A Proactive Approach Toinformation Security

Sandeep Gupta

A PROACTIVE APPROACH TO INFORMATION SECURITY

Sandeep Gupta

B.Math. Computer Science and Business Joint Honors, and MCSD

GSEC Practical

Version 1.4b

December 23, 2003

ABSTRACT

Some software vendors already endeavor to deliver software systems that provide Confidentiality, Integrity, and Availability of a customer's software, hardware, and data assets. However, because of the changing business environment, because of new attack hazards, and because of the damages of an insecure system, all software vendors cannot assume that they are secure. Vendors must be proactive and address security early in the software development lifecycle (SDLC) by focusing on training, by performing risk and threat assessments, and by designing security into the software system.

Each software vendor differs in their implementation of the SDLC. By integrating both the author's experience and multiple sources of industry thought, this paper presents a generalized yet holistic view of integrating security in the SDLC. This paper serves as a springboard for a vendor who has little experience in security, and who is considering integrating security in the SDLC to create a more secure software system.

Table of Contents

1 Making the Case for Information Security in the SDLC

1.1 What is a secure software system?

Any software system consists of software, hardware, and data assets and solves a business problem. A secure software system fulfills its purpose and provides Confidentiality, Integrity, and Availability (CIA) of assets by resisting and surviving a broad range of:

- Deliberate attacks by internal and external attackers,
- Accidents,
- Natural events such as earthquakes and fire,
- Crashes, faults, and other unplanned downtime, and
- Errors in the business process or in the data.

1.2 Changing business environment

Some software vendors already endeavor to provide secure and reliable systems. The vendor's software architects spend time designing security into the system.

On the one hand, the nature of business is moving towards online e-commerce, and sensitive personal and corporate information is available on the network. Businesses make corporate data available to employees on laptops or PDAs equipped with wireless mobility. This technology brings about fundamental shifts in the structure of society and allows not only personal freedom, but also new professional opportunity.

On the other hand, this new environment is inherently hostile giving way to new attack targets and threats which carry significant risk. Many people do not understand or recognize this risk. [Ref 1]

1.3 Attacker's Mindset and a Myriad of Threats

Every system on a network is under attack by intelligent internal and external attackers with patience and time. Experts debate the intent of attackers, but it is safe to say that attackers use malicious software worms and trojans [Ref 2], specially crafted data packets [Ref 3], and social engineering [Ref 4, 5] simply because it is possible. Attackers exploit vulnerabilities in the software system that allow them to either bypass established security controls, or to compromise a system by running code that renders the system insecure. Once attackers bypass security, then assets are at risk.

The security industry places an onus on issuing security badges, deploying firewalls or on encrypting secrets. If the firewall lets the attacker's traffic through to compromise the main web server with a specially crafted TCP/IP packet, then an employee badge, cryptography, not even a firewall can protect the assets.

Other natural or accidental actions or events such as crashes, unintended errors, or even an earthquake may also prevent a system from fulfilling its main functions.

1.4 Damages

If there is a security incident, both the vendor and the customer dispatch incident handling teams to investigate the breach and coordinate recovery procedures. This

requires time, money, and effort. In terms of software development, the resulting opportunity costs of 1) reproducing the exploit, 2) creating, testing, and installing a fix, 3) contacting customers, and 4) writing documentation, can all make a significant impact on productivity and the bottom line.

When the Canadian government was the target of computer theft containing sensitive data about businesses and individuals, Confidentiality was broken. The government sent "letters to approximately 120,000 people who might be affected. … Although the stolen laptop/server was password-protected, the data on the machine was not encrypted."[1] The cost to mail notices to 120,000 people is high; however, the cost of a case of identity theft is also high for the people affected.

Lawsuits also represent negative publicity and a loss of confidence regardless of the merits of the suit.[Ref 8] With security and privacy being a hot topic, customers who incur any losses may take this course of action as is the case with Microsoft® today. [Ref 9, 24, 25].

Irrespective of the merits of a suit, an attacker who successfully exploits software security flaws and penetrates an enterprise computer may not only steal data, but also steal intellectual property as well. Confidentiality and Integrity are broken in addition to the customer's confidence.

The Morris worm that "paralyzed the internet in late 1988"[2] used a buffer overflow vulnerability in the finger daemon. Availability was broken [Ref 6]. While many debate whether this type of catastrophic event can happen again, it is unwise to make any assumptions about the future of technology. Society's dependence on software is growing in all areas of professional life from laptops to PDAs, and of personal life from online gaming to wireless mobility. Even a small incident can result in huge losses for a single user.

With many businesses installing software systems to further productivity and realize competitive advantage, a loss of intellectual property, loss of quality, and even loss of confidence can cripple a vendor's ability to grow market share.

1.5 The Proactive Approach

Because of the changing business environment, because of the attack hazards, and because of the potential damages, all software vendors cannot assume that they are secure. Vendors must be proactive and address security early in the software development lifecycle (SDLC) by focusing on training, by performing risk and threat assessments, and by designing security into the software system.

It is not possible to make a system 100% secure - it will never ship. However, these commitments to security by management empower a software architect to design a more secure software solution.

See [Ref 11] for more information about the definition of secure software systems.

2 Change Management Leads to Training and Security Sub-Culture

Once management and the customer agree that security is a focus area, management initiates a change management process to integrate security into the entire software development lifecycle (SDLC). Depending on the needs and size of the organization, a change management program may be small with email communications, or may be large with training programs.

2.1 Sponsor

The sponsor for the change management program within the software development organization is a high level and influential manager. This manager owns the overall software development program and:

- Works with the office of the executive to align the vendor's policies
  - to integrate security in the entire vendor organization, and
  - to integrate security in the SDLC,
- Establishes a detailed strategy for integrating security throughout the entire software development organization,
- Works with other executives to establish common strategies in other functional areas in the organization,
- Commits funding for security training, and for other specific security activities in the SDLC,
- Designates a software architect who becomes the Lead Security Subject Matter Expert (SME),
- Announces and supports a security training program for all staff within the software development organization,
- Ensures that the training "material is appropriate and timely for the intended audience"[3],
- Ensures that software architects and software designers "have an effective way to provide feedback"[4], ensures that the "material is reviewed periodically and updated when necessary"[5], and,
- Takes responsibility for engaging the customer in the requirements process, and for making the customer aware of related security related laws, policies, and standards.

2.2 Change agents

The sponsor empowers software architects to take actions to support security in the context of the software development program.

A software architect becomes the change agent and is the Lead Security SME within their domain.

Because the architect owns the overall software system architecture and the code itself, it is generally in their best interest to develop a secure system. The architect:

- Understands software designers' current knowledge and needs; therefore, assists in the development of training material, assists in soliciting feedback, and assists in reviewing and updating material for software designers,
- Works with software design managers to draft training plans according to software designers' roles and responsibilities in the system design,
- Conducts training sessions for software designers and shares knowledge of the latest security issues and methodologies with designers,
- Works with program managers, project managers, software design managers, and test managers to ensure that they can correctly identify and prioritize security issues throughout the SDLC,
- Works with managers to ensure all staff get training according to the needs, roles, and responsibilities, and,
- Works with software designers to resolve security issues, and works with lead security SMEs from other functional areas of the organization.

The architect is the Lead Security SME and may appoint other security SMEs who have specific responsibilities. Security SMEs report on architectural and coding issues to the Lead Security SME.

2.3 Change targets

Software designers are the change targets. Many software designers enjoy learning new technology and take an active interest in new coding methods.

The goal of the change management program is to build a knowledge base for software architects and software designers to design and develop a more secure software system. Not wanting to be associated with highly-visible security-based design issues, designers start to take security seriously and start to document their security inspirations on napkins at the lunch buffet. That is when a security sub-culture forms and the real action starts.

2.4 What is the right training material?

Training depends on individual needs, responsibilities, and experience. Training helps people understand how to:

a) Identify and recognize threats in the software system – Potential for a natural, accidental, or deliberate action or event to exploit a vulnerability which may have detrimental effects that prevent the system from fulfilling either its purpose or its goals of CIA,

b) Identify and recognize vulnerabilities in the software system – Locations in the business process or the software system where a weakness exists such design or coding flaws, invalid assumptions, invalid input data, or insecure storage or transmission of data,

c) Conduct a risk determination - Identify and measure the probability and result on the CIA of assets or on the ability of the system to fulfill its purpose should a threat-action or event inflict damages,

d) Identify and recognize mitigation techniques – Tools and techniques that minimize or eliminate the risk and allow the system to either fulfill its purpose or its goals of CIA,

e) Keep current on security issues such as viruses, trojans, as well as vulnerabilities and exploits.

The vendor gears the training material to peoples' level of experience and responsibility.

See [Ref 10] for ideas on training.

2.5 Responding to objections

A classic objection to security is that people do not have time to attend security training or to resolve security issues during the SDLC. The concern reflects the triangle, illustrated in Figure 1 which is the balance between labor, schedule, and features. The answer to this is quite simple: addressing security early in the SDLC results in a cost savings and reduced risk. Addressing security later in the SDLC or even after the software is released results in significant damages and additional project risk.[6] [Ref 13]

The project manager ensures that the schedule provides enough time throughout the SDLC to handle security issues. Designers and managers also need to impress upon the importance of this time.

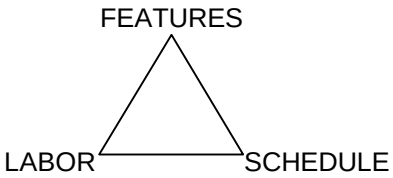

Figure 1 Triangle

With security on consumer's minds, any security issues can have severe repercussions. It is unwise to make any assumptions about the future or about the security of a software system.

While something is not possible today, technology may make it possible tomorrow. That is what technology is all about, is it not?

2.6 What about other functional areas?

Other functional areas also require a similar change management program which leads to a security sub-culture within the vendor; however, the scope of this discussion is limited to software development.

2.7 What about the customer?

The customer may also require a security training program; however, the scope of this discussion is limited to software development.

3 The Software Development Lifecycle

On the one hand, the "constant demand for novelty means that software is always in the bleeding-edge phase, when products are inherently less reliable."[7] On the other hand, Computer Science college professors stress the proven fact that a well-thought-out design reduces the number of issues that architects and designers deal with at later stages in the SDLC.

Now that software designers have a basic knowledge foundation and both management and the customer place a high priority on building a more secure software system, it is possible for security to pervade the SDLC.

Table 1 Phases of the SDLC, describes at a high level the 5 phases of the SDLC and what security actions the vendors takes during each phase.

Table 1 Phases of the SDLC

Phase of the SDLC	**Security in this Phase**
Section 3.1 Phase 1	Perform preliminary Threat and Risk Assessment. Identify functional security requirements broken down into Confidentiality, Integrity, and Availability.
Section 3.2 Phase 2 - Architecture and Design	Perform detailed threat and vulnerability identification. Perform risk determination. Architect and Design mitigation techniques.
Section 3.3 Phase 3 - Software Coding	Enforce coding and code submission practices.
Section 3.4 Phase 4 - Test and verification	Conduct vulnerability and threat testing.
Section 3.5 Phase 5 - Deployment and Maintenance	Identify and classify security issues. Fix security issues based on risk.

3.1 Phase 1 - Needs Analysis and the Business Case

In the needs analysis and business case phase, the vendor obtains, understands and documents the customer's needs, and then prioritizes the customer's requirements with their approval.

The vendor outlines proposed solutions in a feasibility study along with cost factors, a preliminary schedule, and solution alternatives. If the solution represents an appropriate return on the vendor's investment, and if management gives the green light

to proceed, then the program managers, test managers, and software architects draft system requirements and functional requirements. These new requirements document the details of the system and should map directly back to the customers requirements. See Section 3.4 Phase 4 - Test and verification for more information on why the test organization is involved early in the SDLC.

3.1.1 Security in Phase 1

The vendor guides the customer and documents additional security related system and functional requirements according to the customer's security needs.

3.1.1.1 Regulatory Environment

The vendor discusses and documents security needs with the customer based on current security related laws and policies, government regulation, international standards. This may include, among others, HIPAA [Ref 26, 27], ISO 17799 [Ref 27], or the new Canadian Privacy Law [Ref 28].

3.1.1.2 Perform a Preliminary Threat and Risk Assessment (TRA)

The goal of a preliminary TRA is to identify the assets, to identify the environment in which the assets operate, and to identify the requirements to protect the assets.

The vendor performs a preliminary TRA for new and existing features and functionality in the system. "A preliminary risk assessment should define the threat environment in which the … system will operate. This assessment is followed by an initial identification of required security controls that must be met to protect the … system in the intended operational environment."[8]

The preliminary TRA documents the operating environment by studying the current state of the system, the customer's business process, and the end users of the system in the context of security. This information is generally already available in other forms of documentation and used as input to the preliminary TRA.

The vendor documents the current state of the system by examining the customer's existing network, by examining existing features and functionality in the software system, and by examining the current security architecture. As a result, the vendor understands the current configuration management as well as other deployment and maintenance considerations. This analysis yields the configuration and versions of patches, operating systems, databases and storage networks, firewalls, desktop applications, as well as the current access control, and authorization mechanisms. Other documentation such as network configurations, disaster recovery plans, and data backup plans provides detailed information on the customer's business continuity plans.

The lead security SME understands not only the customer's business process, but also the end-users of the system.

One aspect of the customer's business process includes the customer's policies, procedures, and best practices. This allows the vendor to understand the customer's business environment and competitive environment.

Another aspect of the customer's business process is the data itself and the users of that data. A high level systems analysis can represent valuable input to a preliminary TRA. A high level systems analysis enables a further understanding of the customer's business by providing information about:

- Data,
- User types and roles in the system,
- Users thoughts and concerns security and privacy,
- Interactions between the data and its various users, error checking,
- Importance and sensitivity/classification of the data,
- The processes which operate on the data, and
- The flow and storage of data both on paper and within the system.

In essence, an analysis of the business process and the end users assists in identifying the assets and areas where CIA is required. By questioning each assumption, architects learn how to protect the assets in later phases of the SDLC.

In addition to examining the customer's environment, the vendor examines its own environment. This includes not only the vendor's network and software platforms, but the software system code and architecture itself as well. This is done only in the context of the customer's security requirements and may include identifying invalid architectural assumptions, or identifying homegrown encryption or access control code that may not fulfill the security needs. The architect identifies any necessary architectural modifications for further study to ensure that there is no unforeseen scheduling or cost overrun.

It is impractical to conduct a thorough TRA this early in the SDLC. The TRA is a living process. As the SDLC progresses, the TRA gets more detailed.

By conducting a preliminary TRA, it is possible to accurately identify the assets to protect, and to determine the functional security requirements. The architect is later able to define the threats to the system and the mitigation techniques later phases of the SDLC.

See [Ref 10] for detail on risk management.

See [Ref 12] for more detail on determining security goals.

3.1.2 Expressing Security Needs

Now that the architect fully understands the customer's business, business process, and the software system's operating environment, the architect can express security in terms of Confidentiality, Integrity, and Availability functional requirements.

See [Ref 11] for detail on expressing security requirements.

3.2 Phase 2 - Architecture and Design

In Phase 2, the architect makes specific decisions regarding the software, system, and component architecture in order to fulfill the functional requirements. The architect documents and explores any necessary architectural modifications and decides how to proceed with the modifications.

The program managers, projects managers, and the architect also draft a detailed schedule. It is important for all staff involved to understand that security is a key priority in the SDLC, and therefore, management must allocate sufficient resources for the security tasks.

3.2.1 Security in Phase 2

The architect, program managers, software designers, and testers perform a detailed TRA. There is a four-step process.

1. Identify the threats to the system,
2. Identify the vulnerabilities in the system,
3. Determine the risk to the system, and
4. Develop risk mitigation techniques.

The test organization also gets involved and begins drafting test plans that tests mitigation techniques designed into the software system.

3.2.1.1 Identify the threats to the system

In order to deliver CIA within the software system, the architect and the designers understand what are the possible actions or events that may compromise the system's ability to deliver CIA or fulfill its mission.

Following from the analysis in section 3.1, Perform a Preliminary Threat and Risk Assessment (TRA), the following are high-level inputs to the threat identification process:

- Functional Requirements, and Functional Security requirements,
- High level systems analysis – data,
- Networking environment – network configuration and connectivity,
- Operating environment – operating systems, databases, software and hardware platforms,
- Architectural, design, and coding assumptions, and application dependencies,
- Security architecture including access control, authorization, and accounting mechanisms, and
- Studies about the users.

All of these inputs represent the overall architecture of the system. It is not necessary to get too detailed at this point; a high-level overview is sufficient. The inputs represent identified software, hardware, and data assets and are often called threat targets.

A threat-source or threat-agent is the source of threat. It is person or object from where the threat originates, for example, a lighting strike, or an untrained employee. The threat-source takes action or initiates an event against a threat target.

Some examples of threat-source/threat-agents are:

- Earthquakes
- Floods
- Electricity suppliers
- Criminals
- Hackers
- Fire
- Disgruntled shareholders or employees
- Competing companies
- Employees – internal (full-time, part-time, Temp)

Some examples of threat actions or events that affect Confidentiality, Integrity and Availability are:

- Impersonation
- Social engineering
- Bribery, theft
- Distribute passwords freely
- Bypass access control
- Unauthorized access to data
- Unauthorized modification of data
- Unauthorized installation of software or hardware
- Invalid, incomplete, inaccurate data entry
- Tamper or delete data and logs
- Circumvent error checking
- Corrupt data
- Repudiation of transactions
- Authorized access to users is denied
- System goes off-line and cannot deliver vital services
- Usurp/spoof privileges or unauthorized privilege escalation

The architect uses threat identification to identify:

- Sources of threats,
- Type of threats (natural, accidental, deliberate),
- Actual threat action or event,
- Whether the action or event affects Confidentiality, Integrity, or Availability,
- Whether the target is a data, hardware, or software asset, and
- Threat target itself,

Often, attackers combine multiple threat-actions or events in order to attack a target. To this end, the architect considers threat trees. McGraw and Viega write:

> In assessing risk, we like to identify not only what the risks are, but also the potential that a risk can actually be exploited, along with the cost of defending against the risk.
>
> The most methodical way we know of achieving this goal is to build attack trees. Attack trees are a concept derived from "fault trees" in software safety (see Nancy G. Leveson's *Safeware: System Safety and Computers* [Addison-Wesley, 1995]). The idea is to build a graph to represent the decision-making process of well-informed attackers. The roots of the tree represent potential goals of an attacker. The leaves represent ways of achieving the goal. The nodes under the root node are high-level ways in which a goal may be achieved. The lower in the tree you go, the more specific the attacks become.[9]

This threat identification process identifies actions and events that have a detrimental impact on the system which prevent it from fulfilling either its purpose or CIA. Later on in this process, the architect ranks the threats.

Other useful references to consult on building threat trees are [Ref 30, 31].

3.2.1.2 Identify the vulnerabilities in the system

Threats only have a detrimental effect on the system when there is a vulnerability to exploit, when there is a threat source, and when there is a threat target.

This is where the real work begins, and this is where it is necessary to become realistic in the process. Malicious attackers do not play according to a ‘rule book’.

A classic objection is that nobody will attack the system with a malformed packet to a TCP/IP socket, with a buffer overflow, or with database stored procedures. The answer to that is simple. Any internal employee or external attacker attempts everything that they can when they are determined to compromise system security. Even if something is not possible today, technology makes it possible tomorrow.

Architects and designers alike must be able to identify and recognize how threats can manifest themselves, as well as understand how attackers exploit vulnerabilities. While it is important to avoid underestimating threats and vulnerabilities, it is also important to avoid overestimating them. This comes with experience and with building threat trees.

Some examples of vulnerabilities are:

- Buffer overflow!
- Buffer overflow!
- Buffer overflow!
- Excess or rigid, internal or external dependencies
- Unencrypted data storage or transmission
- Running features, services, protocols that are not required
- Invalid or incomplete assumptions
- No logs or audits, and no redundancy or fault tolerance implemented
- Improper implementation of cryptography
- Not handling failures securely
- Incomplete, incorrect validation of input or output data
- Difficult to disable features that are not required
- No access control or requiring administrator access

3.2.1.3 Determine the risk to the system

Once the architect identifies threats and vulnerabilities, it is possible to determine whether there is a measure of risk to warrant implementation of mitigation techniques. There are many different techniques to evaluate the risk; however, the basic function is composed of:

- Impact and size of damage
    - Whether the damage is localized or widespread affecting a critical system or a non-critical system. A customer's viewpoint has a strong bearing on whether a system or service is critical or non-critical.
    - The number of users affected by a threat is large or small.
    - A malformed packet may be able to cause a software system crash resulting in a denial of service. The denial of service may render a financial network inaccessible for hours affecting millions of customers.
- Likelihood of threat-source to take threat action/event, and exploit vulnerability
    - Whether a threat-source is motivated and is likely to exploit a vulnerability. Be careful with assumptions in this measurement. Use a database such as SecurityFocus at http://www.securityfocus.com to help guide the determination.

- Steps required to exploit the vulnerability
    - Whether a threat-source is able to find a vulnerability and produce an exploit.
    - Whether a threat-source can easily bypass mitigation techniques, and whether a threat-source is easily able to exploit the vulnerability.

Note that risk determination does not include the cost and effort required to fix damages as part of the calculation.

Occasionally, a delay in the schedule tempts managers to also delay a fix so that the team meets the 'go-live' or delivery date. This strategy also increases the overall risk to project if the threat is serious enough and an exploit either exists or an attacker can develop an exploit easily. Remember that attackers have patience and time.

The program managers, project managers, design managers, testers, and the software architects all collectively evaluate and rank risks. Once the architect measures risk and ranks and prioritizes the threats, it is possible to determine mitigation techniques to offset the possibility of any loss.

There are many different sources on managing risk and methodologies on conducting threat and risk assessments. See [Ref 10, 12, 14, 15, 16, 17, 18, 30, 31, and 32] for more detail.

3.2.2 Develop risk mitigation techniques

There are two different types of mitigation techniques: Corporate procedures and deployments as well as Architectural and design principles – Secure the Weakest Link.

3.2.2.1 Corporate procedures and deployments

During the change management process, the vendor and the customer alike develop procedures, policies, and guidelines for users using the software system. This encompasses policy and procedure development and strict enforcement including but not restricted to:

- Acceptable use policies, ethics policies,
- Guidelines to address social engineering and media enquiries,
- Guidelines addressing privilege creep, account termination procedures,
- System disposition, data sanitation, shredding,
- End user training on viruses and Trojans,
- Business continuity plans (and disaster recovery) which address natural and accidental threats-actions and events,
- Password usage policies and guidelines,
- Best practices for hardening network and operating systems,
- Firewall, web content filter, and intrusion protection system deployment

While these principles are important, in order to be proactive in the SDLC, it is important to focus on the principles in section 3.2.2.2 Architectural and design principles.

3.2.2.2 Architectural and design principles – Secure the Weakest Link

Architects question all assumptions and ask that others justify their assumptions in architectural and design reviews.

For example, the system can use cryptography to keep secrets, but attackers normally attack the system or the business process before the data encryption or after the data decryption.

> Let's say the bad guy wants access to secret data being sent from point A to point B over the network (traffic protected by SSLv2). A clever attacker will target an endpoint, try to find a flaw like a buffer overflow, and then look at the data *before* it gets encrypted or *after* it gets decrypted, because attacking encrypted data is too much work. Cryptography can't help you if there's an exploitable buffer overflow.[10]

By using a systems analysis as input to the threat modeling process, it is possible to find those areas in the system which may be susceptible to attack.

3.2.2.2.1 Defense In Depth

Practice defense in depth by applying more than one mitigation technique to protect the assets once the TRA identifies the vulnerabilities.

For example, for an e-commerce application,

> the credit card number should be encrypted before being stored in a database. Keep the key for the credit card number on a different machine (this requires decrypting and encrypting on a machine other than the one on which the database lives). That way, if the database gets compromised, the attacker still has to find the key, which requires breaking into another machine. This second line of defense raises the security bar.[11]

Applying multiple risk mitigation techniques protects all the software, hardware, and data assets helps the system to fulfill its mission and deliver CIA.

3.2.2.2.2 Buffer Overflow

By glancing through the SecurityFocus database, it is obvious that attackers heavily exploit the buffer overflow vulnerability over the network. For the technically minded, the 'Inside the Buffer Overflow Attack' paper [Ref 22] describes how overflows work. It is important to note that attackers who exploit buffer overflow vulnerabilities can take complete control of the system. There are several lessons stemming from this vulnerability.

1. When copying data, copy only the amount of data that the buffer can hold, and no more. Remember to account for the NULL terminating character if applicable.
2. Do not assume that the compiler will flag buffer issues for you – the load-builder can disable compiler checks in the regular load-build!
3. Check for all possible function return codes, especially for memory allocation.

4. Initialize all pointers immediately on pointer allocation.
5. Check for NULL pointers before accessing pointers.
6. Enforce and check array bounds before accessing arrays.
7. Ensure all strings are NULL terminated.
8. Centralize data processing close to the data itself. This allows greater control of the data and allows architects to question assumptions about control and use of the data.
9. Using encryption does not protect against software design flaws.

Verifying the design for buffer overflow vulnerabilities strengthens Confidentiality, Integrity, and Availability by resisting attackers trying to take control of the system.

See the following resources for more information on buffer overflows [Ref 12, 31, 33].

3.2.2.2.3 Cryptography

If the data is to be kept secret, there are several considerations in implementing cryptography.

- If the data is secret, consider not storing it at all. In that case, there is no threat-target.
- Consider not displaying secret data to the user.
- Many cryptographic algorithms use random number generators that generate random numbers. Some algorithms are vulnerable because the random number or the random number seed is predictable and therefore, the results are not uniformly distributed in the range.
- COTS cryptographic libraries are readily available to use in software systems. It is unwise to try to create a new cryptographic algorithm. Leave that up to mathematicians.
- There are issues in key management. With improper key management, storage, and exchange, an attacker can find the key and crack even the most powerful algorithms. Making it easy to perform key management allows people to follow the key management process. If the process is difficult and time-consuming, people cut corners -- compromising the key.

Cryptography is a large area of discussion. For more information see [Ref 12 chapter 8,9]. See Ref [23] for information and issues on Public Key Infrastructure.

3.2.2.2.4 Migrations

In software development, maintaining compatibility with old systems is a ‘hot-button’ issue because it is expensive. A mandatory migration to a new platform is a better option but it is also expensive. The problem with old systems is that supporting obsolete software platforms and discontinued operating systems with well-known security issues represents a significant security risk. Because administrators rarely patch these systems after the fact, internal and external attackers easily target them and compromise Confidentiality, Integrity, and Availability.

3.2.2.2.5 Principle of Least Privilege

For a transaction, the system and user requires the minimum possible privilege and for that transaction only.

Operations requiring root access should be done on a transactional basis, and should be done only for the most critical of tasks. When the transaction completes, remove the root privilege and revert to regular privileges. There are a number of reasons for this principle:

1. If a user has root privilege, then an attack can break confidentiality and integrity by allowing the attacker to escalate or usurp privileges.

   This includes databases which connect as an administrator. An attacker can then drop tables, delete table rows, and tamper with users, logs and stored procedures. Use COTS authentication systems or integrated authentication with the operating system for database access. As a result, users connect to the database as their own operating system account.

2. Root or administrator passwords may be handed out freely causing an account maintenance problem. The passwords to these accounts are seldom changed. In the hands of an attacker, it is gold. This breaks confidentiality, and integrity.

3. Permission defaults. An architecture that asks for read and write access unnecessarily breaks Integrity when only read access is necessary. Question the assumptions requiring access to all the resources when only access to one resource is required.

4. Not requiring any access control for a transaction breaks confidentiality, integrity, and availability.

5. Run detailed logging routines that check object access, logins/logouts, and so on. Automated tools that parse logs can look for anomalies that may require additional attention.

If an attacker tries to exploit a vulnerability, there is less impact because there are no privileges. Writing to the disk or modifying system settings or data may be specifically denied. If functionality does not work without root privilege, then software security checks are working and should deny non-privileged users access to features or functionality that they should not have.

By forcing software to run with unnecessary privileges, a relatively small attack can compromise the system.

Also tied to the principle of least privilege is the concept of Default Security installations. In this case, security is designed by default when the vendor deploys and installs the application. Getting administrators to turn on behavior or install features that are disabled by default allows administrators to be more careful when granting access and authorization to resources. If all the services and features are turned on by default, then administrators have a difficult time determining how to turn these features off if they are not needed. As a result, the features become targets for attackers.

It is important for the architect to spend time in this phase to ensure that the software system operates under the principle of least privilege. This preserves Confidentiality as well as Integrity, and Availability.

For more information on the principle of least privilege see [Ref 12, 31, 34].

3.2.2.2.6 Don't Trust User Input

There are three commandments with respect to user input in order to preserve Integrity.

1. Input from the user should be handled in centralized component or process. This allows the input handling and verification code be centralized as well.

   Centralizing input handling code in one place allows developers to not only accurately maintain assumptions about what is happening to that data both inside and outside the component, but to centralize the validation process as well. If code is spread out in multiple locations, it is difficult to verify assumptions.

2. Question any assumptions regarding the validity and correctness of data until the handling and verification code validates the data.

   In a design environment, a designer makes certain assumptions of the code and the data. Question the assumption that a component and process has complete control over the data. Question the assumptions about what the user inputs into the system.

3. Validate all input.

   For example, when asking the user for a first and last name, an attacker might input:

   <SCRIPT>MsgBox "Wazzup!"</SCRIPT>

   The characters !, /, <, >, ", and SCRIPT are not part of people's names. A better method of addressing user input is to check for what is valid.

   Watch for user input that is later displayed as output. This approach also helps to protect against cross-site scripting attacks which can operate through firewalls and access data via the client web browser. [Ref 35]

   Ensure that the data is the correct length. If 15 characters are required for a data field, then take only the first 15 characters and validate.

An attacker that compromises the system with bogus data causes Integrity and Availability issues. For more information on validating user input see [Ref 31, 12, 36].

3.2.2.2.7 Fail Securely

When a process, component or function call is unsuccessful perhaps because it failed to allocate memory, failed to load a module, or failed to lock a semaphore, it is important that the architecture provide the ability to handle the failure securely.

Keep failure handling code close to the failure as well; otherwise implement a central error handling routine which handles errors for a specific component.

For example, this simple piece of code has a security issue:

```
ChkReturnCode= CheckData(lpStrInputdata) ;
If ChkReturnCode = PASS then
        ‘ great, keep motoring
end if
```

While a simple check for failure is a step in the right direction, it is not enough. It is important to check each specific return code and handle each specific error appropriately.

Question the default behavior of a failure and ensure that it does not violate any security protocols already in place. Be careful what return codes the system lets the users see. It is possible for an error message to give away information that leads the attacker to other threat targets.

Failures are not necessarily an issue unless an attacker can take advantage of the failure to compromise Confidentiality, Integrity, or Availability. If a credit card based order processing system goes down for an entire day, it may be an inconvenience for customers. If the system does not divulge any credit card numbers or other personal data, then the system has preserved CIA.

For more information, see [Ref 12, 31, 37]

3.2.2.2.8 Using constants

The architect can build a software module containing constant integers, strings, and other static read-only variables. This allows designers to use the constants rather than hardcode values in the middle of code. If there is any issue with the constant or static, then it only needs to be changed in one place.

Hard-coding requires too many assumptions that attackers test. Avoid hard-coding and use constants and parameters whenever possible.

3.2.2.2.9 Using parameters

Designers also use allocated parameters for function calls as opposed to passing data directly in function calls. Function calls copy the data for use within the function rather than a pointer to the data where practical and where possible. If the memory is freed before the function completes, it may cause a segmentation fault or pointer access fault.

The same concept of using parameters also defends against SQL injection attacks. Hand-constructing SQL statements in code combined with invalid user input data causes huge problems. By using a parameterized query, the database can help

validate the input and return an error before the SQL statement executes. The strong typing in the SQL language and the database will return an error if the data does not conform. SQL statements can be expensive; using parameters can improve performance.

Using parameters helps preserve data Integrity and Availability. See [Ref 12, 31, 38, 36] for more information.

3.2.2.3 Wrapping it up

Throughout the SDLC, it is important for the vendor to keep accurate, detailed, and up to date documentation regarding the TRA. This allows managers and designers alike to understand the status of vulnerabilities and mitigation techniques.

3.3 Phase 3 - Software Coding

In this phase, software designers write the code according to the functional requirements and the architecture of the system. Designers also test and submit code from the debugging load-build to the production load-build.

3.3.1 Security in Phase 3

3.3.1.1 Own the Code

Software architects structure the code so that certain key designers and designated security SMEs take ownership of a particular area of code.

Architects structure architectural and coding dependencies such that these areas can compile on their own without too many unnecessary 'includes', 'uses', or references. As a result, the code compiles on its own without affecting too many other users.

This also allows code owners to question function calls and data usage from modules outside their own code.

Many design and debugging tools now exist in order to determine how function calls and data trace through the system.

3.3.1.2 Write Code and Unit Test

Designers write code and consult security SMEs whose code is affected when the code is submitted. The designer also provides full comments in the code, especially comments related to security.

At this time, the designer is working with a debug private version of a software build. The designer compiles and tests the code in their own environment without affecting the production build.

The designer can also conduct attacks against their own code to test the mitigation techniques, but this can only be done with known vulnerabilities.

Designers then inform the design team when the code is ready for code review.

3.3.1.3 Code reviews

Designers who want to submit code into the production build must first hold code reviews. Designers can submit small blocks of code to allow for modifications and to allow for time to unit test the code for security vulnerabilities. Security SMEs who own code affected by a code submission are present at the code review and conduct extra security checks. Once the code owners are satisfied that designers achieve 100% code coverage, they can allow the code submission into the production build.

If the designer claims that 100% code coverage is not possible, then the designer documents those non-reproducible cases directly in the code and other design documentation.

3.3.1.4 Code Submission and Revision control systems

The designer does not submit code without the express authorization of the code owner.

Once a designer submits code into the production build, the load-builder compiles the production build on a regular basis. Architects must insist on implementing a revision control system. The load-builder can back out code submissions without too much interruption if there is load-build compile issue.

3.3.1.5 Metrics and a Post Mortem

Keep a track of security issues discovered both before and after code submission. For each security issue, conduct a post mortem to determine how and why the security issue happened. Use this data to further educate the staff and to prevent issues from occurring again.

The issues should also go into the issues database so that program managers, design and test managers, and architects can later dive into the issues database looking for security issues.

The issues database also holds information regarding unsolved issues for later resolution.

3.3.1.6 Encouragement

When designers spend time to go through the design and discover a vulnerability, provide rewards. Immediate returns like these encourage fixing security issues and encourage the project managers to allow designers to fix security issues throughout the SDLC.

3.4 Phase 4 - Test and verification

In this phase, the vendor tests the system to ensure that it fulfills the customer's requirements as well as the functional requirements.

3.4.1 Security in Phase 4

The test organization tests that the mitigation techniques are working against threats, and lead post-mortem activities including identifying how and why security issues occur.

3.4.1.1 Test Mitigation Techniques

In phase 4, the tester's role is to prove that mitigation techniques designed into the system are working against the documented threats in the TRA. The tester has permission to test against the vulnerabilities discovered in sections 3.2.1.1 and 3.2.1.2. Test plans written in Phase 2 are an important part of the testing.

The test organization uses the same inputs to test plans as did the software architect in phase 2. With this approach, the test organization actively participates in the earlier phases of the SDLC.

Similar to the fact that it is not possible to make a system 100% secure, it is also not possible that testing will uncover every possible security issue. By keeping the testing process in mind from phase 1 and 2, it is possible to design a more secure software system.

It is important to keep test managers and testers in the loop throughout the SDLC. Undoubtedly in this phase, testing will uncover issues that may or may not be fixed before the system 'goes live'. At this time, it is important to rank the threats and the security issues and take the time to fix the most serious issues first.

See Ref [12, 31] for more information on security testing in the SDLC.

3.5 Phase 5 - Deployment and Maintenance

In Phase 5, the vendor ships the software system to the customer. The vendor installs the system and it 'goes live'.

3.5.1 Security in Phase 5

Undoubtedly in this phase, issues arise. The vendor ranks the security issues by working with the customer to outline the nature of the security issue, the circumstances under which the issue occurs, and a response plan. The architect conducts a threat analysis, determines if an exploit exists for the vulnerability, and ranks the risk.

If the risk is great, then the designer who wrote the code fixes the security issue. This is only possible with strict revision control systems and code ownership.

Management need not reprimand the designer for security issues. Designers cannot be aware of everything that an attacker might try to do when a system goes live, but if there is no ownership of the issue, then it is impossible for designers to learn from events and actions in the field.

Designers are eager and willing to learn, therefore, it is important to provide that learning aspect in later phases of the SDLC. Allowing designers to meet with the customer and learn from the customer's business allows designers to think about the customer's needs. This also helps design security into the system from the beginning of the SDLC.

See [Ref 19] for more general information on security in the SDLC.

4 Closing Remarks

Ultimately, it is not possible to ship a 100% secure software system. It will never ship! But, it is possible to make a system more secure from the new threats and vulnerabilities in the field which poses risk to creating new business opportunities.

If a vendor initiates a change management process, then a security sub-culture within the organization forms and people begin to learn to take security seriously. By committing and being realistic to the risk determination process, and by taking action to mitigate that risk, a vendor can realize a competitive advantage and gain a stronger footing in this new business marketplace.

Security does not reduce the performance of a system, but rather enhances the performance by ensuring it safeguards Confidentiality, Integrity, and Availability of the assets.

5 Glossary

All definitions taken from [Ref 14].

Confidentiality	Confidentiality is the privacy, secrecy, or nondisclosure of information except to authorized individuals.[12]
Integrity	From a user's or application owner's perspective, integrity is the quality of data that is based on attributes such as accuracy and completeness. From a system's or operation's perspective, integrity is the quality of data that it is only changed in an authorized manner or that the system/software/process does what it is supposed to do and nothing more.[13]
Availability	Availability is the state when data or a system is in the place needed by the user, at the time the user needs it, and in the form needed by the user.[14]

6 List of References

The complete list of references consulted for this paper. They will assist the reader to become familiar with security issues.

1. Gates, Bill. "Building Trust in Technology." Microsoft PressPass. 23 January 2003. URL: http://www.microsoft.com/presspass/ofnote/01-03davos.asp (20 December 2003).
2. Todd, Matthew. "Worms as Attack Vectors: Theory, Threats, and Defenses." SANS Reading Room. 31 January 2003. URL: http://www.sans.org/rr/papers/index.php?id=930 (20 December 2003).
3. Gibson, Steve. "The Strange Tale of the Denial of Service Attacks Against GRC.COM." 6 October 2003. URL: http://www.grc.com/dos/grcdos.htm (20 December 2003).
4. Mitnick, Kevin. The Art of Deception: Controlling the Human Element of Security. John Wiley & Sons Canada, Ltd., 2003.
5. Gragg, David. "A Multi-Level Defense Against Social Engineering." SANS Reading Room. December 2002. URL: http://www.sans.org/rr/papers/index.php?id=920 (20 December 2003).
6. Schmidt, Charles and Darby, Tom. "The What, Why, and How of the 1988 Internet Worm." July 2001. URL: http://www.snowplow.org/tom/worm/worm.html (20 December 2003).
7. Suppa, Carly. "Computer theft hits CCRA." Computer World Canada 17 October 2003 (2003): 1, 3.
8. Mann, Charles. "Why software is so bad and how to fix it." MIT Technology Review July/August 2002 (2002): 33-38.
9. Reuters. "Microsoft Sued for Security Breaches." MSNBC Technology & Science Hacks, Viruses, Scams & Spam. 2 October 2003. URL : http://msnbc.msn.com/id/3131161/ (20 December 2003).
10. Wilson, Mark and Hash, Joan. "Building an Information Technology Security Awareness and Training Program." National Institute of Standards and Technology. October 2003. URL: http://csrc.nist.gov/publications/nistpubs/800-50/NIST-SP800-50.pdf (20 December 2003).
11. Linger, Richard and Lipson, Howard and McHugh, John and Mead, Nancy and Sledge, Carol. "Life-Cycle Models for Survivable Systems." CMU-SEI CERT® Coordination Center. October 2002. URL: http://www.cert.org/archive/pdf/02tr026.pdf (20 December 2003).
12. Howard, Michael and LeBlanc, David. Writing Secure Code, 2nd Edition. Microsoft Press, 2003.
13. Brooks, Fred. The Mythical Man Month: Essays on Software Engineering. Addison Wesley Professional, 1995.
14. Grance, Tim and Hash, Joan and Stevens, Marc. "Security Considerations in the Information System Development Life Cycle." National Institute of Standards and Technology. October 2003.

URL: http://csrc.nist.gov/publications/nistpubs/800-64/NIST-SP800-64.pdf (20 December 2003).

15. Stoneburner, Gary and Goguen, Alice and Feringa, Alexis. "Risk Management Guide for Information Technology Systems." National Institute of Standards and Technology. October 2001. URL: http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf (20 December 2003).
16. Communications Security Establishment. "Threat and Risk Assessment Working Guide." Government of Canada Publications IT Security Guidance. October 1999. URL: "http://www.cse-cst.gc.ca/en/documents/knowledge_centre/gov_publications/itsg/itsg04.pdf (20 December 2003).
17. Communications Security Establishment. "A Guide to Security Risk Management for Information Technology Systems." Government of Canada Publications IT Security Guidance. 1996. URL: http://www.cse-cst.gc.ca/en/documents/knowledge_centre/gov_publications/itsg/mg2.pdf (20 December 2003).
18. Communications Security Establishment. "A Guide to RISK ASSESSMENT AND SAFEGUARD SELECTION for Information Technology Systems." Government of Canada Publications IT Security Guidance. 1996. URL: http://www.cse-cst.gc.ca/en/documents/knowledge_centre/gov_publications/itsg/mg3.pdf (20 December 2003).
19. Software Development Magazine. Security. URL: http://www.sdmagazine.com/security/ (20 December 2003).
20. McGraw, Gary and Viega, John. "The Weakest Link." Software Development Magazine. Security. December 2002. URL: http://www.sdmagazine.com/documents/s=818/sdm0212e/ (20 December 2003).
21. McGraw, Gary and Viega, John. "The One-Click Trick." Software Development Magazine. Security. June 2002. URL: http://www.sdmagazine.com/documents/s=818/sdm0306c/ (20 December 2003).
22. Donaldson, Mark. "Inside the Buffer Overflow Attack: Mechanism, Method, & Prevention." SANS Reading Room. 3 April 2002. URL: http://www.sans.org/rr/papers/index.php?id=386 (20 December 2003).
23. Nash, Andrew and Brink, Derek and Duane, William and Joseph, Celia. PKI: Implementing & Managing E-Security. Osborne/McGraw Hill Media Group, 2001.
24. Evers, Joris. "Security suit against Microsoft could turn huge." Infoworld. 2 October 2003. URL: http://infoworld.com/article/03/10/02/HNmssecsuit_1.html (20 December 2003).
25. Rist, Oliver. "Microsoft sued over security? No surprises here." Infoworld. 10 October 2003. URL: http://www.infoworld.com/article/03/10/10/40enterwin_1.html (20 December 2003).
26. Filkins, Barbara. "Getting Started: The Impacts of Privacy and Security Under HIPAA - A Case Study." SANS Reading Room. 10 July 2003. URL: http://www.sans.org/rr/papers/index.php?id=1214 (20 December 2003).

27. Borkin, Sheldon. "The HIPAA Final Security Standards and ISO/IEC 17799." SANS Reading Room. 15 July 2003. URL: http://www.sans.org/rr/papers/index.php?id=1193 (20 December 2003).
28. Government of Canada. "Privacy Legislation." Government of Canada. Privacy Commissioner of Canada. 20 December 2003. URL: http://www.privcom.gc.ca/legislation/index_e.asp (20 December 2003).
29. McGraw, Gary and Viega, John. "Risk Analysis: Attack Trees and Other Tricks." Software Development Magazine. Security. August 2002. URL: http://www.sdmagazine.com/documents/s=818/sdm0208a/ (20 December 2003).
30. Amoroso, Edward. Fundamentals of Computer Security Technology. Prentice Hall Canada, 1994.
31. McGraw, Gary and Viega, John. Building Secure Software: How to Avoid Security Problems the Right Way. Addison-Wesley, 2001.
32. "Octave." CMU-SEI CERT® Coordination Center. URL: http://www.cert.org/octave/ (20 December 2003).
33. Raynal, Frederic and Blaess, Christophe and Grenier, Christophe. "Avoiding security holes when developing an application - Part 3 : buffer overflows." May 2001. URL: http://www.linuxfocus.org/English/May2001/article190.shtml (20 December 2003).
34. McGraw, Gary and Viega, John. "Less Is More." Software Development Magazine. Security. March 2003. URL: http://www.sdmagazine.com/documents/s=818/sdm0303f/ (20 December 2003).
35. Stephens, Cheryl. "Cross-Site Tracing: Protecting Businesses from a Simple Attack." SANS Reading Room. 1 June 2003. URL: http://www.sans.org/rr/papers/index.php?id=1140 (20 December 2003).
36. Hurlbut, Robert. "Robert Hurlbut's SQL Server Blog." 28 September 2003. URL: http://sqljunkies.com/weblog/rhurlbut/posts/243.aspx (20 December 2003.)
37. McGraw, Gary and Viega, John. "Failing Safely." Software Development Magazine. Security. February 2003. URL: http://www.sdmagazine.com/documents/s=818/sdm0302g/ (20 December 2003).
38. Sonnemans, Fons. "SQL-strings considered harmful." Reflection it. 10 March 2003. URL: http://www.reflectionit.nl/SqlInsert.aspx (20 December 2003).

7 Endnotes

[1] Suppa, p.1, 3.

[2] Schmidt, Charles and Darby, Tom. "The What, Why, and How of the 1988 Internet Worm." July 2001. URL: http://www.snowplow.org/tom/worm/worm.html (20 December 2003).

[3] Wilson and Hash, p.4.

[4] see note 3

[5] see note 3

[6] paraphrased from Howard and LeBlanc, p.40.

[7] Mann, p.36.

[8] Grance Hash and Stevens, p.10.

[9] McGraw, Gary and Viega, John. "Risk Analysis: Attack Trees and Other Tricks." Software Development Magazine. Security. August 2002. URL: http://www.sdmagazine.com/documents/s=818/sdm0208a/ (20 December 2003).

[10] McGraw, Gary and Viega, John. "The Weakest Link." Software Development Magazine. Security. December 2002. URL: http://www.sdmagazine.com/documents/s=818/sdm0212e/ (20 December 2003).

[11] McGraw, Gary and Viega, John. "The One-Click Trick." Software Development Magazine. Security. June 2002. URL: http://www.sdmagazine.com/documents/s=818/sdm0306c/ (20 December 2003).

[12] Grance Hash and Stevens, p.10.

[13] Grance Hash and Stevens, p.10.

[14] Grance Hash and Stevens, p.10.