Global Information Assurance Certification Paper

Critical System Lifecycle: A Security Perspective

Geoffrey A. Pascoe
SANS Institute
Submitted in Consideration for GSEC Certification
(Version 1.3)
29 March 2002

Abstract

IT security threats are becoming more serious and more numerous. As a consequenc e, the frequency of security patches, antiviral software updates, and new virus signatures is increasing. This is particularly problematic for critical systems whose failure can result in the disruption of essential services, large financial losses, or lo ss of life. Applying patches and updates to critical systems without thorough testing can result in software defects that cause these systems to fail. But not applying updates and patches in a timely fashion can result in security failures that are equal ly serious. By addressing security in the way products are developed, deployed, supported, serviced, and retired, software vendors can make a significant contribution to the reliability and security of critical systems.

Keywords: *Computer Security, Critic al Systems, Product Lifecycle, Software Development Processes, Reliability*

1 Introduction

Critical systems are information technology solutions whose function is fundamental to an organization's proper functioning. Failure of these systems could cause irrep arable harm. An important subset of these types of systems are special purpose, often turnkey, systems such as ERP for business operations, financial systems for banking and securities trading, and medical systems for the diagnosis, monitoring, and treatm ent of patients.

There are many ways in which IT systems can fail:

Physical Failure

- A hardware malfunction makes the system unavailable.
- A natural disaster results in the loss of power, communication, or physical destruction of the hosting facility.
- An employee inadvertently removes power from a critical server. Loss of power would result in loss of function and possible data corruption.

Operational Failure

- Backup policies are not followed or backup integrity is not verified. When restoration is needed, i t is not possible.
- Inadequate system monitoring results in performance degradation or exhausted storage.
- Failure to follow operational policy results in the miscategorization or deletion of critical data.

Quality Failure

- Inadequate software testing causes unreliable system behavior and crashes.
- The installation and integration of the system into the organization is not adequately tested. Data communicated between peer systems in the organization is lost or corrupted.

Security Failure

- The platform on which the system is built has a trap door that is subsequently exploited by a cracker. The cracker steals sensitive business information.
- A cracker chooses the system for the target of a denial of service attack. The system cannot be used for its intended purp ose.
- A hostile, or defective, worm infects the system causing the system to crash.
- A user, frustrated with restrictive password selection policies, writes their password on a piece of paper taped underneath their keyboard. A visitor opportunistically exploits the password to compromise the system for a subsequent network attack.

(Neumann) describes a wide variety of computer failures from the mid -1980's and (Levenson) describes a particularly egregious quality failure in a medical system, resulting in the injury or death of several patients.

Different types of failures are not independent —one type of failure can lead to another type and steps taken to prevent or mitigate one type of failure may have a negative consequence on the ability to prevent or mitiga te another type of failure. An important dependency is the relationship between security and quality failures. While many security failures can also be considered quality failures, sometimes an attempt to correct a potential security failure (i.e., a vul nerability) may lead to a quality failure.

System Patches

It is generally good policy to quickly patch systems when new security vulnerabilities are discovered (Lucero). Unfortunately, if patches are installed without adequate testing, the system could fa ll victim to a quality failure. Conversely, not patching the system in a timely fashion in order to perform more thorough testing could lead to a security failure. It has been reported that, because of the volume patches released by software platform vendors, administrators have been hard pressed to simply apply the required security patches in a timely fashion (Verton) —thorough testing of these patches on critical systems is far more time consuming.

Antiviral Software

A common measure for preventing cert ain types of security failures is the installation of commercial antiviral software ("VirusScan")("Norton"). One strategy used by antiviral software is the detection of infections by scanning the system for signatures of known malicious software using pre -determined virus definitions (a.k.a., virus signatures). Both antiviral software and virus signatures require periodic updating, and virus signatures require very frequent updates —some organizations mandate automatic daily virus signature updates. Simil ar to system security patches, untested software in the presence of

antiviral software updates is a significant risk to a critical system. Therefore, the effectiveness of antiviral software may be limited, or its simple presence not tolerable. [1]

Preventing failures includes, but is not exclusively concerned with security. An important goal of system administrators and software designers is to minimize system failure, security or otherwise. Any realistic approach to system security needs to balance these, sometimes, competing forces.

2 Development Processes

The causes of security vulnerabilities are numerous ranging from inadequate assessment of risks and non-existent organizational security policies to poor operational compliance with those security policies. However, one major cause is simply defects in software systems, resulting from the way they are specified, constructed, and tested (Ghosh). In the development of a product, security is closely related to quality. But, instead of being concerned with how the system behaves under normal operation, where failure may be due to mischance, developers must be concerned with failure due to an intelligent and malicious agent attempting to force failures. Ross Anderson refers to these two scenarios as "programming Murphy's computer" and "programming Satan's computer," respectively (Anderson, Satan)(Anderson, Security). Murphy is bad luck, whereas Satan is actively trying to get you.

For the most part, IT security has focused on measures taken in deploying organizations:

> *The types of approaches and point solutions advocated by computer security professionals to date have aimed at system administrators, chief information officers, and other personnel involved in system infrastructure management. These solutions usually focus on addressing an enterprise's security defenses (such as firewalls, routers, server configuration passwords, and encryption) rather than on one of the key underlying causes of security problems—bad software. (Ghosh 14)*

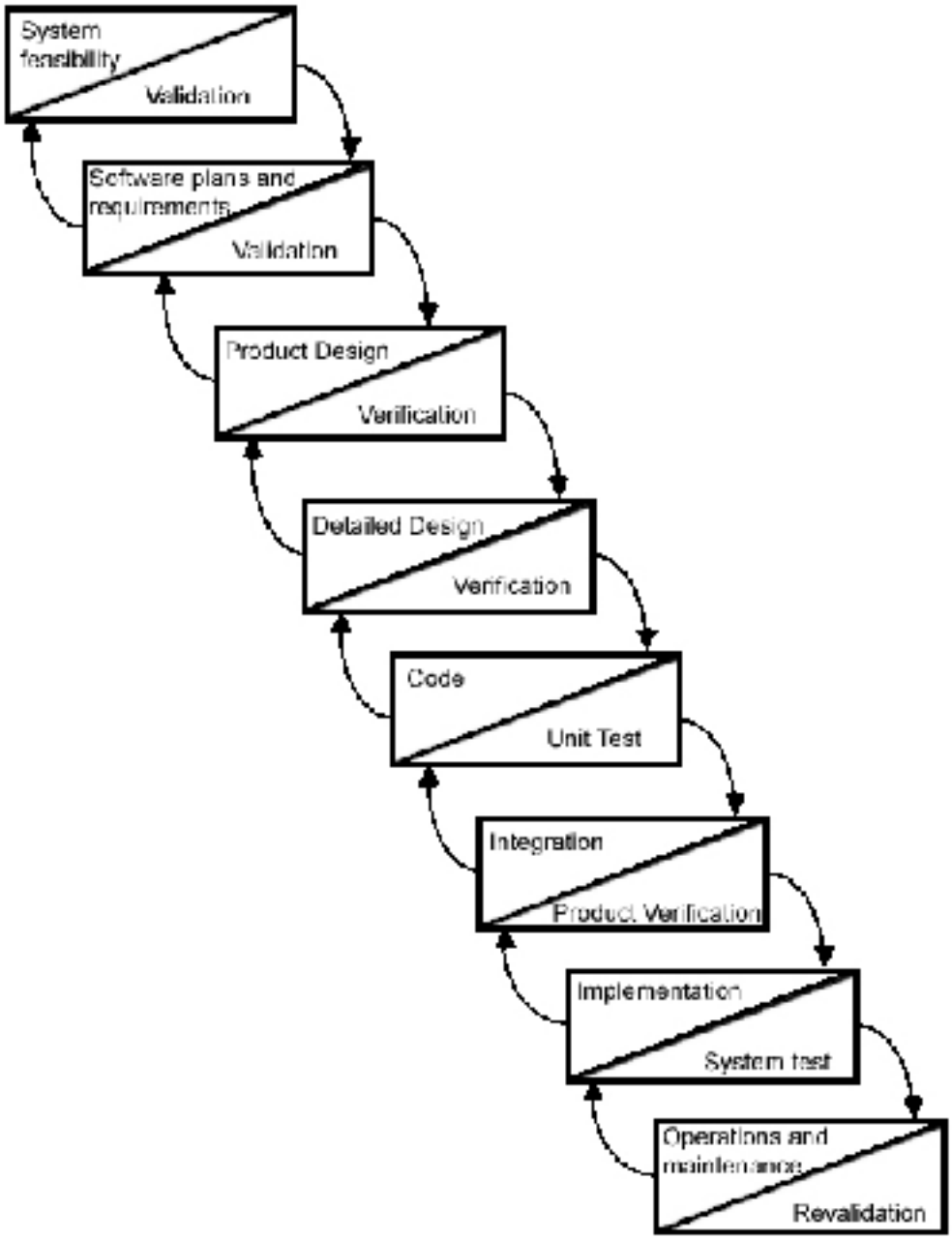
To provide security in-depth for critical systems, vendors need to build systems that are higher quality and more secure. By doing so, they also will create systems that require less frequent patching, avoiding the "penetrate-and-patch" cycle (Gosh 15)—the need for frequent patches creates an undesirable tension between the need to fully test systems and the need to close security vulnerabilities as soon as possible.

To achieve this goal developers must extend the concept of software quality to incorporate a concern for security. Quality software systems are the result of a quality development process. It therefore follows that improvement in the security of a system can be addressed in the processes that are used to develop systems.

It is beyond the scope of this paper to provide a comprehensive overview of software development processes. Traditionally, the waterfall model (Boehm, Software) has been used (see Figure 1). In the waterfall model, development cascades (as if in a waterfall) from early phases of exploration and requirements analysis through implementation and operation, with the ability to return to previous phases if deemed necessary.

Figure 1—The Waterfall Model of the Software Life Cycle
(Source: Software Engineering Economics—Boehm)

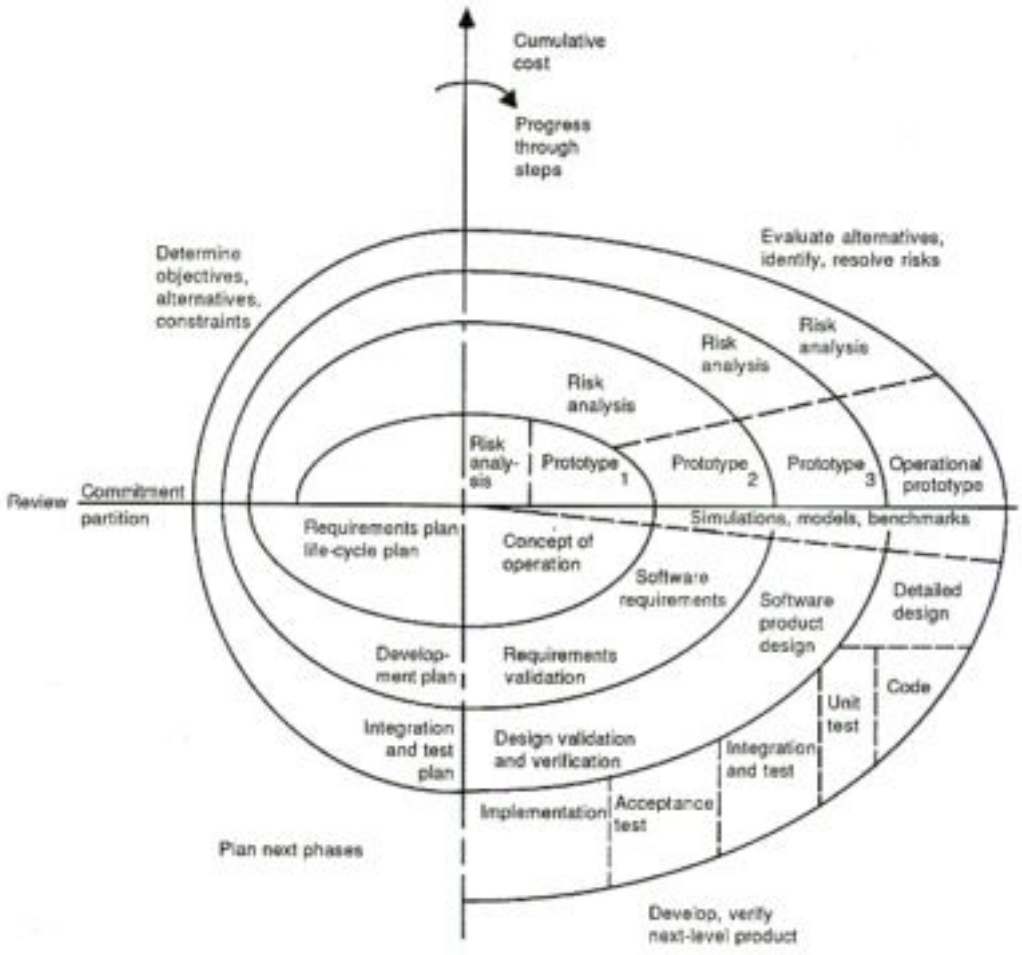
[1] Some systems, particularly real-time systems, do not tolerate the presence of antiviral software that scans persistent storage looking for infections. Data can be lost because an intensive scan interferes with the system's ability to respond in real-time.



Unfortunately, the waterfall model suffers from a number of deficiencies. One serious problem is that developers, users, and managers, have few indications of problems early in the development process. They do not see the system fun ctioning at all until very late. The spiral model (Boehm, Model)(Boehm, Development) was developed to address the deficiencies of the waterfall model (see Figure 2). In the spiral model, development consists of a series of develop ment cycles, which are repeated. Each cycle results in a working system, that provides an opportunity for the stakeholders to evaluate and plan the next cycle. What makes the spiral model fundamentally different from the waterfall model is that the water fall model assumes that previous phases of development are complete before proceeding with later phases. The spiral model recognizes that complete

specification and knowledge at any phase is not feasible. Instead, the developers incrementally refine the system, gaining more knowledge and confidence with each subsequent cycle.

Figure 2—Spiral Model of the Software Process
(Source: A Spiral Model of Software Development and Enhancement —Boehm)



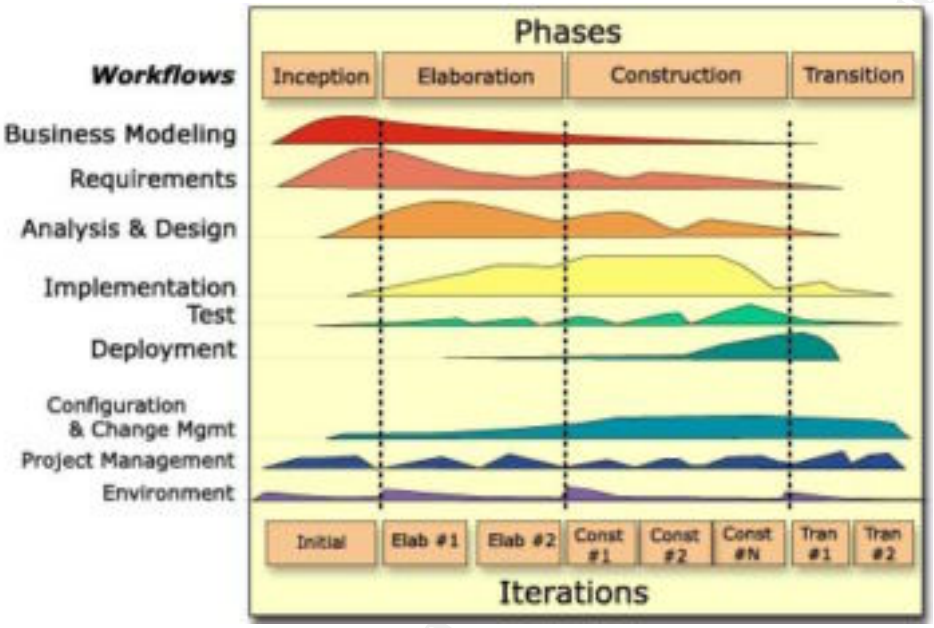
Recently the Unified Software Develop ment Process (Jacobson)("Rational")(Kruchten), a spiral-like process, has been gaining acceptance. In the Unified Software Development Process (also known as the "Rational Unified Process", or "RUP") each product release cycle consists of four phases: inc eption, elaboration, construction, and transition. Each phase may involve a series of iterations, and each iteration includes a set of activities; the RUP calls these activities "core workflows". The RUP defines five core workflows [2]:

- Requirements

[2] Figure 3 shows different workflows than that in the text, but the definitive RUP source (Jacob son 11) identifies these five. The author's opinion is that these five workflows are invariant across almost all development projects, whereas the other workflows in Figure 3 from (Kruchten) provide a useful, but expanded view. No t all development projects will do business modeling or be involved in deployment, and the other supporting workflows are only an adjunct to the main development activities.

- Analysis
- Design
- Implementation
- Test

Figure 3—Phases of the Rational Unified Process
(Source: What is the Rational Unified Process? —Kruchten)



Other development processes define similar activities. But, regardless of the particular development process, the natural question to ask is "What should be done in each development activity to help enhance the security of the system to be developed?"

2.1 Risk Analysis

One of the goals of spiral software development processes is the management of risk . Boehm defines risk this way:

> ***Risks*** *are situations or possible events that can cause a project to fail to meet its goals. They range in impact from trivial to fatal and in likelihood from certain to improbable. A risk management plan enumerates the ris ks and prioritizes them in degree of importance, as measured by a combination of impact and likelihood of each. For instance, the risk that technology is unready may be mitigated by an appropriate prototype implementation in an early cycle (Boehm, Developm ent 3).*

This view of risk is very expansive, going beyond security or quality, but encompassing them. When thinking about security the concept of **impact** can be viewed as the consequences of security compromise with respect to confidentiality, availability , and integrity; the concept of **likelihood** can be viewed as the probability of a successful attack, which is a function of the likely threats, the environment within which the system will be deployed, and the inherent vulnerabilities of the system itself.

What risks are considered acceptable is a policy decision, but generally speaking, high-severity/high-probability risks are considered unacceptable, whereas low-severity/low-probability risks are acceptable. Risk can be described by the following equation:

$$Risk = probability \cdot severity$$

It is beyond the scope of this paper to detail risk analysis methodologies and techniques — however (Trice) provides an example for the use of Failure Mode Effect Analysis (FMEA).

Table 1—Example FMEA Frequency/Impact Matrix

	Negligible Impact	**Low Impact**	**Medium Impact**	**High Impact**
Incredible	1	1	2	3
Remote	1	2	3	4
Occasional	2	3	4	5
Probable	3	4	5	6
Frequent	4	5	6	6

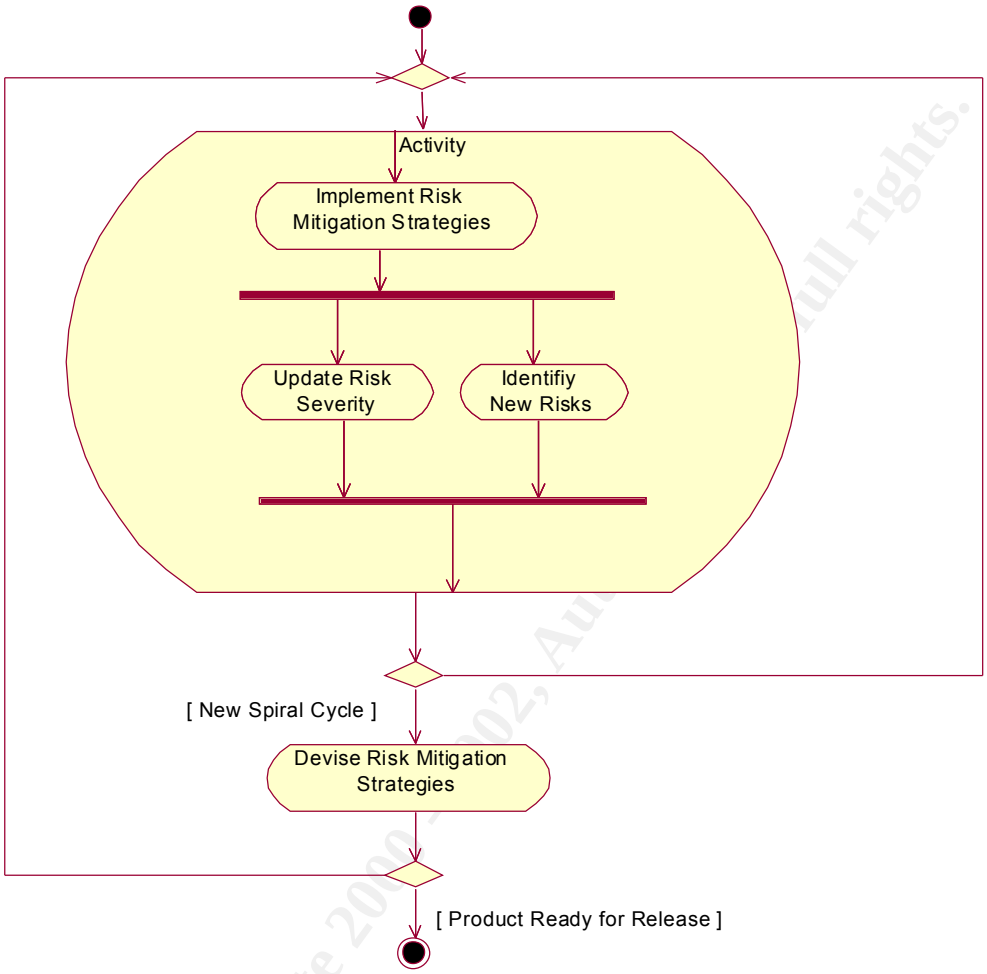
Table 1 shows an example of a FMEA Frequency/Impact matrix, where risks are categorized in bins that can be assigned descriptions, if desired:

- Category 1 is a Negligible Risk
- Category 2 is an Acceptable Risk
- Category 3 is an Undesirable Risk
- Category 4 is a Tolerable Risk
- Category 5 is a Significant Risk
- Category 6 is an Intolerable Risk

The number of impact columns and frequency rows can be adjusted to suit a project's needs. Each risk is categorized and specific actions should be outlined for different categories of risks.

The spiral model shows risk analysis performed once each cycle (see Figure 2). However, the character of security risks understood, or introduced, in each development activity varies widely (e.g., an implementation vulnerability has a different character than one resulting from incomplete requirements analysis or specification). As a result, it is the author's opinion that risk analysis should be performed during each activity in a cycle, with risk mitigation strategies designed once per cycle (see Figure 4). Risk analysis is transitive across all activities, though the emphasis and focus of risk analysis will differ in each activity.

Figure 4—Proposed Risk Analysis Process



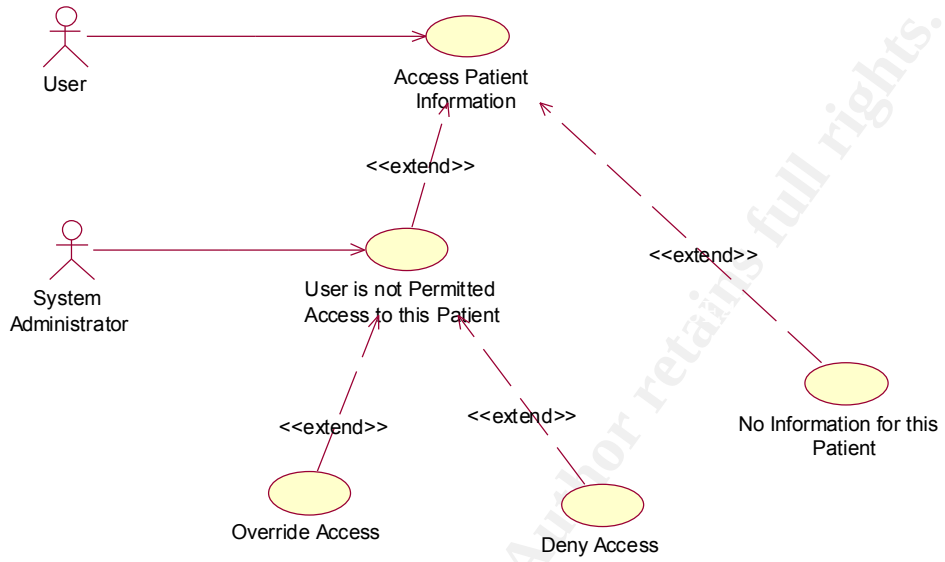
2.2 Requirements

The requirements activity identifies what the system must do and how it should behave. Requirements can be characterized as functional or non-functional. Functional requirements are requirements about the useful characteristics of a system (e.g., the system allows a user to delete accounts, or the user logs into the system). Non-functional requirements are all other requirements (e.g., calculation of the balance sheet must take no more than one second, or the system must not allow unauthorized users to access identifying information of the patient).

There are many ways of structuring requirements. The Unified Process employs use cases and use case diagrams to describe functional requirements —a use case is a functional requirement (Jacobson 33-58)(Booch 219-241)(Rumbaugh 63-66)(Kulak). Typically, non-functional requirements are attached to use cases. Security requirements may be functional or non-functional. For example, a use case for authenticating a user is a security-related requirement, but also represents functionality. The need to transfer money from an external system is a functional requirement, but the security requirement that the transfer

be properly authenticated and the communication encrypted is a non-functional requirement.

Figure 5—Example Use Case Diagram for Access Control



Confidentiality-Availability-Integrity

Confidentiality, availability, and integrity are the three bedrock principles of security (Northcutt); all must be considered and a weakness in one area can undermine another [3].

Detailed security requirements should be driven by a statement of the overall security policy goals and requirements, as well as assumptions about the security environment within which the system will be deployed (e.g., developers might assume that the intranet on which the system is deployed will be protected from the internet by a firewall which filters for IP spoofing attacks). Developers should consider what information should be protected, the consequences of non-availability of the system, and the impact of compromised integrity to the operation of the system or the data it manages.

This approach can be systematically applied to detailed security requirements by considering confidentiality, availability, and integrity for each functional requirement. This enforces a discipline of thinking about what risks exist and what measures can be taken in the functionality of the product to strengthen the system's security [4].

Risk Analysis in the Requirements Activity

Risk analysis in the requirements activity should take every non-functional requirement and categorize it according to the project's frequency/impact matrix. Since the structure of the system may not yet be known, estimating the frequency of a successful compromise cannot take into account the inherent vulnerabilities in the system, but an understanding of

[3] For an example of how compromising one principle can weaken another see (Shimomu ra).
[4] The Common Criteria ("Common")(Knight) also provides useful, and much more extensive, framework for specifying and evaluating security requirements. These are referred to as "functional requirements" and "assurance requirements" in parts two and th ree of the specification, respectively. The recommendation presented here, and the structuring of security requirements used by the Common Criteria, are not mutually exclusive.

the likely threats and deployment environment should be sufficient. Impact should be determined from the system's security goals.

2.3 Analysis

The purpose of Analysis is to gain a better understanding of the system that is to be built. Through analysis the developers of a system gain understanding of the system at the conceptual level. Analysis uses more formal modeling techniques than use cases, helps refine the requirements, and bridges the gap between the requirements and design.

From a security perspective, analysis provides a means to better understand the security requirements and discover potential vulnerabilities. For example, a system may require role-based access controls. By conceptually modeling the relationships between users, roles, and the types of access permitted (see Figure 6), the developers can understand what roles permit access to what information and under what conditions. Developers can ask and answer questions: Can a user play more than one role simultaneously? What happens if a user's access permissions conflicts with the user's current role?

Analysis can also help developers understand the dynamics of a system. If the system interacts with another system, thorough modeling may uncover opportunities for spoofing attacks or other security vulnerabilities.

Some risk mitigation strategies may be implemented during analysis. For example, the simple analysis shown in Figure 6 shows access rights that allow a user to change or review patient information at the granularity of a patient. However, the system's security policies and goals may require that patient information be protected at a finer granularity because some information may be more sensitive than others (e.g., the result of an AIDS test).

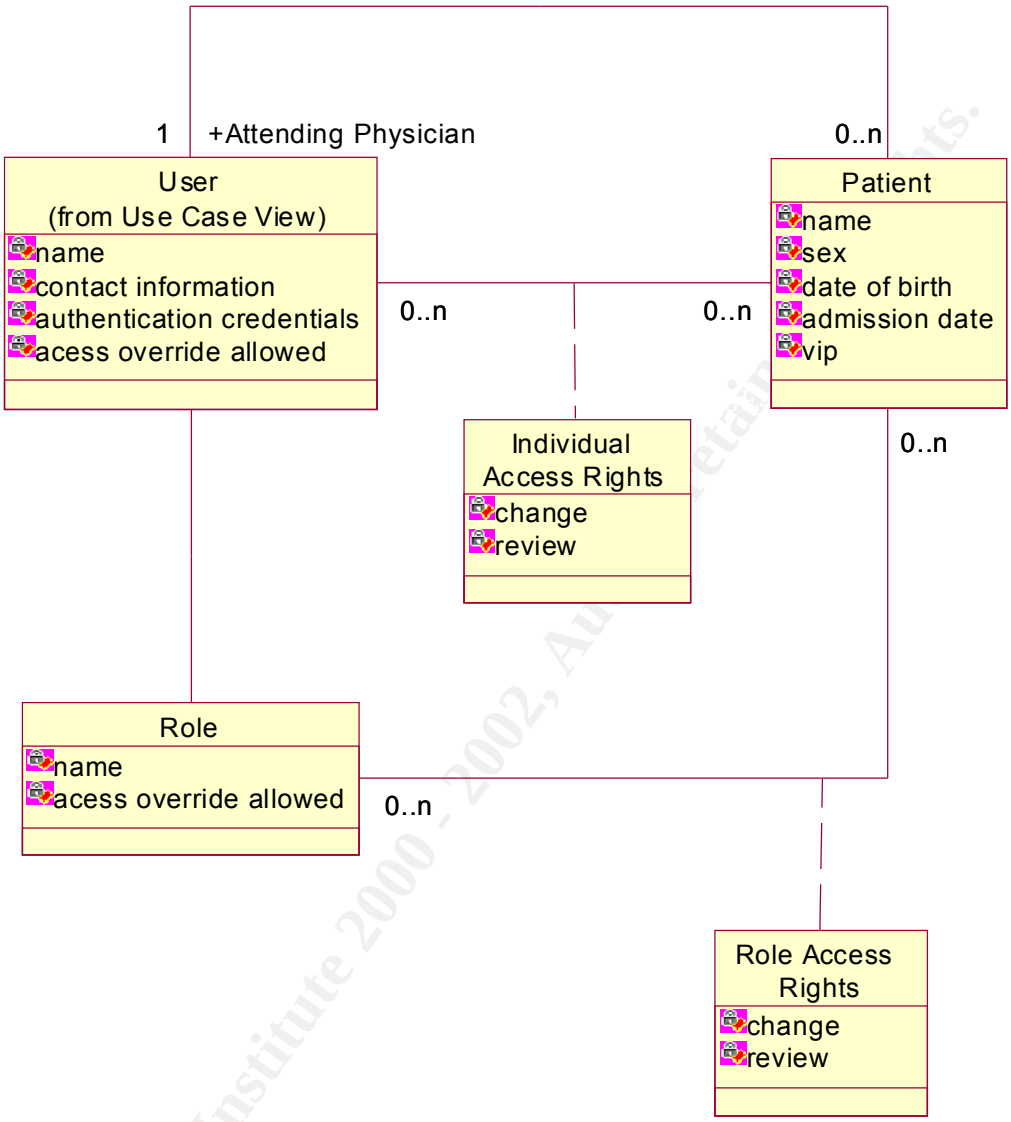
Risk Analysis in the Analysis Activity

Revisit all risks and update their severity category and use the analysis model to look for new risks in the system's specification by identifying logical holes and inconsistencies that may be exploited by an attacker.

2.4 Design

Design determines the structure and dynamics of the system to be built. It differs from analysis in that the system is decomposed into subsystems or components, reflecting the technological constraints of the hardware, software platform, and tools with which the system is to be built. Analysis is conceptual and abstract, whereas design is concrete. The design is the blueprint for those implementing the system.

Figure 6—Example Class Diagram for Access Control



The design may implement risk mitigation strategies. However, the design activity can expose new security risks that cannot be discovered from the requirements or analysis activities. For example, the design migh t partition the system into a set of distributed components and the communication between these components might expose a potential vulnerability. Or, the design might leverage pre -existing components, which might have their own vulnerabilities.

Because design vulnerabilities might not be associated with any one requirement (e.g., a single component may fulfill a wide variety of use cases), confidentiality -availability-integrity vulnerabilities should be associated with components, subsystems, or the interfaces between them. Risk analysis can be an extension of the risk analysis started during the requirements activity.

Risk Analysis in the Design Activity

Update the severity category of risks that have been mitigated during design. Look for new risks: external interfaces, third party components, and platform vulnerabilities. Because the structure and dynamics of the system are defined, it is now possible to look for vulnerabilities by considering likely vectors of attack (e.g., network communications, user authentication, file system, trust relationships, etc.)

2.5 Implementation

Implementation is concerned with the construction of the system. The developers code and unit test the system in implementation. Just as the design activity can result in security threats that cannot be discovered in requirements and analysis activities, implementation defects have a unique character that requires special attention. It is impossible to enumerate all possible implementation defects, but some common implementation defects include (Raynal)(Wheeler):

Failure to adhere to the design

If the implementation does not adhere to the design then the security analysis done during the design activity will not be accurate.

Improper error detection and handling

If the system does not detect or properly handle error conditions, then the system will arrive at an unknown state, making it vulnerable to attack.

Buffer overflows

Buffer overflow is a common defect, which can easily be exploited when the buffer is associated with the receipt of network traffic. However, buffer overflows anywhere in a system can cause unpredictable behavior, leading to a potential denial of service attack, or more direct penetration.

Incorrect input validation

If input data is not properly validated then the system can arrive at an unknown state, making it vulnerable to attack. This may be considered a subclass of improper error detection that is relatively easy for an attacker to exploit.

Uninitialized variables

Use of uninitialized variables can cause unpredictable behavior. Uninitialized variables can cause crashes resulting in denial of service attacks, but they also provide a determined attacker a means for getting the system to use a variable that is "initialized" by the attacker. For example, if a routine uses an uninitialized stack variable, then by writing to the stack attackers can initialize that variable to any value they choose.

Format String Attacks

String formatting of the type that became common with the advent of the C language is enormously powerful because it allows the use of arbitrary format strings to interpret data. These format strings are also data, so if a defect exists that allows an attacker to input strings that are used as format strings, the attacker can manipulate the behavior of a program in a very flexible way (Newsham).

Race Conditions

Race conditions occur when a system performs concurrent activities and these activities are not adequately synchronized. An attacker can exploit race conditions because the order of operations on common resources can lead to unpredictable, or unforeseen states. A race condition vulnerability often refers to a condition where a critical piece of sequential code accesses a resource that is not locked; an intruder acting on a separate thread t hen changes the resource without the code's knowledge (e.g., an intruder might change the permissions on file, or replace the file, in between the time the file is initially examined and the time it is used).

Security conscious implementation is analogous to multithreaded programming—it requires a specific mindset. It is difficult to retrofit a system to be thread -safe, just as it is difficult to retrofit security. (Raynal) describes security -related implementation defects from a Linux perspective; (Wheel er) is also a useful resource. Developers need to thoroughly understand these types of security -related defects to avoid them during coding, and that these defects may be manifested in the acts of an intelligent and malicious agent (Zeltser), not the resu lt of random events. Even a "normal" defect that causes a system to crash can be exploited for a denial of service attack. Particular attention should be paid to defects in subsystems that "touch" the outside world (e.g., users, network traffic, file inp ut, etc.)

In general, many types of defects can lead to security vulnerabilities, therefore the things that developers do to reduce defects during implementation, in general, can help address security vulnerabilities, specifically. In addition to careful consideration during coding, developers should develop unit test cases with the potential for security exploits in mind. Code reviews should target previously identified risks with obvious vectors (e.g., code involved in network communication) (Reiter). F urthermore, code review checklists should include common implementation defects that impact security.

Risk Analysis in the Implementation Activity

Security conscious code reviews and unit tests will help identify implementation vulnerabilities. Developers normally address unit test defects immediately, but code reviews often occur after -the-fact. Vulnerabilities identified during code review should be used to update the risk analysis.

2.6 Test

The test activity is concerned with testing the system overall (un it tests are performed during implementation). Tests typically involve integration and complete system tests.

Since the system's use cases describe the functionality of the system they also guide the functional testing of the system. The security require ments identified in the requirements activity should also be rigorously tested.

In addition to testing the system, it is worthwhile to test the security posture of the platform on which the system runs. Depending on the criticality or sensitivity of the s ystem, this might extend to the testing of the hardware for vulnerabilities (e.g., physical security, emissions vulnerability, etc.) ("TAMPER"). More commonly, security assessment tools that assess software platforms for known vulnerabilities can, and sho uld, be employed ("Harris").

Finally, penetration testing (SANS, Penetration) should be employed during the test activity. Although penetration testing is more commonly done by, or for, system administrators in an organization, it is also a useful securit y testing technique.

Development organizations and software quality assurance departments that do not have skills in this area should consider retaining the services of a professional.

While system security tests, security assessments, and penetration tes ts cannot uncover every security defect, they can help characterize the security posture of the system. It is not necessary that the systems have no security vulnerabilities, just as it is not necessary for systems to ship with no known defects. To decid e whether a discovered vulnerability is acceptable, the testers and developers should examine their risk analysis.

Risk Analysis in the Test Activity

System tests will refine the impact and severity estimates of previously identified risks. Security assess sment tools and penetration testing can refine estimates of identified risks, while also discovering new ones.

2.7 Recommendations

A concern for system security should be central in the development of critical systems. The following table summarizes what deve lopment practices can be used to develop more secure critical systems.

Table 2—Summary Security Measures Taken During Development

Activity	Security Measures

Requirements	• Establish overall security policies and goals. What informat ion are you protecting and why? What are the implications of non - availability? Are there any regulatory constraints?
	• Start the risk analysis:
	  o Focus on likely threats (frequency).
	  o Understand the impact of each risk, using the overall security policies and goals as a guide.
	• Attach non-functional security requirements to use cases. Consider confidentiality, availability, and integrity vulnerabilities for use case.

Analysis	• Update the risk analysis:
	  o Identify logical unknowns and inconsistencies that can be exploited.
	  o Ensure the analysis concepts are consistent with the system security goals.
	• Consider concepts to mitigate risks

Design	• Update the risk analysis.
	• Design with security awareness.
	• Identify new risks:
	  o Examine interfaces between components and to th e outside world.
	  o Test and examine third party components.

	○ Consider likely vectors of attack. • Associate design vulnerabilities with components, subsystems, and interfaces.
Implementation	• Code and unit test with security awareness. • Perform security-aware code reviews. • Update the risk analysis from code reviews.
Test	• Update the risk analysis: ○ Refine impact and severity estimates of previously identified risks from system tests. ○ Identify new risks from platform security assessments and penetration testing.

In addition to refinement of the software development process to accommodate system security, there are a number of other things that development organizations can do.

Developer Training

Successfully implementing a secure critical system according to the process outlined above presumes that the developers are familiar with system security concepts and common system vulnerabilities. Unfortunately, this is not common. Development organizations should undertake measures to educate their developers through a systematic training program.

Hardened Platforms

Many exploitable system security vulnerabilities are the result of weaknesses in the platforms on which they are built. There are likely a number of reasons for this. First, platforms are complex pieces of software that provide many more opportunities for security-related defects. Second, platforms have a familiarity and commonality that make them more frequent targets for direct hacking/cracking and malware (malicious software) attacks. Niche system products do not receive the intense scrutiny of such wide community, so though they may have vulnerabilities they are far less likely to be successfully exploited.

To address the problem of platform vulnerabilities, development organizations should consider a parallel effort for platform hardening. A number of reference materials exist for securing commercial software platforms (SANS, Linux)(SANS, Securing)(SANS, Windows) and the techniques outlined should be judiciously applied. Technology can also be used to harden a platform, including host-based firewalls intrusion prevention systems ("StormWatch")("ZoneAlarm")("Tiny").

A parallel secure platform effort would be particularly efficacious for vendors with multiple product lines on the same platform. The platform can be secured, ghosted ("Symantec"), and reused for multiple products during development and deployment. Unfortunately, a secure platform is a moving target and the development organization should plan for a continuous effort of monitoring and correcting platform vulnerabilities as they are revealed.

Optimized Revision Cycles

Although critical systems require more extensive testing cycles, customers should not have to wait longer than is necessary for revisions to close security vulnerabilities. Vendors should implement concurrent development and testing activities specifically for reacting to new security threats; utilizing automated regression tests can facilitate a quicker response. Waiting for the next "planned" product release is no longer an option.

Coding Standards

Coding standards are used in development organizations to increase productivity, quality, maintainability, and understandability. Coding standards can make the resulting system more consistent, portable, reusable, and testable (Straker). Coding standards should incorporate usage rules that help prevent security vulnerabilities, as well.

Isolated Development Network

Development organizations should consider isolating their development networks from their corporate networks. If the development systems are compromised, then every product developed using those systems is vulnerable. Viral infections are possible, but they are more easily detected and repaired than the covert installation of Trojans and trap doors in systems[5].

3 Support

A critical issue for vendors is the support infrastructure required for their products. If support simply involves periodic patches, new software versions, and occasional telephone consultation, then the security implications are minimal. However, complex remote support requires that a vendor's support personnel be able to access and control these systems. This is potentially a serious security vulnerability, because if remote support personnel can gain access to these systems, so might malicious unauthorized individuals. This is a particularly problematic if, as is often the case, remote support personnel have administrative access. Using the remote support infrastructure, an intruder can gain control of a system on an organization's intranet, putting the organization's entire IT infrastructure at risk.

3.1 Remote Support Techniques

Remote system support requires a communication mechanism, a way to view the system state, and a way to manipulate the system. Several techniques are commonly employed:

Dial-In Modems

A modem is attached to the system allowing a support engineer to "dial-in" to the system. This is a particularly dangerous communication mechanism because an intruder can gain unfettered, unfiltered, and unmonitored access to a system specifically intended to allow the system to be manipulated. Implementing personnel procedures that require local personnel to activate and deactivate the modem at specific times, for specific purposes, can provide an additional measure of security. Unfortunately, access is still unfiltered, unmonitored, and personnel procedures are susceptible to social engineering attacks.

Centralized Remote Access Server

[5] (Thompson) describes how a Trojan can be written so that no trace of the Trojan can be found in the system's source code.

A Remote Access Server (RAS) is used in many organizations to allow external access. RAS can also be used to give the vendor's support engineers remote access to the systems they support. Connections and traffic on remote access servers can be monitored. Unfortunately, access provided to the network through a centralized RAS that is general enough for a wide variety of users is too general for support personnel that only need access to specific systems, violating the "principle of least privilege." (Saltzer)

Remote Control Software

Software products like pcAnywhere ™ are popular for remote support because they allow support personnel to fully manipulate the system ("pcAnywhere"). Unfortunately, such power gives remote personnel effective access to the overall network, again violating the principle of least privilege. Furthermore, anyone else on the network, either local or remote, can gain complete access to this critical system. The advantage for the system developers is that they do not have to build specialized support tools or client-server software to monitor and control the system; support personnel can have flexible access to the system in unforeseen circumstances. The risk from unauthorized personnel can be partially mitigated through the use of secure authenticated and encrypted communication channels[6]. Nevertheless, system administrators tend to be nervous about granting this level of remote control to someone outside their organization.

Client/Server Support Platform

Specialized access and control mechanisms can be built into systems to allow support personnel to monitor and control the system in well-defined ways. This can be done through a web service over http, using a browser at the client site, or with custom client and server software. This support technique requires more development effort, but it limits the type of control support personnel can exert. On the positive side, when administrative control is limited it's harder for support personnel or unauthorized people to gain complete access to the internal network. On the negative side, it is difficult to fully enumerate all the support functions that will need to be performed; if the necessary support cannot be provided remotely, because the client/server support platform does not permit it, then a costly and time consuming on site visit may be necessary.

3.2 Support Organization Security

If remote support organizations need to gain remote access to systems at a customer's site then their clients' system and network and security is only as good as the support organization's. If the support organization is penetrated, then penetrating every client's site is enormously simplified.

Remote support organizations need to be aware that their vulnerabilities are also their clients'. Administrative passwords to client systems, especially, need to be secured. Sadly, it is not unusual for multiple client systems to share support passwords. This simplifies support, but is a huge security vulnerability.

3.3 Recommendations

Secure systems should be developed with the security of the support infrastructure in mind. Consistent with principle of least privilege, developers should consider developing specialized client/server platform that only permits specific support functions to be

[6] pcAnywhere ™ incorporates a number of security features, including embeddable security codes, a variety of authentication mechanisms, and various forms of encrypted communication.

performed remotely. These support functions should be determined in development during the requirements activity.

Communications access to internal systems for remote support should use a centralized remote access server specifically for that purpose, enabling centralized monitoring and control. However, an internal firewall should isolate the remote access server from the rest of the internal network and be configured to only allow network traffic for remote support of specific systems. System administrators should consider procedures that require their involvement during vendors' remote support activities, with remote support blocked at all other times.

Support organizations need to secure their own systems to prevent security compromise of their clients. Passwords should conform to accepted practices to prevent password cracking and should be distinct for each system.

Security incident reporting procedures should be implemented to track and correct security vulnerabilities.

4 Services

Every organization within which a system is deployed has a different security environment (different security policies and infrastructure), therefore each deployed system will have different security risks. It is impossible to take every possible security environment into account during system development. Vendors should offer services that adapt a system to the security requirements of their clients' organization. Such services can take many forms:

Security Assessments

The vendor can perform a security assessment to determine the risks to the system within the client's organization. The risk analysis performed during development should be updated to reflect different threat profiles or reassess the impact of a failure in the deployed environment. Particular attention should be paid to the assumptions made during the requirements activity of development to ensure that these assumptions have not been violated.

Configuration and Installation

The vendor can provide configuration and installation services to ensure that the system is configured and installed in such a way that it does not violate any assumption relied upon during development (e.g., physical environment, disabled CD or floppy drives, etc.).

Additional Security Infrastructure

To provide additional security in-depth, the vendor can install additional security infrastructure, including isolated networks, secure perimeters with internal firewalls (Bridge) and intrusion detection systems.

5 System Retirement

One, often overlooked, phase in the lifecycle of a system is its retirement. When it is considered, it is usually viewed as the end of the system's support lifetime. But from a security perspective, this is not sufficient.

It is possible that a system may be at the end of its support life, but is still deployed at a customer's site and, therefore, still vulnerable to security attacks. Even worse, if the

system becomes unused or neglected, it can be come a source for security compromises to the entire IT infrastructure. Like a rotting corpse in the middle of a busy town square, its only function is to spread disease.

Both vendors and deploying organizations need to guard against these necrotic system s by establishing rules for their retirement when they are deployed. Vendors should provide ample notice of when the system will no longer be supported and system administrators need to remain vigilant by making plans for the retirement, and possible repl acement, of outdated, unsupported systems.

6 Conclusion

Critical systems require special measures to secure them —failures of any kind can result in irreparable damage. Vendors of these systems can make a major contribution to their systems' security by chan ging the way they specify, develop, deploy, service, and retire systems. These changes are fundamental and pervasive, affecting the entire product lifecycle. It is tempting to think that minor changes to a software product can "fix" the security problem. Unfortunately, security threats are becoming more serious and more numerous—simple solutions are not possible. System administrators are becoming more security conscious and taking measures to secure their IT infrastructure, but they are becoming overwh elmed by threats and the flood of patches and updates foisted upon them by vendors caught in an endless and costly "penetrate -and-patch" cycle.

These changes will not be easy, but the cost of not changing can be higher. There is a bright side, however. B y addressing security directly and pervasively, the frequency of security breaches can be reduced and vendors can focus on developing products rather than responding to new security threats with an endless series of security patches. The result will be more satisfied customers, and *that* is a competitive advantage. Finally, from a societal perspective, we all benefit when we can rely on our critical infrastructure.

7 References

[1] Anderson, Ross and Needham, Roger. "Programming Satan's Computer." Cambridge University Computer Laboratory. 1995.
URL:http://www.ftp.cl.cam.ac.uk/ftp/users/rja14/satan.pdf (15 March 2002)

[2] Anderson, Ross. Security Engineering: A Guide to Building Dependable Dis tributed Systems, Foreword by Bruce Schneier. New York: John Wiley & Sons, Inc., 2002: xxiii-xxiv

[3] Beyond Security, Ltd. "Race Condition in FreeBSD AIO Implementation." Beyond-Security's SecuriTeam.com Monthly Report. 11 December 2001.
URL:http://www.securiteam.com/exploits/6L00E0A3FC.html (26 March 2002)

[4] Boehm, Barry W. Software Engineering Economics. Englewood Cliffs: Prentice -Hall Inc., 1981: 35-56.

[5] Boehm, Barry W. "A Spiral Model of Software Development and Enhancement." IEEE Computer, Vol. 21, No. 5. May 1988: 61 -72.

[6] Boehm, Barry. "Spiral Development: Experience, Principles, and Refinements." Spiral Development Workshop. July 2000.
URL:http://www.sei.cmu.edu/cbs/spiral2000/february2000/SR08.pdf (18 March 2002)

[7] Booch, Grady. The Unified Modeling Language User Guide. Reading: Addison Wesley Longman, Inc., 1999.

[8] Bridge, Steve. "Achieving Defense -in-Depth with Internal Firewalls." SANS Institute Information Security Reading Room. 15 August 2001. URL:http://rr.sans.org/firewall/internal.php (25 February 2002)
[9] "Common Criteria." commoncriteria.org. URL:http://www.commoncriteria.org/ (29 March 2002)
[10]Ghosh, Anup K., et. al. "Building Software Securely from the Ground Up." IEEE Software. Vol. 19, No. 1. January/February 2002: 14 -16.
[11]Jacobson, Ivar. The Unified Software Development Process. Reading: Addison Wesley Longman, Inc., 1999.
[12]"Harris STAT® Scanner." Harris, Inc. URL:http://www.statonline.harris.com/ (18 March 2002)
[13]Knight, Jeff. "The Common Criteria for Information Technology Security Evaluation—In with the new and out with the old." SANS Institute Information Security Reading Room. 9 December 2000. URL:http://rr.sans.org/securitybasics/criteria.php (29 March 2002)
[14]Kruchten, Philippe. "What is the Rational Unified Process?" The Rational Edge. January 2001. URL: http://www.therationaledge.com/ content/jan_01/f_rup_pk.html (20 March 2002)
[15]Kulak, Daryl and Guiney, Eamonn. Use Cases: Requirements in Context. ACM Press and Addison-Wesley. 2000.
[16]Levenson, Nancy G. and Turner, Clark S. "An Investigation of the Therac -25 Accidents." IEEE Computer, Vol. 26, No. 7. July 1993: 18 -41.
[17]Lucero, Angelina. "Three Defenses to a Secure System: Virus Scanning, Applying Patches and System Monitoring." SANS Institute Information Security Reading Room. 18 September 2001. URL: http://rr.sans.org/win/3defenses.php (15 March 2002)
[18]Neumann, Peter G. "Risk to the Public in Computers and Related Systems." ACM Software Engineering Notes, Vol. 13, No. 2. April 1988: 5 -18.
[19]Newsham, Tim. "Format String Attacks. " Help Net Security. 2000. URL:http://www.net-security.org/text/articles/string.shtml (15 March 2002)
[20]Northcutt, Stephen. "Information Assurance Foundations." The SANS Institute. 24 October 1999 - 19 June 2000. URL: http://www.sans.org/y2k/iafoundations.pdf (26 March 2002)
[21]"Norton AntiVirus™ Professional Edition." Symantec, Inc. URL:http://www.symantec.com/ (15 March 2002)
[22]"pcAnywhere™." Symantec, Inc. URL:http://www.symantec.com/ (15 March 2002)
[23] "Rational Unified Process: Best Practices for Software Development Teams." Rational, Inc. URL:http://www.rational.com/products/whitepapers/100420.jsp (18 March 2002)
[24]Raynal, Frédéric, et. al. "Avoiding Security Holes when Developing an Application, parts 1 through 6." LinuxFocus. January 2001-November 2001. URL:http://www.linuxfocus.org/English/November2001/article203.shtml (15 March 2002)
[25]Reiter, Michael. "Security Code Review." SANS Institute Information Security Reading Room. 18 November 2000. URL: http://rr.sans.org/code/code.php (26 March 2002)
[26]Rumbaugh, James. The Unified Modeling Language Reference Manual. Reading: Addison Wesley Longman, Inc., 1999.
[27]Saltzer, Jerome H. and Schroeder, Michael D. "The Protection of Information in Computer Systems." Proceedings of the IEEE. Vol. 63, No. 9. March 1975: 1278 -

1302. URL: http://web.mit.edu/Saltzer/www/publications/protection/index.html (29 March 2002)
[28] SANS Institute. "Penetration Testing." SANS Institute Information Security Reading Room. 17 January 2002. URL: http://rr.sans.org/penetration/penetration_list.php (18 March 2002)
[29] SANS Institute. Securing Linux Step-by-Step. Version 1.0. Bethesda: The SANS Institute, 2000.
[30] SANS Institute. Windows NT Security Step-by-Step. Version 3.03. Bethesda: The SANS Institute, February 2001.
[31] SANS Institute. Securing Windows 2000 Step-by-Step. Version 1.5. Bethesda: The SANS Institute, 1 July 2001.
[32] Shimomura, Tsutomu. "Technical details of the attack described by Markoff in NYT." Newsgroups: comp.security.misc,comp.protocols.tcp-ip, alt.security. 25 January 1995. URL: http://gulker.com/ra/hack/tsattack.html (18 March 2002)
[33] "StormWatch™ Intrusion Prevention Software." Okena, Inc. URL: http://www.okena.com/ (26 March 2002)
[34] Straker, David. C Style: Standards and Guidelines. New York: Prentice Hall. 1992: 5-7.
[35] "Symantec Ghost™ Corporate Edition." Symantec, Inc. URL: http://www.symantec.com/ (15 March 2002)
[36] "TAMPER Lab Home Page." University of Cambridge Computer Laboratory. URL: http://www.cl.cam.ac.uk/Research/Security/tamper/ (18 March 2002)
[37] Thompson, Ken. "Reflections on Trusting Trust." Communications of the ACM, Vol.27, No. 8. August 1984: 761-763. URL: http://www.acm.org/classics/sep95/ (26 March 2002)
[38] "Tiny Personal Firewall." Tiny Software, Inc. URL: http://www.tinysoftware.com/ (26 March 2002)
[39] Trice, Roland. "Results of the UKERNA Risk Analysis Programme." SuperJANET4. 2 February 2000. URL: http://www.superjanet4.net/risk/ukerna_rep.pdf (21 March 2002)
[40] Verton, Dan. "Study: Constant security fixes overwhelming IT managers." Computer World. 30 November 2001. URL: http://www.computerworld.com/storyba/0,4125,NAV47_STO66215,00.html (15 March 2002)
[41] "VirusScan™." McAfee, Inc. URL: http://www.mcafee.com/ (15 March 2002)
[42] Wheeler, David A. "Secure Programming for Linux and Unix HOWTO." Linux Documentation Project. 12 March 2002. URL: http://www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO/ (15 March 2002)
[43] Zeltser, Lenny. "The Evolution of Malicious Agents." SANS Institute Information Security Reading Room. 2 May 2000. URL: http://rr.sans.org/malicious/agents.php (25 February 2002)
[44] "ZoneAlarm® Pro." Zone Labs, Inc. URL: http://www.zonealarm.com/ (26 March 2002)