Chapter 1

Vulnerabilities in Information Systems

Cyberspace: From terra incognita to terra nullius.

Introduction

Vulnerability in any system is the result of an intentional or unintentional omission or of an inadvertent design mistake that directly or indirectly leads to a compromise in the system's availability, integrity, or confidentiality. In information assurance, vulnerabilities may hide in each level of security, be it information access security, computer and storage security, communications security, or operational and physical security. In the case of information systems, the major components are people, hardware, and software. Therefore, the presence of vulnerabilities must be sought in each of these three areas. Figure 1.1 illustrates the factors that contribute to a secure cyberspace and the expectations, out of cybersecurity.

Over half a century ago, designers, engineers, and scientists successfully quantified the concept of reliability for the design and maintenance of hardware and of software to a lesser extent. Today, efforts are being made to quantify the abstract concept of vulnerability as it applies to the security of information systems. The aim is to express the perceived level of security in a way that is measurable, standardized, and understood and to improve "the measurability of security through enumerating baseline security data, providing standardized languages as means for

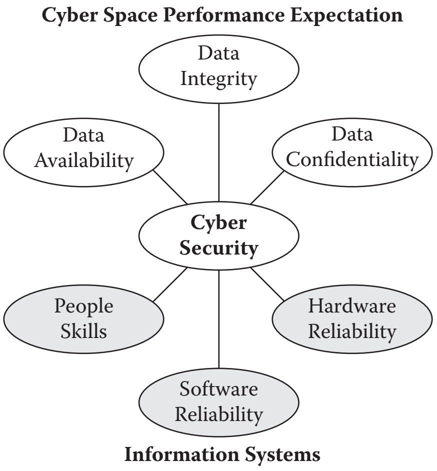


Figure 1.1 Cybersecurity serving as the infrastructure of the cyberspace.

accurately communicating the information, and encouraging the sharing of the information with users by developing repositories [1].”

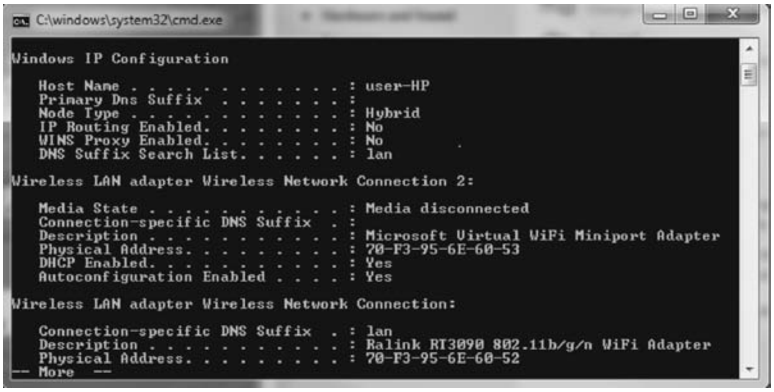
Vulnerabilities can be hidden in data, code, and most often in processes that inadvertently allow unauthorized access. Intrusions, however, can occur not only in the Internet, but also in the intranets, where most often security is not as strong. Security can be strengthened through intelligent mechanisms of authentication applied at both ends—the user side as well as the server side.

At the user side, authentication can be greatly fortified with the introduction of additional mechanisms, such as one-time passwords (OTP), provided via intra- or extra-Internet channels [2]. Such channels can be biometrics, questionnaires, or additional transparent parameters related to the user device identification numbers, such as manufacturer’s serial number, Media Access Control (MAC), or International Mobile Equipment Identity (IMEI).

MAC, also referred to as *Physical Address*, is a 48-bit number, expressed as 12 hexadecimal digits, that uniquely identifies the network interface of the computer. The network interface circuit may be an insertable network card or may be embedded in the computer’s motherboard. Figure 1.2 shows how the MAC address of a personal computer can be identified.

IMEI similarly uniquely identifies devices that utilize mobile telephony and is a number, usually 13 to 15 digits long. Mobile telephony providers assign a telephone number to devices linking them to their IMEI identification. Figure 1.3 illustrates the IMEI numbers available inside mobile phones.

In addition to the available MAC and IMEI numbers, device serial numbers and network parameters can also be used for authentication, such as intranet and Internet addresses. The above apply to client authentication toward the server.



```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : user-HP
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No
   DNS Suffix Search List. . . . . . : lan

Wireless LAN adapter Wireless Network Connection 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Microsoft Virtual WiFi Miniport Adapter
   Physical Address. . . . . . . . . : 70-F3-95-6E-60-53
   DHCP Enabled. . . . . . . . . . . : Yes
   Autoconfiguration Enabled . . . . : Yes

Wireless LAN adapter Wireless Network Connection:

   Connection-specific DNS Suffix  . : lan
   Description . . . . . . . . . . . : Ralink RT3090 802.11b/g/n WiFi Adapter
   Physical Address. . . . . . . . . : 70-F3-95-6E-60-52
-- More  --
```

Figure 1.2 MAC address in a personal computer, 70-F3-95-6E-60-52.

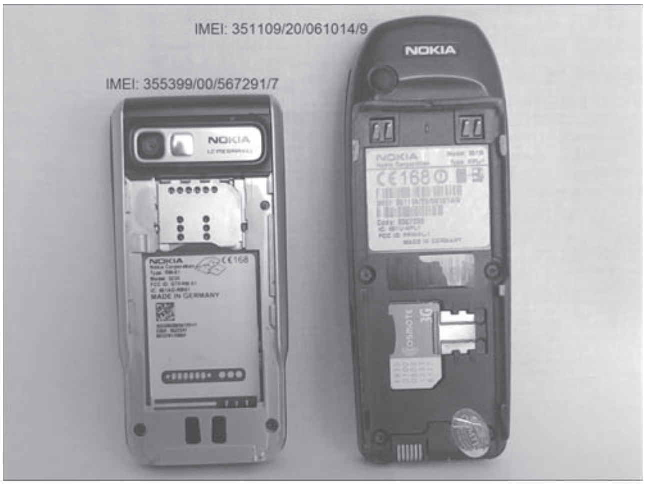


Figure 1.3 Mobile phone IMEI numbers.

At the server side, the use of certificates, IP restrictions, and data encapsulations can greatly enhance authentication and security. While in transit, data can be protected by hash codes, such as the *cyclic redundancy code* (CRC), and by the private key/public key encryption mechanisms.

Vulnerabilities in information systems can originate in a very wide variety of causes, ranging from firewall penetrations and Trojan horse attacks to decentralization and static resource allocation. Most frequently, vulnerabilities are introduced while systems are being upgraded or adapted to new operational environments.

Measuring Vulnerability

The US National Institute of Standards and Technology (NIST) has developed a protocol for the standardized classification and assessment of the security content in a software system in order to "standardize the way security vulnerability and configuration information is identified and catalogued [3]." The protocol is named Security Content Automation Protocol (S'CAP, pronounced "es-cap") and consists of the following six components.

1. **Common Vulnerabilities and Exposures (CVE).**[*] This is a depository of registered known information security vulnerabilities, where each occurrence has its own unique identification number. Initially, such occurrence is defined as a candidate vulnerability, and if accepted it becomes an *entry* registered in the MITRE CVE List [4] and is eventually registered in the National Vulnerability Database (NVD) [5]. As of late summer 2010, the NVD contained 43,163 CVE Vulnerabilities, increasing at a rate of 11 vulnerabilities per day. In that database, one will find security weaknesses of numerous software, including well-known operating systems and Internet browsers.
2. **Common Configuration Enumerator (CCE).**[†] This is a similar depository, but contains security vulnerabilities and interfacing inconsistencies found in system configurations. Such information can help in regulatory compliance, in determining proper interoperability, as well as with audit checks. The provided information is in narrative form identifying existing problems and usually recommending solutions [6].
3. **Common Platform Enumerator (CPE).**[‡] This component of the protocol is concerned with the proper naming of the software and offers a hierarchical

[*] Common Vulnerabilities and Exposures (CVE) is "international in scope and free for public use, CVE is a dictionary of publicly known information security vulnerabilities and exposures." http://cve.mitre.org/.

[†] Common Configuration Enumerator (CCE) "provides unique identifiers to system configuration issues in order to facilitate fast and accurate correlation of configuration data across multiple information sources and tools." http://cce.mitre.org/.

[‡] Common Platform Enumerator (CPE) "is a structured naming scheme for information technology systems, platforms, and packages. Based upon the generic syntax for Uniform Resource Identifiers (URI), CPE includes a formal name format, a language for describing complex platforms, a method for checking names against a system, and a description format for binding text and tests to a name." http://cpe.mitre.org/.

structure. This way, software are defined explicitly, greatly facilitating software inventory management [7].

4. **Common Vulnerability Scoring System (CVSS).**[*] This is an algorithm that takes into account parameters related to the development and use of the subject software and provides a score expressing the level of calculated security [8]. The provided algorithm is available for use at no cost, and it enjoys widespread use by system designers and security analysts performing risk analysis and system planning. A CVSS calculator available online implements the developed algorithm [9].
5. **Extensible Configuration Checklist Description Format (XCCDF).**[†] This is an XML template that facilitates the preparation of standardized security guidance documents that present vulnerabilities or security concerns about software at large or about specific configurations or uses of addressed software, in a normalized "configuration content through automated security tools [10]."
6. **Open Vulnerability and Assessment Language (OVAL).**[‡] This serves as a common thread "across the entire spectrum of information security tools and services . . . (and) . . . standardizes the three main steps of the assessment process," namely, the representation of system information, the expression of the specific machine states, and finally the assessment reporting, all in a language that is common in the information systems security community. Enterprises are using OVAL in a wide variety of critical functions, including vulnerability assessment, configuration management, patch management, policy compliance, benchmark documentation, and security content automation [11].

[*] Common Vulnerability Scoring System (CVSS) "provides an open framework for communicating the characteristics and impacts of IT vulnerabilities. Its quantitative model ensures repeatable accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the scores." http://nvd.nist.gov/cvss.cfm?version=2.

[†] Extensible Configuration Checklist Description Format (XCCDF) "is a specification language for writing security checklists, benchmarks, and related kinds of documents. An XCCDF document represents a structured collection of security configuration rules for some set of target systems. The specification is designed to support information interchange, document generation, organizational and situational tailoring, automated compliance testing, and compliance scoring." http://S'CAP.nist.gov/specifications/xccdf/.

[‡] Open Vulnerability and Assessment Language (OVAL) "is international in scope and free for public use, OVAL is an information security community effort to standardize how to assess and report upon the machine state of computer systems. OVAL includes a language to encode system details, and an assortment of content repositories held throughout the community." http://oval.mitre.org/.

"The Security Content Automation Protocol (S'CAP) is a synthesis of interoperable specifications derived from community ideas."* Figure 1.4 illustrates the components that comprise the protocol.

Outside of the S'CAP, other initiatives provide standardization in additional areas, as illustrated in Figure 1.5. Among them are the Common Weakness Enumerator (CWE),† the Common Malware Enumerator (CME),‡ the Common Weakness Scoring System (CWSS),§ the Malware Attribute Enumeration and Characterization (MAEC),¶ and the Common Attack Pattern Enumeration and Classification (CAPEC).**

Collectively, the above standards comprise a very powerful infrastructure for information systems vulnerability assessment and reporting. Thus, vulnerabilities, weaknesses, and malware can be described in an accurate, standardized, quantitative, and explicit way. Using the S'CAP technology, information systems can be classified as to their level of security in a way that is standard and eventually industry-wide acceptable.

Avoiding Vulnerabilities through Secure Coding

In the software development culture, not too long ago, the design objective used to be the development of effective programs with a minimum number of lines of code or to be executable in a minimum amount of time or some combination of

* The Security Content Automation Protocol (S'CAP) "is a synthesis of interoperable specifications derived from community ideas . . . [and it] continually evolve(s) to meet the ever changing needs of the community." http://S'CAP.nist.gov/.

† Common Weakness Enumerator (CWE) "provides a unified, measurable set of software weaknesses that is enabling more effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design." http://cwe.mitre.org/.

‡ Common Malware Enumerator (CME) "was created to provide single, common identifiers to new virus threats and to the most prevalent virus threats in the wild to reduce public confusion during malware incidents." http://cme.mitre.org/.

§ Common Weakness Scoring System (CWSS), like the CVSS above, is similarly based on a comprehensive weaknesses algorithm that takes into account numerous factors that "can contribute to a vulnerability within that software." http://cwe.mitre.org/cwss/index.html.

¶ Malware Attribute Enumeration and Characterization (MAEC) "is a standardized language for encoding and communicating high-fidelity information about malware based upon attributes such as behaviors, artifacts, and attack patterns." http://maec.mitre.org/.

** Common Attack Pattern Enumeration and Classification (CAPEC) "is sponsored by the Department of Homeland Security as part of the Software Assurance strategic initiative of the National Cyber Security Division. The objective of this effort is to provide a publicly available catalog of attack patterns along with a comprehensive schema and classification taxonomy." http://capec.mitre.org/.

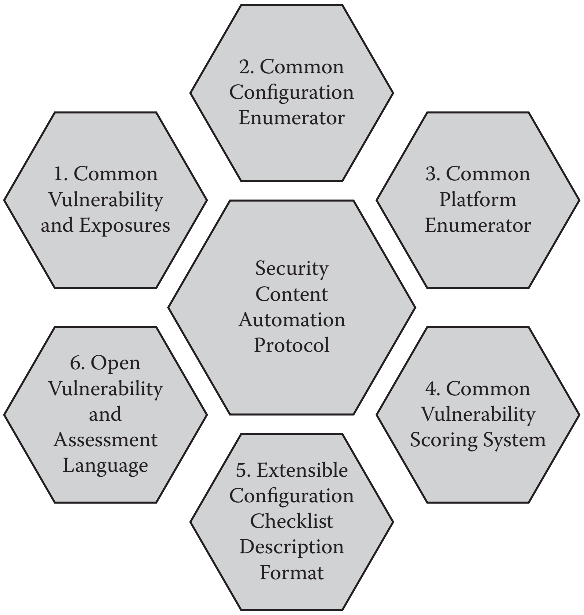

2. Common Configuration Enumerator
1. Common Vulnerability and Exposures
3. Common Platform Enumerator
Security Content Automation Protocol
6. Open Vulnerability and Assessment Language
4. Common Vulnerability Scoring System
5. Extensible Configuration Checklist Description Format

Figure 1.4 Components of S'CAP. http://scap.nist.gov/.

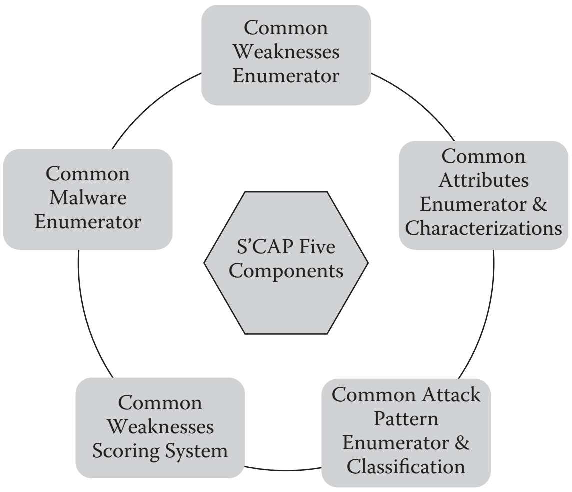

Common Weaknesses Enumerator
Common Malware Enumerator
Common Attributes Enumerator & Characterizations
S'CAP Five Components
Common Weaknesses Scoring System
Common Attack Pattern Enumerator & Classification

Figure 1.5 S'CAP five additional areas of standardization.

both. Reflecting the high cost of memory and the low speed of the processors, the valuable commodities were memory space and execution speed. Fault tolerance for data integrity was often entering the mind of the designers, but extra code for security never, since all systems were stand-alone and physically isolated from each other.

In major software designs, vulnerabilities can be minimized by segmenting the code and the data into *resident* and *transient* sections, following the principles of operating system design. That is, when an application is called, routines and data access are brought in as the immediate need calls for, rather than bringing in the entire inventory of the application. This way, should there be a malware intrusion, the damage will be contained to the downloaded (memory loaded) part of the application.

Today, generally speaking, neither memory space nor processing speed seem to be programming constraints any more. That era is irreversibly gone, being replaced by a world of high-speed total interconnectivity. The Internet offers such a high level of utility that every connectible piece of equipment, from cell phones to supercomputers, seeks to be connected. The provided level of this advanced utility and convenience results in enhanced productivity that often overshadows the accompanying security risk. Experts openly say: "The Internet is a hostile environment, so you must . . . [be able to] withstand attack[s to survive] [12]," and it is often referred to as the Wild Wild Web, WWW.

Consequently, the code that will be used on the Web has to be designed with the possibility of hostile attacks in mind, in exactly the same way that a building has to be designed and built to withstand earthquakes. The criterion as to what constitutes good code has now changed, from minimal code to minimally vulnerable [13].

There is a Greek popular proverb that advises, "It is better to spend your time securing your donkey onto a tree than spending it searching for the lost donkey." This directly applies to the design of Internet-exposed software. It is better to spend our time designing secure software rather than facing the consequences of intrusions—costly development of patches, bad publicity, etc. In other words, today, the concepts of software quality and software security are embedded within each other. Thus, if the software specifications call for protection, a corresponding security mechanism has to be in place. An increasing trend is for web applications to come with their own firewalls rather than exclusively relying on those of the hosting system.

The cost of fixing a vulnerability is usually very high if one considers the intangible damage that is being produced. During the fixing, resources will have to be taken away from other assignments to attend to this matter in a very urgent way. In monetary terms, the cost of fixing a vulnerability is a five- to seven-digit dollar figure, depending on how deep in the design the vulnerability exists. The necessary procedure to rectify a vulnerability typically takes the following steps, also illustrated in Figure 1.6.

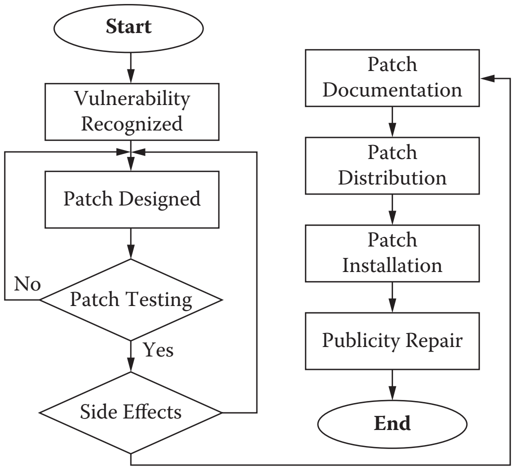


Figure 1.6 Steps necessary to rectify a software vulnerability.

1. Location of the origin of the vulnerability
2. Design of a patch that will strengthen the code and eliminate the vulnerability
3. Application and testing of the patch
4. Confirmation that there are no side effects
5. Drafting of patch documentation
6. Preparation of a patch distribution plan to all affected clients
7. Patch installation
8. Confirmation of the effectiveness of the patch; public relations campaign to offset prior negative publicity

Unless the origin of the vulnerability is explicitly identified, any patch may cover a problem without eliminating the root of the vulnerability. The most frequent vulnerabilities caused by insecure code are

- **Buffer overflow.** Program activities, especially inputs, allocate a certain amount of finite space for the new, created data to be stored. Excessive amount of data fills up the buffer, and the program will crash, unless the program has preventive measures [14].
- **Arithmetic overflow.** This usually occurs when there is an accumulation, and the accumulator's maximum number is exceeded. The next amount to be added *rolls over* the accumulator, creating a false result unless, again, preventive measures are incorporated [15].
- **Format string attacks.** Insecure code allows user-provided input to be treated as a command when it can only be data. If permitted, attackers can install their own executable code [16].

- **Command injection.** Lack of input validation will permit disallowed commands from being accepted and executed [17].
- **Cross-site scripting (XSS).** This is injection of malicious code into a client software, usually to bypass access controls [18].
- **SQL injections.** This is an SQL code that misleads an application into executing it [19].
- **Insecure direct object reference.** This applies to a webpage retrieval. Rather than being referenced by its HTML file name, it is referenced by the direct object definition that the page has in a database. This reveals the identity of the database, and an attacker may use it to retrieve other files [20].
- **Insecure storage.** This is storage of critical files without encryption or read/write protection. Poor examples of steganography fall in this category [21].
- **Weak cryptography.** This is file encryption that can be easily deciphered [22].
- **Race conditions.** This occurs when one resource is granted to a second user before the first has finished [23].

Software security must originate in the product specifications. It is not a step in the software development, but it is a solution that has to be interweaved with the functional design and code writing. That is, the software designer's mindset must have two parallel tracks—functionality and security—with security being a mechanism that reveals and subsequently blocks intrusions. Such security mechanism should be robust and well designed and should never be replaced by some data *obscurity scheme* that attempts to *outsmart* the attacker.

Besides thinking of the possible attacker, the designer must also think of the user convenience and provide a security mechanism that poses no hindrance to the normal operations, being as transparent to the user as possible.

Mistakes Can Be Good

No matter where, mistakes are always viewed negatively and are identified as moments of failure. This is a wrong attitude. When Thomas Edison was told that he failed one hundred times before perfecting the incandescent lamp, his response was that, since he learned something new from each of the one hundred times, his success process simply took one hundred steps. Of course, making the same mistake again and again is not a sign of wisdom. The point to make here is that there should be no fear of making mistakes, as long as they are made in good faith, do not lead to catastrophes, and have, hopefully, added some new knowledge.

It would be a mistake to extend an application, or a user for that matter, more security privileges than needed to perform an assigned task. When applications are being installed, all needed resources must be listed with the required privileges, such as read, write, create, delete. This way, should a malware install itself in the application, the potential damage will be minimal, and the malware will not be able to roam inside the system.

Similarly, when a computer is a single user, it should not necessarily operate in the *administrator mode*, but in the *user mode*. This way, should a malware manifest itself in the *user mode*, it will not propagate into the *administrator mode* privileged areas. By having privileged layers—*user, super-user, admin, super-admin*—malware is contained to a certain extent.

Threats Classification

Basically, there are three types of threats, namely, illegal data change, illegal data access, and illegal obstruction to data access, as illustrated in Figure 1.7.

The information security researchers at Microsoft broadly classified all threats in six categories, the acronym of which conveniently make up the word STRIDE (spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege) [24]. Table 1.1 lists the six threat categories and the associated characteristics.

Threat Modeling Process

In information systems, threats—potential attacks that may capitalize on a system's vulnerabilities—originate in a variety of sources and bear various levels of gravity. There is a need for a structured approach toward defining and dealing with threats. Below is a sequence of steps that can serve as a mechanism for that purpose.

1. Define what constitutes the information system in question. That is, determine the functional and geographical borders of what we refer to as being our system. Beyond what point is it not our responsibility or liability?
2. Now that we have quantified our system, we attempt to identify what we consider as being the threats that are based on internal or external vulnerabilities. The STRIDE categories, mentioned previously, can serve as a starting point to threats classification.

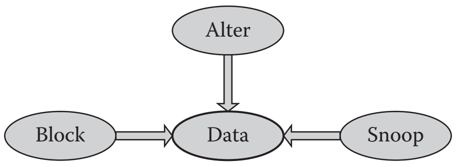


Figure 1.7 Three major types of threats to data.

Table 1.1 Information Systems Threats

Categories Threats	*Definition of Threat*	*Affected Security Property*	*Possible Solution: Enhanced Mechanism for . . .*
Spoofing	Presentation of false identity	Authentication user identity confirmation	User recognition
Tampering	Modification of content (code, data, or process)	Code, data, or process integrity	User access
Repudiation	Denying performance of documented past action	Absolute verification of past action	Activities fogging
Information disclosure	Unauthorized access to content (code, data, or process)	Data confidentiality	Data fencing
Denial of Service	Partial or total incapacitation of a resource	System accessibility and availability	Packet filtering
Elevation of privilege	Unauthorized upgrading to a higher level of access	Allocation of access authorization	Level recognition

3. Recognize which of the threats, in your context of operations, constitute absolute danger that may lead to a catastrophic impact on the system. Define the modes under which such threats can become successful attacks.
4. Develop defense options for each and every recognized threat, and rank the considered defense mechanisms based on effectiveness and demand of resources.
5. Select optimum approaches to threat eliminations, balancing effectiveness, probability of occurrence, severity of occurrence, and solution development cost.

This is an iterative process requiring the participation of all who are associated with the secure performance of the system, namely, analysts, designers, programmers, users, trainers, sales, and evaluators, as illustrated in Figure 1.8. This process needs to be reviewed at certain time intervals or when new threats are known to have surfaced.

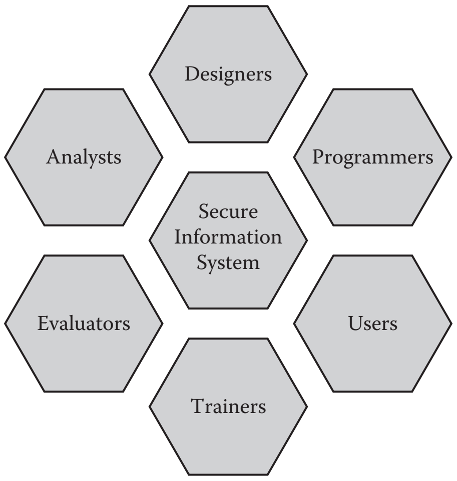


Figure 1.8 Participants in the development of a secure information system.

Security Starts at Home

Of course, home is the creative phases of the Software Development Life Cycle (SDLC), especially the program design and coding stages. SDLC is often described as consisting of seven basic transitional stages. Namely,

1. From abstract needs to formal requirements—Analysis
2. From formal requirements to overall design—Design
3. From overall design to program code
4. From all of the above to documentation drafting
5. From all of the above to system testing
6. From system testing to system utilization (or to a previous step)
7. From system utilization to needs enhancement (Step 1)

During the analysis phase the *security requirements* are being defined, while in the design phase the *security mechanisms* are being developed. Later, in the testing the vulnerabilities are hopefully discovered and eliminated.

Vulnerabilities elimination usually takes one of two forms. One is removal of the causes of the vulnerability, and the other is removal of the effects of the vulnerability. Depending on the prevailing rationale, normally one of the two solutions is implemented.

It is now a standard practice and a legal requirement in all major construction sites to have safety engineers overseeing all activities for the workers' safety. Similarly, in information systems development, software security professionals must oversee the development process from a to z and perform the following [25]:

- Identify common programming errors that lead to vulnerabilities.
- Establish standard secure coding practices.
- Educate software developers (/designers).

Furthermore, intra-organizational software development initiatives may include

- Programmers certification in secure coding
- Software certification as to secure coding
- Utilization of software analysis tools

Software tools are very important in the development of secure and well-documented code and can be classified in the following four categories [26]:

- *Code Coverage*—Keeps track of the code and data locations that have been created, read, or modified.
- *Instruction Trace*—Records the execution of each instruction, making it available for subsequent step-by-step analysis.
- *Memory Analysis*—Keeps track of the memory space utilization, looking for possible violations.
- *Performance Analysis*—Based on the user's criteria, software are analyzed and fine-tuned for performance optimization.

This way, information system applications will be secure as well as reliable. The Software Engineering Institute at the Carnegie Mellon University is leading in the development of standards, tools, and training aiming at the design of reliable and secure information systems [27–30].

The major problem with software development is their life cycle economics and logistics, where very often the prerelease investment does not match the postrelease revenues. Often, the idea-to-product-release-time is minimized, frequently resulting in the release of nonsecure software.

To the rescue of fast software development have come the object-oriented languages and the structured programming concepts that underlie the design of modern information systems. Yet, *secure software by design* still constitutes a new principle preaching that security must be an integral part of functionality.

Security in Applications

Vulnerabilities also come with insecure applications. Microsoft Word 2003 is a typical example. Its defect is that, upon requesting a *save*, the previously deleted text, though not appearing in the final document anymore, still remains part of the file. When the Word file is opened in the Notepad, deleted text can be clearly seen, while in the Word document it is not part of the final visible text.

This vulnerability violates the user's privacy, and implied confidentiality, without offering any added functionality. In this case, the only solution is, at the end of the document preparation, to *select all*, *copy* it, and *paste* it in a brand new document.

However, the average users neither suspect the existence of vulnerabilities, especially with big name products, nor do they have the technical skills to discover and rectify software design defects. It is the responsibility of the organization's information security officers to be knowledgeable of the existence of defects in software that they approve for use and to advise users accordingly.

The United States Department of Homeland Security (US-DHS) maintains an extensive Computer Emergency Readiness Team (US-CERT) that warns the public of the existence of vulnerabilities in software that are widely used and that maintains online a *Vulnerability Notes Database* [31].

Vulnerability Note VU#446012, for example, identifies and describes a "Microsoft Word object pointer memory corruption vulnerability [32]." The Note points to Microsoft's own Security Bulletin that states: "When a user opens a specially crafted Word document using a malformed object pointer, it may corrupt system memory in such a way that an attacker could execute arbitrary [malware] code [33]." The Note continues that "Office documents can contain embedded objects" (that may contain malware).

It is, therefore, advisable prior to using an application to visit this database and become aware of the presence of any vulnerabilities of interest. Sorting the above-mentioned database [34] by *Severity Metric,* one may recognize that appearing on the top are vulnerabilities that are caused by buffer overflow. It is consequently imperative that buffers be of dynamic size, as well as self-cleaning of the obsolete data.

A most common gateway to malware attacks is the web browser. On one hand, it has to be open and powerful to facilitate the display of data; on the other hand, it has to be configured to protect the host computer. Security experts believe that "Many web browsers are configured to provide increased functionality at the cost of decreased security." An excellent guide to *Securing Your Web Browser* is available online that can benefit novices as well as experts [35].

International Awareness

Similar to the US-CERT, other countries concerned with the safety of cyberspace have established similar government agencies for that purpose. Table 1.2 provides a partial list of foreign government agencies with cybersecurity responsibilities.

Table 1.2 Foreign Government Cybersecurity Agencies

Country	*Agency Name and URL*
Australia	The Australian Government Computer Emergency Readiness Team http://www.ag.gov.au/www/agd/agd.nsf/page/GovCERT
Canada	The Canadian Cyber Incident Response Centre (CCIRC) http://www.publicsafety.gc.ca/prg/em/ccirc/abo-eng.aspx
European Union	The European Network and Information Security Agency (ENISA) http://www.enisa.europa.eu/
France	French Networks and Information Security Agency (FNISA) http://www.ssi.gouv.fr
Germany	German Federal Network Agency, the Bundesnetzagentur (BnetzA) http://www.bundesnetzagentur.de
New Zealand	The Centre for Critical Infrastructure Protection (CCIP) http://www.ccip.govt.nz/
United Kingdom	The Centre for the Protection of National Infrastructure (CPNI) http://www.cpni.gov.uk/

Exercises

1. Review paper "Common Control System Vulnerability" at http://www.us-cert.gov/control_systems/pdf/csvul1105.pdf (7 pp.), and present your summary in a two-page report.
2. Review paper "Vulnerability Analysis of Certificate Validation Systems" at http://66.135.41.137/about/library/whitepapers/w06-02v1-vulnerability_analysis_of_validation_systems.pdf, and present your summary in a two-page report.
3. Review paper "System Vulnerability Mitigation" at http://www.sans.org/reading_room/whitepapers/awareness/system-vulnerability-mitigation_1339 (19 pp.), and present your summary in a two-page report.
4. Open 2003 MS Word and create a file with only the words "Good morning." Delete everything and write "Good night." Save the file under the name Good.doc. Open the file using the Notepad. View the lower part of the code. You will see the deleted words "Good morning." What conclusions do you

draw? Visit the referenced United Nations report http://www.aquol.com/archives/2005/10/redactions_on_h.php that shows how such vulnerability created a diplomatic incident.

5. In a two-page report, explain the buffer overflow vulnerability. Cite five references—news releases—where this has been identified as the cause of denial of service.
6. Visit the websites of Table 1.2, and research toward finding more such agencies and their policies.
7. Identify the preferred performance of residential alarm systems, and develop the formal (quantified) requirements.
8. Select two web browsers and make a comparative study of their security features.
9. Research the use of the terms *administrator* and *super administrator.*
10. Research the use of the terms *user* and *super user.*