Piessens F., ed. *Proceedings of the OWASP Europe 2006 Conference*: Departement Computerwetenschappen, Katholieke Universiteit Leuven Report CW448; 2006, 5– 17.
4. CCIB. Common Criteria for Information Technology Security Evaluation, September 2006. Version 3.1.
5. Goguen JA, Meseguer J. *Security policies and security models*. *Proceedings of the 1982 IEEE Symposium on Security and Privacy*. 1982, 11– 20.
6. Schneider FB. Enforceable security policies. *ACM Trans Inform Syst Secur* 2000, 3:30– 50.

19390068, 2010, 5, Downloaded from https://wires.onlinelibrary.wiley.com/doi/10.1002/wics.106 by Test, Wiley Online Library on [30/06/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

**554**  2010 John Wiley & Sons, Inc. Volume 2 , September/October 2010