Summer Workshop on Cyber Security

Software Security

Dr. Akbar Namin

Texas Tech University
August 12-16, 2013

Supported by National Science Foundation
Secure and Resilient Software Development
Software is ubiquitous, even in places you wouldn’t imagine. Software is so seamlessly interwoven into the fabric of modern living that it fades into the background without notice. We interact with software not only on home or office computers, but in our routine everyday activities – as we drive to the office in our cars as we withdraw cash from ATM, and even when we listen to music.
1. How Does Software Fail Thee?
Let us Count the Ways

Software is not used just by a small cross section of the modern-day society-the entire population depends on it. Airlines, banks, telecommunication companies, hospitals, supermarkets, gas stations, voting infrastructures, and countless other institution rely on software.

At this stage of technological innovation, we’ve come to realize that software must not only function properly but also be available to us at all times and in all places so that we can continue to thrive in the digital ways of life to which we’ve grown accustomed.
While we’d like to believe that software is as reliable as it need to be, reality proves us wrong every time. We’ll examine what makes software fragile, brittle, and resistant to reliability and resilience. What we refer to as software resilience is an adaption of the National Infrastructure Advisory Council (NIAC) definition infrastructure resilience:

Software resilience is the ability to reduce the magnitude and/or duration of disruptive events. The effectiveness of a resilient application or infrastructure software depends upon its ability to anticipate, absorb, adapt to, and/or rapidly recover from a potentially disruptive event.
1.1 Vulnerabilities Abound

- A vulnerability, in the context of software, is a defect in the implementation that opens a pathway for an attacker with the right set of skills to exploit the defect and cause the software to behave in ways the software developer never anticipated.

- A vulnerability in computer security, a weakness which allows an attacker to reduce a system’s information assurance. Vulnerability is the intersection of three elements: a system susceptibility or flaw, attacker access to the flaw, to be vulnerable, an attacker must have at least one applicable tool or technique that can connect to a system weakness.
1.1.1 Security Flaws Are Omnipresent

Just as software is everywhere, flaws in most of that software are everywhere too. Flaws in software can threaten the security and safety of the very systems on which they operate. These flaws are not present just in the traditional computers we think of, but also in critical devices that we use, such as our cell phones and cars and hospital equipments, etc.

Ex: Owners of iPhone can use the optional passcode lock feature of their phone to protect all the confidential documents and personal information on it. However, anyone who has obtained a protected stolen iPhone can easily bypass the lock and gain access to all the persona data on the phone just by double clicking the home button on the phone. Apple did fix this issue on later version of the iPhone operating system.
1.1.2 Cars Have Their Share of Computer Problems Too

In 2005, several thousand Toyota Prius cars, fuel-efficient hybrid vehicles, had to be recalled for safety purposes. Toyota did not immediately comment on this problem, but in February 2010 the company announced a recall of 400,000 2010 Priuses for a problem that involved the cars’ antilock brakes. Toyota said it was a software glitch. Subsequently, Toyota extended the recalls to cover millions of its vehicles, including addressing an apparent software glitch that did cause sudden acceleration.
From where do these problems emanate? Programmers writing custom code for corporation to use internally or on the internet, programmers working at software development companies that produce commercial off-the-shelf programs, programmers working in the public domain, all suffer from the same fundamental problem: They don’t know any better because they were *never taught how to write* secure and resilient programs.
Writing software, like driving a car, is a habit. Until someone teaches us how to drive safely, we don’t personally know the dangerous of driving and the skills needed to prevent or avoid accidents. Cars often have safety mechanisms built into them, but as drivers, we have to consciously use our own safe driving skills. Experience teaches us that we are better off instilling safe driving skills before we let people loose on the roads, since their first accident may be their last.
1.3 What Are the True Costs of Insecure Software to Global Enterprises

Every year, organizations across the globe spend millions of dollars on securing their software infrastructure. Security spending focuses on detecting existing vulnerabilities in the software that organizations already own and on finding ways to reduce the risks associated with using it. Rewriting software, whether to fix a problem or fundamentally change what the software does, also results in tremendous corporate expenditures every year. Losses in productivity when an application or a system goes down also results in direct or indirect losses to the business.
Ex: A hacker penetrated a server at payment processor RBS WorldPay and later stole nearly $9 million from ATMs. In the ATM attacks, the attackers had compromised account numbers and other magnetic-stripe data needed to clone debit cards. More than 130 ATMs in 49 cities from Moscow to Atlanta were hit simultaneously just after midnight Eastern time on November 8, 2008. Customers have since filed a class-action lawsuit against RBS WorldPay. Hundreds of millions of dollars are lost every year and account for direct financial losses because of fraudulent and computer-crime activities.
1.4 Addressing Security Questions

Addresses resilience

Experts define *information security* as the ability to protect the confidentiality, integrity, and availability of the information.

**Confidentiality** is the goal of being assured that information is protected from being accessed by unauthorized users.

**Integrity** is concerned with implementing controls to ensure that information cannot be modified without proper authorization and that stored information cannot be tampered with. It is applicable for information at rest or when in transit. Usually, techniques such as hashing and digital signatures are used to protect the integrity of information.
Availability is making sure that the information is made available to authorized users whenever they need it. This is a complicated problem that requires extensive planning and testing. Integrating security into the software development life cycle is the key to eliminate current problems once and for all.
Characteristics of Secure and Resilient Software

2.1 Functional VS Nonfunctional Requirements

Software is useful for what it does. People purchase it fulfills their need to perform some function. Functions and features are the reasons terms that people purchase or pay for the development of software.

What software is expected to “do” is described by users in what are called *Functional requirements*. *Nonfunctional requirements* are the quality, security, and resiliency aspect of software that only show up in requirements documents when they’re deliberately
Here’s another definition of requirements, from the Institute of Electronics and Electrical Engineering (IEEE) Software Engineering Glossary.

*Functional requirement*: A system or software requirement that specifies a function that a system/software or system/software component must be capable of performance. These are software requirements that define system behavior, the fundamental process that the system’s software and hardware components perform on inputs to produce output.
Nonfunctional requirements: A software requirement that describes not what the software will do but how the software will do it. Nonfunctional requirements are sometimes difficult to test, so they are usually evaluated subjectively.
2.2 testing Nonfunctional Requirements

Software testing that focusing only on functionality testing for user acceptance can uncover errors (bugs or flaws) in how the software operates.

Resilience and security testing flips the problem of user acceptance testing on its head. Resilience tests not only verify that the function designed to meet a nonfunctional requirement or service (e.g., security function) operate as expected, it also validates that the implementation of those functions is not flawed or haphazard.
2.3 families of Nonfunctional Requirements

They are listed alphabetically, not order of importance:
- Availability
- Capacity
- Efficiency
- Extensibility
- Interoperability
- Manageability
- Maintainability
- Performance
- Portability
Privacy
Recoverability
Reliability
Scalability
Security
Serviceability
2.4 Availability

Availability Levels and Measurements:

- **High availability** – The system or application is available during specified operating hours with no unplanned outages.

- **Continuous operations** – The system or application available 24 hours a day, 7 days a week, with no scheduled outages.

- **Continuous availability** - The system or application available 24 hours a day, 7 days a week, with no planned or unplanned outages.
2.5 Capacity

Capacity planning is made far simpler when runtime environments can be changed on the fly to accommodate changes in user traffic, changes in hardware, and other runtime-related considerations.
2.6 Efficiency

Efficiency refers to the degree that a system uses scarce computational resources, such as CPU cycles, memory, disk space, buffers and communication channels. Efficiency can be characterized using these dimensions:

- **Capacity** – Maximum number of users or transactions.
- **Degradation of service** – The effects of a system with capacity of $X$ transactions per time when the system receives $X+1$ transactions in the same period of time.
2.7 Interoperability

Interoperability is the ability of a system to work with other system or software from other developers without any special effort on the part of the user, the implementers, or the support personnel.

Interoperability can only be implemented when everyone involved in the development process adheres to common standards.

Interoperability requirements should dictate what standards must be applied to these elements, and how the designers and developers can get their hands on them to enable compliant application software.
2.8 Manageability

Manageability allows support personnel to move the application around available hardware as needed or run the software in virtual machine.

Manageability features require designers and developers to build software as a highly cohesive and loosely coupled.

Coupling and cohesive are used as software quality metrics.
2.9 Cohesion

Cohesion is increased when the responsibilities (methods) of a software module have many common aspects and are focused on a single subject, and when these methods can be carried out across a variety of unrelated sets of data. Low cohesion can lead to the following problems:

- Increased difficulty in understanding the modules
- Increased difficulty in maintaining a system, because logical changes in the domain may affect multiple modules, and because changes in related modules.
- Increased difficulty in reusing a module, because most applications won’t need the extraneous sets operations that the module provide.
2.10 Coupling

Strong coupling happens when a dependent class contain a pointer directly to a concrete class that offers the required behavior (method). Loose coupling occurs when dependent class contains a pointer only to an interface, loose coupling provides extensibility and manageability to a designs.
2.11 Maintainability

Software maintenance refers to the modification of a software application after delivery, to correct faults, improve performance or other attributes, or adapt the product to modified environment.

Software maintenance is expensive and time-consuming aspect of development.

Maintenance is made more difficult if the original developers leave the application behind, with little or no documentation for the maintenance programmers.
Software Maintenance Maturity Model (SMmm) was developed to address the assessment and improvement of the software maintenance function by proposing a maturity model for daily software maintenance activities.
Performance (sometimes called quality –of-service) requirements generally address three areas:

- Speed of processing a transaction (e.g. response time)
- Volume of simultaneous transactions (e.g. the system must be able to handle at least 1,000 transactions per second)
- Number of simultaneous users (e.g. the system must be able to handle a minimum of 50 concurrent user sessions)

The end users of the determine these requirements, and they must be clearly documented if there’s to be any hope of meeting them.
2.13 Portability

Portability is the process of adapting software so that an executable program can be created for a computing environment that is different from the one for which it was originally designed. Also used in general way to refer to the changing of software /hardware to make them usable in different environments. Portability is a key issue for development cost reduction, and sufficient time must be allowed to determine the optimal languages and development environments needed to meet the requirement without the risk of developing different environment.
2.14 Privacy

Privacy is related to security in that many privacy controls are implemented as security controls, but privacy include nonsecurity aspects of data collection and use.

The principles for collecting information:

1. **Notice/Awareness** - In general, a website should tell the user how it collects and handles user information.

2. **Choice/Consent** – Website must give consumers control over how their personally identifying information is used.

3. **Access/Participation** – Users should be able to review, correct, and in some cases delete personally identifying information on particular website.

4. **Security/Integrity** – website must do more than reassure users that their information is secure with a "feel-good" policy statement.
2.15 Recoverability

*Recoverability* is related to reliability and availability, but is extended to include requirements on how quickly application must be restored in the event of a disaster, unexpected outage, or failure of a dependent system.
2.16 Reliability

Reliability may be defined in several ways:
- The capacity of a device or system to perform as designed
- The resistance to failure of a device or system
- The ability of a device or system to perform a required function under stated conditions for a specified interval of time
- The probability that a functional unit will perform its required function for a specified interval under stated condition
- The ability of something to “fail well”
2.17 Scalability

Scalability is the ability of a system to grow in its capacity to meet the rising demand for its service offered and its related to capacity NFRs. System Scalability criteria might include the ability to accommodate increasing number of:

- Users
- Transactions per second
- Number of database that can run and provide results simultaneously
2.18 Security

Security NFRs are needed to preserve the goals of confidentiality, integrity, and availability.

Here are just a few objectives that are needed for software that’s expected to be secure and resilient:

1. Ensure that users and client applications are identified and that their identities are properly verified.
2. Ensure that all access or modify data are logged and tracked.
3. Ensure users and client applications can only access data and service for which they have been properly authorized.
4. Ensure that communications and data are not intentionally corrupted.
5. Ensure that confidential communications and data are kept private.
6. Ensure that application can survive an attack, or fail securely.
To ensure that these objectivities will be met, you’ll need to document specific and detailed security requirements for the following:

- Identification requirements
- Authentication requirements
- Authorization requirements
- Immunity requirements
- Integrity requirements
- Intrusion-detection requirements
- Nonrepudiation requirements
- Privacy requirements
- Security audition requirements
- System maintenance security requirements
Serviceability and Supportability refer to the ability of application support personnel to install, configure, and monitor computer software, identify expectations of faults, debug or isolate faults to perform root-cause analysis, and provide hardware or software maintenance to aid in solving a problem and restoring the software to service.
Some examples of requirements that facilities serviceability and supportability:

I. Help desk notification of exceptional events
II. Network monitoring
III. Standardized documentation tools and processes
IV. Event logging
V. Logging of program state
VI. Procedure entry and exit with input and return variable states
## 2.20 Characteristic of Good Requirements

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cohesive</td>
<td>The requirement addresses one and only one thing</td>
</tr>
<tr>
<td>Complete</td>
<td>The requirement is fully stated in one place with no missing information</td>
</tr>
<tr>
<td>Consistent</td>
<td>The requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation</td>
</tr>
<tr>
<td>Correct</td>
<td>The requirement meets all or part of a business or resilience need as authoritative stated by stakeholders</td>
</tr>
<tr>
<td>Current</td>
<td>The requirement has not been made obsolete by the passage of time</td>
</tr>
<tr>
<td>Externally observable</td>
<td>The requirement specifies a characteristic of the product that is externally observable or experienced by the user</td>
</tr>
<tr>
<td>Characteristic</td>
<td>Explanation</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Feasible</td>
<td>The requirement can be implemented within the constraints of the project</td>
</tr>
<tr>
<td>Mandatory</td>
<td>The requirement represents a stakeholders-defined characteristic or constraint.</td>
</tr>
<tr>
<td>Unambiguous</td>
<td>The requirement is stated concisely, without unnecessary technical jargon acronyms, or other esoteric terms or concepts.</td>
</tr>
</tbody>
</table>
2.21 Eliciting Nonfunctional Requirements

Different methodologies dictate differing documentation techniques for requirements gathering and analysis. Fans of the Unified Modeling Language and Rational Unified Process are very familiar with the documentation tool called use cases to capture functional requirements but may find that they are not well suited for capturing NFRs.
2.22 Documenting Nonfunctional Requirements

- Nonfunctional requirements may be documented in any form that suits the development process in use but should be standardized across all development terms and should be included in all analysis and design documentation.
Security and Resilience in the Software Development Life Cycle

3.1 Resilience and Security Being from Within

The only reliable way to ensure that software is constructed secure and resilient is by integrating a security and resilience mindset and process throughout the entire software development life cycle (SDLC).

The security processes is often just “common sense” improvements and any organization can and should adopt them into its existing environment.
Figure 3.1 provides a high-level overview of the fundamental security and resilience processes that should be integrated into the various SDLC phases, from requirements gathering to deployment and beyond. Each process yields its own findings, and recommendations are prepared to make appropriate change to design, architecture, source code, use of third-party components, development configurations, and other considerations to help understand and reduce risk down to an acceptable level.
Security in SDLC

Requirements
- Threat Modeling
- Security Design Review

Design
- Static Analysis
- Peer Review

Development
- Security Test Cases
- Dynamic Analysis

Test
- Final Security Review
- Application Security & Monitoring Response Plan

Deployment

Additional information:
Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
Here you will find guidance that you should consider implementing for each phase of development:

- Requirements gathering and analysis.
- System detail designs.
- Application coding and reviews.
- Testing steps.
- Deployment steps.
3.2 Requirements Gathering and Analysis

The key activities during the requirements gathering and analysis phase are intended to map out and document the nonfunctional requirements (NFRs) for the system under development.

To be effective, business systems analysts and systems designers should be sure they are very familiar with the environment in which they are operating, by reviewing and maintaining their knowledge about:

i. Organizational security policies and standards.
ii. Organizational privacy policy.
iii. Regulatory requirements.
iv. Other relevant industry standards.
The NFRs are then mapped against the critical security and resilience goals of:

- Confidentiality and privacy.
- Integrity.
- Availability.
- Nonrepudiation.
- Auditing.

See figure 3.2 for an example of this type of mapping.
Requirements Phase

Key Input

- Organizational requirements
- Privacy requirements
- Statutory requirements
- Industry requirements

Key Deliverable

- Prioritized security & privacy requirements
- Key security-related design goals

Map Security & Privacy requirements

Summer Workshop on Cyber Security August 12 - 16, 2013 – Software Security, TTU
3.3 System Design and Detailed Design

Threat modeling and design reviews are the two major resilience processes that you will encounter during the design phase. There are two classes of vulnerabilities: design-related and implementation-related vulnerabilities. Detailed threat modeling is an excellent way to determine the technical security posture of an application to be developed or under development. It consists of four key steps:

• Functional decomposition
• Categorizing threats
• Ranking threats
• Mitigation planning
3.3.1 Functional Decomposition

Functional Decomposition is typically performed using data flow diagrams.

The key is to understand the boundaries of untrusted and trusted components.
3.3.2 Categorizing Threats

STRIDE is a framework developed by Microsoft for classifying threats. The different threat categories used are:

- **Spoofing of user identity**: An example, illegally accessing and then using another user’s authentication information.
- **Tampering**: Involves the malicious modification of data
- **Repudiation**: Threats here are associated with users who deny performing an action without other parties having any way an to prove. Nonrepudiation refers to the ability of a system to counter repudiation threats.
Information disclosure: Threats in this area involve the exposure of information to individuals who are not supposed to have access to it.

Denial of service: Denial of service (DoS) attacks deny service to valid users. You must protect against certain types of DoS threats simply to improve system availability and reliability.

Elevation of privilege: Include those in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed.
3.3.3 Ranking Threats

Ranking potential threats for a software system requires a fair amount of subjective judgment. DREAD is a model developed by Microsoft. We arrive rating by asking the following question:

- **Damage potential**: How great is the damage if the vulnerability is exploited?
- **Reproducibility**: How easy is it to reproduce the attack?
- **Exploitability**: How easy is it to launch an attack?
- **Affected users**: As a rough percentage, how many users are affected?
- **Discoverability**: How easy is it to find the vulnerability?
3.3.4 Mitigation Planning

With a list of ranked threats, you can document a high-level mitigation plan by mapping them to the potential vulnerability in the software system.
3.4 Design Reviews

Threat modeling ad design reviews can leverage commercial off-the-shelf tools, custom in-house software, or even simple checklists. Personnel must use their best judgment based on the environment, the organizational, the organizational structure, and existing processes and practices.

See Figure 3.3 for the major steps in the design phase.
Design Phase

Key Inputs
- Inputs from previous phases
- Design principles
- Data flow diagrams
- Technical and nontechnical security control requirements

Key Deliverables
- Well categorized and ranking threats
- High level mitigation plan
- Security architecture and design document

Threat Modeling
Security Design Review

Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
3.5 Development (Coding) Phase

Activities in this phase often generate implementation-related vulnerabilities.

Static analysis and peer review are two key processes to mitigate or minimize these vulnerabilities.
3.5.1 Static Analysis

It involves the use of automated tools to find issues within the source code itself:

- Bug finding
- Style checks
- Type checks
- Security vulnerability review
3.5.2 Peer Review

A peer review is far more time-consuming than automated analysis, but it is an excellent control mechanism to ensure the quality and security of the code base.

Developers review each other’s code and provide feedback to the owners (original coders) of the different modules so they can make appropriate change to fix the flaws discovered during the review.
3.5.3 Unit Testing

Unit testing is another key process that many organizations fail to perform regularly but is important from security and resilience perspective. Unit testing helps to prevent bugs and flaws from reaching the testing phase.

See Figure 3.4 for diagram of the security activities in the development phase.
Development Phase

Key Inputs

- Inputs from previous phases
- Source code
- Source coding standard(s)
- Source configuration standard(s)
- Unit test case

Static Analysis

Peer review

Key Deliverables

- Vulnerabilities from automated analysis
- Vulnerabilities from peer review and unit testing

Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
3.6 Testing

The test phase is critical for discovering vulnerability that were not discovered and fixed earlier.
The security team uses all the assumptions and business processes captured to create several security test cases.
The software is loaded and operated in the test environment and tested against each of the test case.
Dynamic analysis consist of using automated tools to test for security vulnerabilities.
These tests are iterative in nature and result in a list of vulnerabilities that are then errors and sends the ranked for and prioritized.
See Figure 3.5 for a diagram of the security steps in the test phase of the SDLC.
Test Phase

Key Inputs

- Inputs from previous phases
- Requirements documentation
- Software deployed in test environment

Key Deliverables

- Security test cases document
- Prioritized list of Vulnerabilities from automated and manual analysis

Security Test Cases

Dynamic analysis
3.7 Deployment

The deployment phase is the final phase of the SDLC, when the software is installed and configured in the production environment and made ready for use its intended audience.

A key part of changes advisory board (CAB). A CAB offers the multiple perspectives necessary to ensure good decision making.

A CAB is an integral part of defined change management process designed to balance the need for change with the need to minimize inherent risks.
During the deployment phase,

- Security subject-matter experts who may or may not be part of the change advisory board perform a final security review to ensure that the security risks identified during all the previous phases have been fixed or have a mitigation plan in place.

- The development team coordinates with the release management and production support teams to create an application security monitoring and response plan.

See Figure 3.6 for the security activities in the deployment phase of the SDLC.
Deployment Phase

Key Inputs

• Inputs from previous phases
• Finalized application ready to be deployed

Key Deliverables

• Security review sign-off
• Security Monitoring & Response plan

Final Security Review

Application Security Monitoring & Response plan

Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
3.8 Security Training

Even though training may not seem to fit directly into any particular SDLC phase, it plays an important role in improving the overall security and resilience of developed software.

Training should be a prerequisite for anyone who has a role anywhere in the software development environment.
Chapter 4
Proven Best Practice for Resilient Applications
4.1 Critical Concept

Entire network architectures consisting of firewalls, routers, and intrusion detection and prevention devices limited outside traffic to only supported protocols and services, while all others were blocked or prevented from entering the network.
Port 80 and Port 443 (SSL) on corporate firewalls are open wherever there’s a Web server or Web service available to public users or extranet. Since Web traffic is never blocked, attacks are structured and executed directly on Web applications that accept and respond to user inputs on Web from fields.

Web security is a universal programming problem, not a local operations or networking problem.

See Figure 4.1 for an illustration of the problem.
Hole in the Firewall

HTTP and/or HTTPS

FTP

Telnet

Web Traffic

Web Server

Application Server

Database

Server OS
4.2 The Security perimeter

A simple definition of the security perimeter is the border between the assets we want to protect and the outside world. Physical security controls are meant to prevent and/or deter attackers from accessing a private property without authorization. We can consider implementing several measures:

- A licked gate
- A fence or high wall around the property
- A security guard at the entrance
- Security cameras
- Automated security monitoring and alarm systems
This concept of a trusted and secured zone and security checked for whatever enters that zone is applicable to software and networks of today’s business.

Factors to define the ever-expanding security perimeter of an organization:

• Extranets and virtual private networks
• Globally telecommuting employees
• Mobile technologies
• Opening up of Web applications to public users
When the borders of enterprise network blur, it is difficult to rely on traditional security mechanisms to secure assets. The trust models and the security controls implemented to monitor and validate them are completely different and sophisticated.

The application has control over the elements that are inside the application perimeter:

i. Web server
ii. Application server
iii. Database server
The application has no control over the elements outside the application perimeter:

i. Web browsers
ii. Other applications
iii. External databases

The Web application is responsible for ensuring that the proper controls are in place to protect itself from malicious activity and is the last line of defense.
4.3 Attack Surface

A simple definition of an Attack Surface is all possible entry points that an attacker can use to attack the application or system under consideration.

In the case of Web application, the attack surface is defined by:

- All the Web pages the attacker can access—either directly or forcibly
- Every point at which the attacker can interact with the application
- Every function provided by the application
The exact attack surface depends on who the attacker is (internal VS external presence):

- Malicious application users may again access to unauthorized functionalities
- External attackers usually have limited access (unauthenticated areas of the application)
4.3.1 Mapping the Attack Surface

The attack surface is usually larger than a typical application developer or software architect imagines. In the case a Web application, the following techniques are often used.

- Crawl every page of the application (using an automated tool)
- Identify all the available functionalities:
  - Follow every link
  - Fill every form with valid/invalid data and submit.
Look for the points where the user can supply information to the application:
- GET requests with query strings parameters
- POST requests generated by forms
- HTTP headers
- Cookies
- Hidden parameters
4.3.2 Side Channel Attacks

Sometimes attackers target the implementation rather than the actual theoretical weakness in the system. Side channel attacks, are critical area for designers and developers who are designing and deploying secure hardware and software systems.

A simple example of a side channel attacks is the ‘timing attack’. Simple real-world example to help understand this type ‘timing attack’. If a person is asked to pick and retrieve different items, one at a time, at a supermarket, it’s possible measure the time it takes for each item to be brought back to determine the relative positions of the different areas of the store and to guess the location of other related items in the store.
4.4 Application Security and Resilience Principles

The principles are language-independent and architecturally neutral primitives that can be leveraged within most software development methodologies to design and construct applications. Principles are important because they help us make decision in new situations using the same basic ideas.
Following are 10 principles and best practices, adapted from Open Web Application Security Project (OWASP):

1. Apply defense in depth.
2. Use a positive security model.
3. Fail securely.
4. Run with least privilege.
5. Avoid security by obscurity.
7. Detect intrusions.
8. Don’t trust infrastructure.
9. Don’t trust services.
10. Establish secure defaults.
The principle of defense in depth emphasizes that security is increased markedly when it is implemented as a series of overlapping layers of controls and countermeasures that provide three elements needed to secure assets: prevention, detection, and response. In the software world, defense in depth dictates that you should layer security devices in series that protect, detect, and respond to likely attacks on the systems. Figure 4.2 from Cisco’s Unified Contact Center Enterprise (CCE) documentation, illustrates the concept of defense in depth.
Policies, procedures, and training

Physical Security
- Network Security
  - Perimeter Security
  - Internet network

Host Security
- Server Hardening
- Host-based Firewall
- Anti-virus/Spyware
- Intrusion Prevention
- Patch Management

Data Security
- Application & Data

Security policies, procedures, DR strategy, awareness and education

Firewalls, ACL configured routers, VPNs
Network segments, Network based IDS

OS hardening, authentication
Inbound TCP/IP port control
Anti-Virus/Anti-spyware Protection
Day-zero attack protection
Security update management

Endpoint security and secure communication paths (SSL, TLS, IPSec)

Strong passwords, ESAPI, etc

Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
4.6 Practice 2: Usr positive Security Model

The positive security model that is often called *whitelisting* defines what is allowable and rejects everything that fails to meet the criteria.

This positive model should be contrasted with a “negative” (or “blacklist”) security model, which defines what is disallowed, while implicitly allowing everything else.
The positive security model can be applied to a number of different application security areas:

- It should be applied to every field of input (hidden or not)
- Validation routines or frameworks should be implemented to specify the characteristic of input that allowed, as opposed to trying to filter out bad input.
- With access controls, the positive model will deny access to everything, and allow only access to specific authorized resources or functions.
4.7 Practice 3: Fail Security

Major types of errors require special attention:

- Exceptions that occur in the processing of a security control itself
- Exceptions in code that are not “security-relevant”

There are three possible outcomes from a security mechanism:

I. Allow the operation
II. Disallow the operation
III. Exception
In general, you should design your security mechanism so that failure will follow the same execution path as disallowing the operation.
The principles of least privilege recommends that user accounts have the least amount of privilege required to perform their basic business processes. This encompasses user rights and resource permissions such as:

- CPU limits
- Memory
- Network permissions
- File system permissions
4.9 Practice 5: Avoid Security by Obscurity

Security by obscurity, as its name implies, describes an attempt to maintain the security of a system or application based on the difficulty in finding or understanding the security mechanism within it.

An example of security by obscurity is a cryptographic system in which the developers wish to keep the algorithm that implements the cryptographic functions a secret rather than keeping the keys a secret and publishing the algorithm so that security researchers can determine if it is bullet-proof enough for common security uses.
4.10 Practice 6: Keep Security Simple

It means avoiding overly complex approaches to coding with what would otherwise be relatively straightforward and simple code for someone to read and understand.

One way is keep security simple is to break security functions and features down into these discrete objectives:

1. Keep services running and information away from attackers-related to deny access by default.
2. Allow the right users access to the right information –related to least privilege.
3. Defense every layer as if it were the last layer of defense-related to defense in depth.
4. Keep a record of all attempts to access information (logging)
5. Compartmentalize and isolate resources.
4.11 Practice 7: Detect Intrusions

Detecting intrusions in application software requires three elements:

- Capability to log security-relevant events
- Procedures to ensure that logs are monitored regularly
- Procedures to respond properly to an intrusion once it has been detected
4.11.1 Log All Security - Relevant Information

Sometimes you can detect a problem with software by reviewing the log entries that you can’t detect at runtime, but you must log enough information to make that possible and useful.

The logging functionality in the application should also provide a method of managing the logged information to prevent tampering or loss.
4.11.2 Ensure That the Logs Are Monitored Regularly

If a security analyst is unable to parse through the event logs to determine which events are actionable, then logging events provide little to no value.
4.11.3 Respond to Intrusions

Detecting intrusion is important because otherwise you give the attacker unlimited time to perfect an attack. If you detect intrusion perfectly, then an attacker will get only one attempt before he is detected and prevented from launching more attacks.
4.12 Practice 8: Don’t trust Infrastructure

Relying on a security process or function that may or may not be present is a sure way to have security problems. Make sure that your application’s security requirements are explicitly provided though application code or through explicit invocation of reusable security functions provided to application developers to use for the enterprise (e.g., OWASZP Enterprise Security API).
4.13 Practice 9: Don’t Trust service

Services can refer to any external system. Implicit trust of externally run system is not warranted. All external systems should be treated in a similar fashion.

For example, a loyalty program provider provides data that used by Internet banking, providing the number of reward points and a small list of potential redemption items. Within your program that obtains this data, you should check the results to ensure that it is safe to display to end users, and that reward points are a positive number and not improbably large.
4.14 Practice 10: Establish Secure Defaults

Secure by defaults means that the default configuration setting are the most secure setting possible friendly. Example, password aging and complexity should be enabled by default. Users may be allowed to turn these two features off to simplify their use of the application and increase their risk analysis and policies, but doesn’t force them into an insecure state by default.
4.15 Mapping Best Practice to Nonfunctional Requirements

The converges of Practices and NFRs is so dense, applying a security best practice will lead you to solving other non-security-related issues that your development teams and operations support personnel will most appreciate.
Chapter 5

Designing Applications for Security and Resilience
5.1 Design Phase Recommendations

Recommendations for the design phase are shown in Table 5.1

<table>
<thead>
<tr>
<th>SDLC phase</th>
<th>security Control (What to do)</th>
<th>Recommended Tools and processes (How to’s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design</td>
<td>Misuse case modeling</td>
<td>Requirements traceability matrix</td>
</tr>
<tr>
<td></td>
<td>Security design and architecture review</td>
<td>Security plan</td>
</tr>
<tr>
<td></td>
<td>Threat and risk modeling</td>
<td>Threat model</td>
</tr>
<tr>
<td></td>
<td>Security requirements and test case generation</td>
<td>Security test case template</td>
</tr>
</tbody>
</table>
5.1.1 Misuse Case Modeling

Misuse Case: A business process modeling tool used in the software development business. The term misuse case or mis-use case has derived from use case, meaning it is the inverse of a use case.

A key point here is that a use case model identifies regular uses of the application and is useful to model unorthodox or malicious uses of the application. For security architects and software testers, a use case model provides direct security value.
5.1.2 Security Design and Architecture Review

In most software development projects, time and budget are fixed values, and the introduction of security and resilience requirement is generally not well received by software development teams. The best place to introduce the security design and architecture review is when the team are engaged in the functional design and architecture review of the software.

Generating a security plan from the review is good start for documenting the security design and use it as a check-and-balance guide during and after development.
5.1.3 Threat and Risk Modeling

Threat modeling includes determining the attack surface of the software by examining its functionality for trust boundaries, entry points, data flow, and exit point.

Threat modeling is useful for ensuring that the design complements the security objectives, making trade-off and prioritization-of-effort decisions, and reducing the risk of security issues during development and operations.
Risk modeling of software can be accomplished by ranking the threats as they pertain to your organization’s business objectives, compliance and regulatory requirements, and security exposure.

Threat modeling is an iterative technique used to identify the threats to the software under construction. Threat modeling breaks the software into physical and logical construct, generating software artifacts that include data flow diagrams, end-to-end deployment scenarios, documented entry and exit points, protocols, components, identifies, and services.
Attack surface analysis, is a subset of threat modeling and can be performed when generating the software context to zero in on the parts of the software that are exposed to nontrusted users.

The following information should be included in threat-mode documentation:

◊ A diagram, and an enumeration and description of the elements in your diagram.
◊ A threat analysis, since that is core of the threat model.
◊ For each mitigated threat that you identify in threat analysis, the bug or defect number associated with the mitigate plan.
◊ A one- or two-paragraph description of your software components and what they do.
◊ Confirm that threat model data and associated documentation is stored using the document controls system used by the development team.
5.1.4 Risk Analysis and Modeling

During the design phase of development, you need to thoroughly review security and privacy requirements that stem from security concerns and privacy risks. Is referred to *risk analysis*.

Risk analysis considerations include the following:

- Threats and vulnerabilities that exist in the project’s environment or that result from interaction with other system
- Code that was created by external development groups in either source or project form.
Threat models should include all legacy code if the project is a new release of an existing program.

A detailed privacy analysis to document your project’s key privacy aspects. Important issues to consider include:

- What personal data is collected?
- What is the compelling customer value proposition and business justification?
- What notice and consent experiences are provided?
- How is unauthorized access to personal information prevented?
5.1.5 Security Requirements and Test Case Generation

Using a scenario–based security-testing template is effective in ensuring that the bare minimal security test cases are performed in every software development effort, as well as saving time in generating test cases that are essential.

Here are some rules of thumb to use while performing your threat modeling:

- If the data has not crossed a trust boundary, you do not really care about it.
- If the threat requires that the attacker is already running code on the client at your privilege level, you don’t care about it.
- If your code runs with any elevated privileges, you need to be concerned.
- If your code invalidates assumptions made by other entities, you need to be concerned.
If your code listens on the network, you need to be concerned.
If your code retrieves information from the internet, you need to be concerned.
If your code deals with data that came from a file, you need to be concerned.
If your code is marked as safe for scripting or safe for initialization, you need to be concerned.
5.2 Design to Meet Nonfunctional Requirements

Malicious mutating worms and cyber criminals are vital to engineer an application with enough security and resilience characteristics to provide the following:

- Assurance that users and client applications are identified and that their identities are properly verified
- Assurance that users and client applications can only access data and services for which they have been properly authorized
- Ability to detect attempt intrusions by unauthorized people and client applications
Assurance that unauthorized malicious programs do not infect the application or component
Assurance that communications and data are not intentionally corrupted
Assurance that parties to interactions with the application or component cannot later repudiate those interactions
Assurance that confidential communications and data are kept private
Assurance that applications can survive attack, or operate in a degraded mode
5.3 Design Patterns

The Secure Design Patterns report of the software Engineer Intrusion (SEI) describes a set of secure design patterns, which are descriptions or templates for a general solution to a security problem that can be applied in many different situations. Rather than focus on the implementation of specific security mechanisms, the secure design patterns detailed in the report are meant to eliminate the accidental insertion of vulnerabilities into code or to mitigate the consequences of vulnerabilities.
They are categorized according to their level of abstraction, as describes below:

- **Architectural-level patterns:** Focus on the high-level allocation of responsibilities among different components of the system and define the interactions among those high-level components. The architectural-level patterns are
  - Distrustful decomposition
  - Privilege separation
  - Defer to kernel
Design-level patterns: Describe how to design and implement elements of a high-level system component. There design-level patterns are:

- Secure factory
- Secure strategy factory
- Secure builder factory
- Secure chain of responsibility
- Secure state machine
- Secure visitor
Implementation-level patterns: patterns in this class are usually applicable to the implementation of specific functions or methods in the system. Implementation-level patterns include:

- Secure logger
- Clear sensitive information
- Secure directory
- Pathname canonicalization
- Input validation
- Resources acquisition is initialization
5.4 Architecting for the Web

A fundamental principle of Web application design is the separation of processing and control across multiple dedicated server infrastructures that cross a number of network domains.

Web applications are best suited for operating on infrastructure that consist of Web server(s), application server(s), and database server(s). The three-tire model is especially helpful for implementing security controls because it provides loose coupling that enables implementing security controls in exactly the places where they’ll do the most good while supporting the principle of defense in depth.
Three – or $n$-tire architectures also offer these other advantages:

- Centralization permits IT to control and secure programs and servers using an already-accepted, mainframe-life environment that is scalable, predictable, and easily monitored.
- Reliability is enhanced because equipment resides in a controlled environment that can be easily replicated or moved onto fault-tolerant systems.
- Scalability is easier because servers or processors can be added to achieve acceptable levels of performance.
Flexible, well-defined software layers permit the highest degrees of IT responsive to changing business needs.

Existing mainframe services can be reused through the virtue of a flexible data layer. Mainframe services can be made to look just like any other data services layer, thus preserving the transaction-processing capabilities of the mainframe.

Systems based on open industry standards allow companies to incorporate new technologies into the operation rapidly, without concern about interoperability problems that exist in products based on proprietary approaches.
5.5 Architecture and design Review Checklist

The Microsoft Developers Network (MSDN) covers most security and resilience aspects of the architecture and design stage of the SDLC, including the following areas:

- Authentication
- Authorization
- Configuration management
- Sensitive data
- Session management
- Cryptograph
- Parameter manipulation
- Exception management
- Auditing and logging
Chapter 11

Metrics and Model for security and Resilience Maturity

Summer Workshop on Cyber Security August 12-16, 2013 - Software Security, TTU
11.1 Maturity Models for Security and Resilience

Jeremy Epstein, a senior computer scientist at SRI International, wrote about the value of a software security maturity model:

So how do security maturity models like Open SAMM and BSIMM fit into this picture? Both have done a great job cataloging, updating, and organizing many of the "rules of thumb" that have been used over the past few decades for investing in software assurance.
The Open Software Assurance Maturity Model (SAMM) is an open framework developed by the Open Web Application Security Project (OWASP) to help organizations formulate and implement a strategy for software security that is tailored to the specific risks facing the organization.
The resources provided by OpenSAMM will aid in:

- Evaluating an organization's existing software security practices
- Building a balanced software security program in well-defined iterations
- Demonstrating concrete improvements to a security assurance program
- Defining and measuring security-related activities within an organization
OpenSAMM starts with the core activities that should be present in any organization that develops software:

- Governance: is centered on the processes and activities related to how an organization manages overall software development activities.
- Construction: concerns the processes and activities related to how an organization defines goals and creates software within development projects.
Verification: is focused on the processes and activities related to how an organization checks and tests artifacts produced throughout software development.

Deployment: entails the processes and activities related to how an organization manages release of software that has been created.
The three maturity levels for a practice correspond to:
(Implicit starting point with the Practice unfulfilled)
1. Initial understanding and ad hoc provision of the Practice
2. Increase efficiency and/or effectiveness of the Practice
3. Comprehensive mastery of the Practice at scale
11.2.1 Core Practice Areas

11.2.1.1 Governance Core Practice Areas:

- **Strategy & Metrics (SM)** involves the overall strategic direction of the software assurance program and instrumentation of processes and activities to collect metrics about an organization's security posture.

- **Policy & Compliance (PC)** involves setting up a security and compliance control and audit framework throughout an organization to achieve increased assurance in software under construction and in operation.

- **Education & Guidance (EG)** involves increasing security knowledge among personnel in software development through training and guidance on security topics relevant to individual job functions.
11.2.1.2 Construction Core Practice Areas

- **Threat Assessment (TA)** involves accurately identifying and characterizing potential attacks on an organization's software in order to better understand the risks and facilitate risk management.

- **Security Requirements (SR)** involve promoting the inclusion of security-related requirements during the software development process in order to specify correct functionality from inception.

- **Secure Architecture (SA)** involves bolstering the design process with activities to promote secure-by-default designs and control over technologies and frameworks on which software is built.
11.2.1.3 Verification Core Practice Areas

- **Design Review** (DR) involve inspection of the artifacts created from the design process to ensure provision of adequate security mechanisms and adherence to an organization's expectations for security.

- **Code Review** (CR) involves assessment of an organization's source code to aid vulnerability discovery and related mitigation activities as well as establish a baseline for secure coding expectations.

- **Security Testing** (ST) involves testing the organization's software in its runtime environment in order to both discover vulnerabilities and establish a minimum standard for software releases.
11.2.1.4 Deployment Core Practice Areas

• **Vulnerability Management** (VM) involves establishing consistent processes for managing internal and external vulnerability reports to limit exposure and gather data to enhance the security assurance program.

• **Environment Hardening** (EH) involves implementing controls for the operating environment surrounding an organization's software to bolster the security posture of applications that have been deployed.

• **Operational Enablement** (OE) involves identifying and capturing security-relevant information needed by an operator to properly configure, deploy, and run an organization's software.
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities
- Results:
- Success Metrics:
- Costs:
- Personnel:
- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
  The Objective is a general statement that captures the assurance goal of attaining the associated Level.

- **Activities**
- **Results**
- **Success Metrics**
- **Costs**
- **Personnel**
- **Related Levels**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
  The Objective is a general statement that captures the assurance goal of attaining the associated Level.

- **Activities**
- **Results**
- **Success Metrics**
- **Costs**
- **Personnel**
- **Related Levels**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results**
- **Success Metrics**
- **Costs**
- **Personnel**
- **Related Levels**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
  The Activities are core requisites for attaining the Level. Some are meant to be performed organization-wide and some correspond to actions for individual project teams.
- **Results:**
- **Success Metrics:**
- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
  The Activities are core requisites for attaining the Level. Some are meant to be performed organization-wide and some correspond to actions for individual project teams.
- **Results:**
- **Success Metrics:**
- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:
- Success Metrics:
- Costs:
- Personnel:
- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:

The Results characterize capabilities and deliverables obtained by achieving the given Level

- Success Metrics:
- Costs:
- Personnel:
- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
  The Results characterize capabilities and deliverables obtained by achieving the given Level
- **Success Metrics:**
- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
- **Success Metrics:**
- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
- **Success Metrics:**

The Success Metrics specify example measurements that can be used to check whether an organization is performing at the given Level. Data collection and management are left to the choice of each organization, but recommended data sources and thresholds are provided.

- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:
- Success Metrics:

The Success Metrics specify example measurements that can be used to check whether an organization is performing at the given Level. Data collection and management are left to the choice of each organization, but recommended data sources and thresholds are provided.

- Costs:
- Personnel:
- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
- **Success Metrics:**
- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:
- Success Metrics:
- Costs:

The Costs are qualitative statements about the expenses incurred by an organization attaining the given Level.

- Personnel:
- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
- **Success Metrics:**
- **Costs:**

The Costs are qualitative statements about the expenses incurred by an organization attaining the given Level.

- **Personnel:**
- **Related Levels:**
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
- **Success Metrics:**
- **Costs:**
- **Personnel:**
- **Related Levels:**
11.2.2  Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:

- Activities:

- Results:

- Success Metrics:

- Costs:

- Personnel:

These properties of a Level indicate the estimated ongoing overhead in terms of human resources for operating at the given Level.

- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- **Objective:**
- **Activities:**
- **Results:**
- **Success Metrics:**
- **Costs:**
- **Personnel:**

These properties of a Level indicate the estimated ongoing overhead in terms of human resources for operating at the given Level.
• Developers—Individuals performing detailed design and implementation of the software
• Architects—Individuals performing high-level design work and large-scale system engineering
• Managers—Individuals performing day-to-day management of development staff
• QA Testers—Individuals performing quality assurance testing and prerelease verification of software
• Security Auditors—Individuals with technical security knowledge related to software being produced
• Business Owners—Individuals performing key decision making on software and its business requirements
• Support Operations—Individuals performing customer support or direct technical operations support
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:
- Success Metrics:
- Costs:
- Personnel:
- Related Levels:
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:
- Success Metrics:
- Costs:
- Personnel:
- Related Levels:

The Related Levels are references to Levels within other Practices that have some potential overlaps depending on the organization's structure and progress in building an assurance program. Functionally, these indicate synergies or optimizations in Activity implementation if the Related Level is also a goal or already in place.
11.2.2 Levels of Maturity

Each core practice area is further detailed with a defined level of maturity using the following structure:

- Objective:
- Activities:
- Results:
- Success Metrics:
- Costs:
- Personnel:
- Related Levels:

The Related Levels are references to Levels within other Practices that have some potential overlaps depending on the organization's structure and progress in building an assurance program. Functionally, these indicate synergies or optimizations in Activity implementation if the Related Level is also a goal or already in place.
11.2.3 Assurance

Since the 12 Practices are each a maturity area, the successive objectives represent the "building blocks" for any assurance program.

OpenSAMM is designed for use in improving an assurance program in phases by:

- Selecting security Practices to improve in the next phase of the assurance program
- Achieving the next Objective in each Practice by performing the corresponding Activities at the specified Success Metrics
Using interval scorecards is encouraged for several situations:

- **Gap analysis**
  - Capturing scores from detailed assessments versus expected performance levels
- **Demonstrating improvement**
  - Capturing scores from before and after an iteration of assurance program build-out
- **Ongoing measurement**
  - Capturing scores over consistent time frames for an assurance program that is already in place
11.3 The Building Security In Maturity Model (BSIMM)

A tool to help people understand and plan a software security initiative based on the practices the BSIMM developers observed when developing the Software Security Framework.

The project's primary objective was to build a maturity model based on actual data gathered from nine large-scale software development initiatives.
A maturity model is appropriate because improving software security almost always means changing the way an organization works—something that doesn't happen overnight. The model is divided into 12 practices, falling under four categories:

- Governance
- Intelligence
- Software security development life cycle (SSDL) touchpoints
- Deployment
11.3.1 BSIMM Software Security Framework

The BSIMM Software Security Framework (SSF) is shown in Figure 11.5.

<table>
<thead>
<tr>
<th>The Software Security Framework (SSF)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Governance</td>
</tr>
<tr>
<td>Strategy and Metrics</td>
</tr>
<tr>
<td>Compliance and Policy</td>
</tr>
<tr>
<td>Training</td>
</tr>
<tr>
<td>Intelligence</td>
</tr>
<tr>
<td>Attack Models</td>
</tr>
<tr>
<td>Security Features and Design</td>
</tr>
<tr>
<td>Standards and Requirements</td>
</tr>
<tr>
<td>SSDL Touchpoints</td>
</tr>
<tr>
<td>Architecture Analysis</td>
</tr>
<tr>
<td>Code Review</td>
</tr>
<tr>
<td>Security Testing</td>
</tr>
<tr>
<td>Deployment</td>
</tr>
<tr>
<td>Penetration Testing</td>
</tr>
<tr>
<td>Software Environment</td>
</tr>
<tr>
<td>Configuration Management Standards</td>
</tr>
<tr>
<td>and Vulnerability Management</td>
</tr>
</tbody>
</table>

Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
Governance includes those practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.

In the governance domain:

- the strategy and metrics practice encompasses planning, assigning roles and responsibilities, identifying software security goals, determining budgets, and identifying metrics and gates.
The compliance and policy practice focuses on identifying controls for compliance regimens such as PCI and HIPAA, developing contractual control setting organizational software security policy, and auditing against that policy.

Training has always played a critical role in software security because software developers and architects often start with very little security knowledge.
11.3.1.2 Intelligence

*Intelligence* includes those practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization.

The intelligence domain:

- **Attack models** capture information used to think like an attacker: threat modeling, abuse-case development and refinement, data classification, and technology-specific attack patterns.
The security features and design practice are charged with creating usable security patterns for major security controls, building middleware frameworks for those controls, and creating and publishing other proactive security guidance.

The standards and requirements practice involves eliciting explicit security requirements from the organization, determining which COTS software to recommend, building standards for major security controls, creating security standards for technologies in use, and creating a standards review board.
11.3.1.3 SSDL Touchpoints

*SSDL touchpoints* include those practices associated with analysis and assurance of particular software development artifacts and processes.

The SSDL touchpoints domain:

- **Architecture analysis** encompasses capturing software architecture in concise diagrams, applying lists of risks and threats, adopting a process for review and building an assessment and remediation plan for the organization.

- **The code review** includes use of code review tools, the development of customized rules, profiles for tool use by different roles, manual analysis, and tracking/measuring results.
The security testing practice is concerned with prerelase testing including integrating security into standard quality assurance processes. Security testing focuses on vulnerabilities in construction.
11.3.1.4 Deployment

Deployment includes those practices that interface with traditional network security and software maintenance organizations.

In the deployment domain:

- **Penetration testing** focuses on vulnerabilities in final configuration and provides direct feeds to defect management and mitigation.

- **The software environment** practice concerns itself with operating system and platform patching, Web application firewalls, installation and configuration documentation, application monitoring, change management, and ultimately code signing.
The configuration management and vulnerability management practice is concerned with patching and updating applications, version control, defect tracking and remediation, and incident handling.
11.4 BSIMM Activities

11.4.1 Governance: Strategy and Metrics

11.4.1.1 Strategy and Metrics Level 1

I. Attain a common understanding of direction and strategy.

II. Managers must ensure that everyone associated with creating, deploying, operating, and maintaining software understands the written organizational software security objectives.

III. Leaders must also ensure that the organization as a whole understands the strategy for achieving these objectives.

IV. A common strategic is essential for effective and efficient program
Activates Strategy and Metrics

Level 1

1.1 *Publish process (roles, responsibilities, plan), evolve as necessary.*

The process for addressing software security is broadcast to all participates so that everyone knows the plan.

1.2 *Create evangelism role/internal marketing.* In order to build support for software security throughout the organization, the SSG plays an evangelism role. The SSG might give talks for internal groups, extend invitations to outside speakers, author white papers for internal consumption, or create a collection of papers, books, and other resources on an internal website.
Activates Strategy and Metrics

Level 1

1.3 *Educate executives.* Executives learn about the consequences of inadequate software security and the negative business impact that poor security can have. They also learn what other organizations are doing to attain software security.

1.4 *Identify gate locations, gather necessary artifacts.* The software security process will eventually involve release gates at one or more points in the software development life cycle (SDLC) or SDLCs. Importantly at this stage, the gates are not enforced.
1.5 *identify metrics and drive initiative budgets with them.* The SSG chooses the metrics it will use to define software security initiative progress. These metrics will drive the initiative's budget and allocation of resources. Metrics also allow the SSG to explain its goals in quantitative terms.
2.1 *Publish data about software security internally*. The SSG publishes data internally on the state of software security within the organization with the philosophy that sunlight is the best disinfectant. If the organization's culture promotes internal competition between groups, this information adds a security dimension to the game.

2.2 *Enforce gates with measures and track exceptions*. Gates are now enforced: In order to pass a gate, a project must either meet an established measure or obtain a waiver. Even recalcitrant project teams must now play along. The SSG tracks exceptions.
2.3 *Create or grow social network/satellite system.* The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill.

2.4 *Require security sign-off.* The organization has a process for risk acceptance and accountability prior to release.
3.1 *Use internal tracking application with portfolio view.* The SSG uses a tracking application to chart the progress of every piece of software in its purview. The application records the security activities scheduled, in progress, and completed. It holds results from activities such as architecture analysis, code review, and security testing.

3.2 *Run external marketing program.* The SSG markets itself outside the organization to build external support for the software security initiative.
11.4.2 Governance: Compliance and Policy

11.4.2.1 Activities Compliance and Policy Level 1

1.1 Know all regulatory pressures and unify approach. The SSG creates a unified approach that removes redundancy from overlapping compliance.

1.2 Identify personally identifiable information (PII) obligations. The SSG takes a lead role in identifying PII obligations stemming from regulation, customer demand, and consumer expectations. It uses this information to promote best practices related to privacy.
1.3 create policy. The policy provides a unified approach for satisfying the list of external security drivers. The SSG policy documents are sometimes focused around major compliance topics such as the handling of personally identifiable information or the use of cryptography.
11.4.2.1 Activities Compliance and Policy Level 2

2.1 Identify PII data in systems (inventory). The organization identifies the kinds of PII stored by each of its systems. When combined with the organization's PII obligations, this inventory guides privacy planning.

2.2 Require security sign-off for compliance-related risk. The organization has a formal process for risk acceptance. The risk acceptor signs off on the state of the software prior to release.
2.3 Implement/track controls for compliance. The organization can demonstrate compliance with applicable regulations because its practices are aligned with the control statements developed by the SSG. The SSG tracks the controls, shepherds problem areas, and makes sure auditors are satisfied.

2.4 Paper all vendor contracts with SLAB compatible with policy. Vendor contracts include a service-level agreement (SLA) ensuring that the vendor will not jeopardize the organization's compliance story. Each new or renewed contract contains a standard set of provisions requiring the vendor to deliver a product or service compatible with the organization's security policy.
2.5 Promote executive awareness of compliance/privacy obligations. The SSG gains executive buy-in around compliance and privacy activities. Executives understand the organization's compliance and privacy obligations and the potential consequences for failing to meet those obligations.
11.4.2.3 Activities Compliance and Policy Level 3

3.1 *Create regulator eye-candy.* The SSG has the information regulators want. A combination of policy, controls, and artifacts gathered through the SSDL give the SSG the ability to demonstrate the organization's compliance story without a fire drill for every audit.

3.2 *Impose Policy on Vendor.* Vendors are required to adhere to the same information used internally. Vendors must submit evidence that their software security practice pass muster.
3.3 Drive feedback from SSDL data back to policy. Information from the SSDL is routinely fed back into the policy creation process. Policies are improved to find defects earlier or prevent them from occurring in the first place.
11.4.3 Governance: Training

11.4.3.1 Activities Training Level 1

1.1 *Provide awareness training.* The SSG provides awareness training in order to promote a culture of security throughout the organization. Training might be delivered by members of the SSG, by an outside firm, by the internal training organization, or through a computer-based training system.

1.2 *Include security resources in on boarding.* The process for bringing new hires into the engineering organization includes a module on software security. The objective is to ensure that new hires enhance the security culture.
1.3 Establish SSG office hours. The SSG offers help to any and all comers during an advertised lab period or regularly scheduled office hours. By acting as an informal resource for people who want to solve security problems, the SSG leverages teachable moments and emphasizes the carrot over the stick.

1.4 Identify satellite through training: The satellite begins as a collection of people scattered across the organization who show an above-average level of security interest or skill.
11.4.3.2 Activities Training Level 2

2.1 Offer role-specific advanced curriculum (tools, technology stacks, bug parade). Software security training goes beyond building awareness and enables trainees to incorporate security practices into their work. The training is tailored to the role of trainees; trainees get information on the tools, technology stacks, or kinds of bugs that are most relevant to them.
2.2 *Create/use material specific to company history.* In order to make a strong and lasting change in behavior, training includes material specific to the company's history.

2.3 *Require annual refresher.* Everyone involved in making software is required to take an annual software security refresher course. The refresher keeps the staff up to date on security and ensures that the organization doesn't lose focus due to turnover.
2.4 **Offer on-demand individual training.** The organization lowers the burden on trainees and reduces the cost of delivering training by offering on-demand training for individuals.

2.5 **Hold satellite training/events.** The SSG strengthens its social network by holding special events for the satellite. The satellite learns about advanced topics or hears from guest speakers.
11.4.3.3 Activities Training Level 3

3.1 *Reward progression through curriculum (certification or HR).* Knowledge is its own reward, but progression through the security curriculum brings other benefits too. Developers and testers see a career advantage in learning about security.

3.2 *Provide training for vendors or outsource workers.* The organization offers security training for vendors and outsource providers. Spending time and effort helping suppliers get security right is easier than trying to figure out what they screwed up later on.
3.3 Host external software security events. The organization markets its security culture as a differentiator by hosting external security events. The organization as a whole benefits from putting its security credentials on display.
11.4.4 Intelligence: Attack Models

11.4.4.1 Activities Attack Models Level 1

1.1 *Build and maintain a top N possible attacks list.* The SSG helps the organization understand attack basics by maintaining a list of the most important attacks. This list combines input from multiple sources: observed attacks, hacker forums, industry trends, etc.

1.2 *Create data classification scheme and inventory.* The organization agrees on a data classification scheme and uses the scheme to inventory its software according to the kinds of data the software handles. This allows applications to be prioritized by their data classification.
1.3 Identify potential attackers. The SSG identifies potential attackers in order to understand their motivations and capabilities. The outcome of this exercise could be a set of attacker profiles including generic sketches for broad categories of attackers and more detailed descriptions for noteworthy individuals.

1.4 Collect and publish attack stories. In order to maximize the benefit from lessons that do not always come cheap, the SSG collects and publishes stories about attacks against the organization.
11.4.4.2 Activities Attack Models

Level 2

2.1 Build attack patterns and abuse cases tied to potential attackers. The SSG prepares for security testing and architecture analysis by building attack patterns and abuse cases tied to potential attackers.

2.2 Create technology-specific attack patterns. The SSG creates technology-specific attack patterns to capture knowledge about technology-driven attacks.

2.3 Gather attack intelligence. The SSG stays ahead of the curve by learning about new types of attacks and vulnerabilities. The information comes from attending conferences and workshops, monitoring attacker forums, and reading relevant publications, mailing lists, and blogs.
2.4 *Build internal forum to discuss attacks.* The organization has an internal forum where the SSG and the satellite can discuss attacks. The forum serves to communicate the attacker perspective.
11.4.4.3 Activities Attack Models

Level 3

3.1 Have a science team that develops new attack methods. The SSG has a science team that develops new attack methods. The team works to identify and "defang" new classes of attacks before real attackers even know they exist.

3.2 Create and use automation to do what the attackers will do. The SSG arms testers and auditors with automation to do what the attackers are going to do.
11.4.5 Intelligence: Security Features and Design

11.4.5.1 Activities Security Features and Design Level 1

1.1 Build/publish security features (authentication, role management, key management, audit/log, crypto, protocols). Some problems are best solved only once. Rather than have each project team implement all of its own security features, the SSG provides proactive guidance by building and publishing security for other groups to use.
1.2 *Engage SSG with architecture.* Security is a regular part of the organization's software architecture discussion. The architecture group takes responsibility for security the same way they take responsibility for performance, availability, or scalability.
11.4.5.2 Security Features and Design Level 2

2.1 *Build secure-by-design middleware frameworks/common libraries.* The SSG takes a proactive role in software design by building or providing pointers to secure-by-design middleware frameworks or common libraries.

2.2 *Create SSG capability to solve difficult design problems.* When the SSG is involved early in the new-product process, it contributes to new architecture and solves difficult design problems. The negative impact that security has on other constraints (time to market, price, etc.) is minimized.
2.3 Find/publish mature design patterns from the organization. The SSG fosters design reuse by finding and publishing mature design patterns from the organization.
11.4.5.3 Activities Security
Features and Design Level 3

3.1 Form review board or central committee to approve and maintain secure design. A review board or central committee approves and maintains secure design. The group formalizes the process for reaching consensus on design needs and security trade-offs.

3.2 Require use of approved security features and frameworks. Implementers must take their security features and frameworks from an approved list. There are two benefits: Developers do not spend time reinventing existing capabilities, and review teams do not have to contend with finding the same old defects in brand-new projects.
11.4.6 Standards and requirements

11.4.6.1 Standards and Requirements Level 1

1.1 Create security standards. Software security requires much more than security features, but security features are part of the job as well. A standard might describe how to perform authentication using J2EE or how to determine the authenticity of a software update.

1.2 Create security portal. The organization has a central location for information about software security. Typically this is an internal website maintained by the SSG.
1.3 *Translate compliance constraints to requirements.* Compliance constraints are translated into software requirements for individual projects.

1.4 *Create secure coding standards.* Secure coding standards help developers avoid the most obvious bugs and provide ground rules for code review. If the organization already has coding standards for other purposes, the secure coding standards should build on them.
2.1 Communicate standards to vendors. The SSG works with vendors to educate them and promote the organization's security standards. A healthy relationship with a vendor cannot be guaranteed through contract language.

2.2 Create a standards review board. The organization creates a standards review board to formalize the standards process and ensure that all stakeholders have a chance to weigh in. The board could operate by appointing a champion for any proposed standard.
2.3 Create standards for technology stacks. The organization uses standard technology stacks. A stack might include an operating system, a database, an application server, and a runtime environment for a managed language.

2.4 Identify open source in apps. The first step toward managing risk introduced by open source is to identify the open-source components in use. It is not uncommon to discover old versions of components with known vulnerabilities or multiple versions of the same component.
2.5 Create SLA boilerplate. The SSG works with the legal department to create standard SLA boilerplate for use in contracts with vendors and outsourcing providers. The legal department understands that the boilerplate helps prevent compliance or privacy problems.
3.1 Control open-source risk. The organization has control over its exposure to the vulnerabilities that come with using open-source components. Use of open source could be restricted to projects and versions that have been through an SSG screening process.
11.4.7.1 Activities Architecture Analysis Level 1

1.1 *Perform security feature review.* To get started with architecture analysis, center the analysis process on a review of security features.

1.2 *Perform design review for high-risk applications.* The organization learns about the benefits of architecture analysis by seeing real results for a few high-risk, high-profile applications.

1.3 *Have SSG lead review efforts.* The SSG takes a lead role in performing architecture analysis in order to begin building the organization's ability to uncover design laws. Architecture analysis is enough of an art that the SSG needs to be proficient at it before they can turn the job over to the architects, and proficiency requires practice.
1.4 *Use risk questionnaire to rank applications.* At the beginning of the AA process, the SSG uses a risk questionnaire to collect basic information about each application so that it can determine a risk classification and prioritization scheme.
2.1 Define/use AA process. The SSG defines a process for performing architecture analysis and applies it in the reviews it conducts. The process includes a standardized approach for thinking about attacks and security properties.

1.2 Standardize architectural descriptions (include data flow). The organization uses an agreed-on format for describing architecture, including a means for representing data flow.
2.3 *Make SSG available as AA resource/mentor.* In order to build an architecture analysis capability outside the SSG, the SSG advertises itself as a resource or mentor for teams who ask for help conducting their own analysis. The SSG will answer architecture analysis questions during office hours, and in some uses might assign someone to sit side by side with the architect for the duration of the analysis.
3.1 software architects lead review efforts. Software architects throughout the organization lead the architecture analysis process most of the time. The SSG might still contribute to architecture analysis in an advisory capacity or under special circumstances.

3.2 Drive analysis into standard architectural patterns. Failures identified during architecture analysis are fed back to the security design committee so the similar mistakes can be prevented in the future through improved design patterns.
11.4.8 SSDL Touchpoints: Code Review

11.4.8.1 Activities Code Review Level 1

1.1 *Create a top-N bugs list (real data preferred).* The SSG maintains a list of the most important kinds of bugs that need to be eliminated from the organization's code. The list helps focus the organization's attention on the bugs that matter most.

1.2 *Have SSG perform ad hoc review.* The SSG performs an ad hoc code review or high-risk applications in an opportunistic fashion. For example, the SSG might follow up the design review for high-risk applications with a code review.
1.3 Establish coding labs or office hours focused on review. SSG coding labs or office hours are sometimes used for code review. Software security improves without a rigid process.
11.4.8.2 Activities Code Review

Level 2

2.1 *Use automated tools along with manual review.* Incorporate static analysis into the code review process in order to make code review more efficient and more consistent.

2.2 *Enforce coding standards.* A violation of the organization’s coding standard is sufficient grounds for rejecting a piece of code. Code review is objective; it does not devolve into a debate about whether bad code is exploitable or not.
2.3 Make code review mandatory for all projects. Code review is a mandatory release gate for all projects. Lack of code review or unacceptable results will stop the release train.

2.4 Use centralized reporting (close the knowledge loop, drive training). The bugs found during code review are tracked in a centralized repository. This repository makes it possible to do summary reporting and trend reporting for the organization.
2.5 *Assign tool mentors.* Mentors are available to show developers how to get the most out of code review tools. If the SSG is most skilled with the tools, it could use office hours to help developers establish the right configuration or get started interpreting results.
11.4.8.3 activities Code Review

Level 3

3.1 *Use automated tools with tailored rules.* Customize static analysis to improve efficiency and reduce false positives. Use custom rules to find errors specific to the organization's coding standards or custom middleware.

3.2 *Build a factory.* Combine assessment techniques so that multiple analysis sources feed into one reporting and remediation process. The SSG might write scripts to invoke multiple detection techniques automatically and then combine the results into a format that can be used by a single downstream review and reporting solution.
3.3 Build capability for eradicating specific bugs from entire code base. When a new kind of bug is found, the SSG can write rules to find it, then go through the entire code base to identify all occurrences.
11.4.9.1 Activities Security Testing Level 1

1.1 *Ensure that QA supports edge/boundary value condition testing.* The QA team goes beyond functional testing to perform basic adversarial tests. They probe simple edge cases and boundary conditions. No attacker skills are required.

1.2 *Share security results with QA.* The SSG shares results from security reviews with the QA department. Over time, QA engineers learn the security mindset.
11.4.9.2 Activities Security Testing

Level 2

2.1 Integrate black box security tools into the QA process (including protocolfuzz0ing). The organization uses one or more black-box security testing tools as part of the quality assurance process. The tools are valuable because they encapsulate an attacker's perspective, albeit in a generic fashion.

2.2 Allow declarative security/security features to drive tests. Testers target declarative security mechanisms and security features in general.
2.3 begin to build/apply adversarial security tests (abuse cases). Testing begins to incorporate test cases based on abuse cases provided by the SSG. Testers move beyond verifying functionality and take on the attacker's perspective.
3.1 *Include security tests in QA automation.* Security tests run alongside functional tests as part of automated regression testing; the same automation framework houses both. Security testing is part of the routine.

3.2 *Perform fuzz testing customized to application APIs.* Test automation engineers customize a fuzzing framework to the organization's APIs. They may begin from scratch or use an existing fuzzing toolkit, but customization goes beyond creating custom protocol descriptions or file format templates.
3.3 *drive test with risk analysis results.* Testers use architecture analysis results to fat their work.

3.4 *Leverage coverage analysis.* Testers measure the code coverage of their security tests in order to identify code that is not being exercised. Code coverage drives increased security testing depth.
11.4.10 Deployment: Penetration Testing

11.4.10.1 Activities Penetration Testing Level 1

1.1 *Use external penetration testers to find problems.* Many organizations are not willing to address software security until there is unmistakable evidence that the organization is not somehow magically immune to the problem.

1.2 *Feed results to defect management and mitigation system.* Penetration testing results are fed back to development through established defect management or mitigation channels, and development responds using their defect management and release process.
11.4.10.2 Activities Penetration Testing Level 2

2.1 *Use pen testing tools internally.* The organization creates an internal penetration testing capability that makes use of tools. This capability can be part of the SSG, with the SSG occasionally performing a penetration test.

2.2 *Provide penetration testers with all available information.* Penetration testers, whether internal or external, are equipped with all available information about their target.

2.3 *Periodic scheduled pen tests for app coverage.* Test applications periodically according to an established schedule (which might be tied to the calendar or to the release cycle). The testing serves as a sanity check and helps ensure that yesterday's software isn't vulnerable to today's attacks.
11.4.10.3 Activities Penetration Testing Level 3

3.1 Use external penetration testers to perform deep dive (one-off bugs/fresh thinking). The organization uses external penetration testers to do deep-dive analysis for critical projects and to introduce fresh thinking into the SSG. These testers are experts and specialists.

3.2 Have SSG customize penetration testing (tools and scripts). The SSG either creates penetration testing tools or adapts publicly available tools so they can more efficiently and comprehensively attack the organization's systems.
11.4.11 Deployment: Software Environment

11.4.11.1 Activities Software Environment Level 1

1.1 *Use application input monitoring*. The organization monitors the input to software it runs in order to spot attacks. The SSG may be responsible for the care and feeding of the system. Responding to attack is not part of this activity.

1.2 *Ensure that host and network security basics are in place*. The organization provides a solid foundation for software by ensuring that host and network security basics are in place.
2.1 *Use code protection*. In order to protect intellectual property and make exploit development harder, the organization erects barriers to reverse engineering.

2.2 *Publish installation guides created by SSDL*. The software development life cycle requires the creation of an installation guide to help operators install and configure the software.
2.3 *Use application behavior monitoring and diagnostics.* The organization monitors the behavior of production software, looking for misbehavior and signs of attack.
11.4.11.3 Activities Software Environment Level 3

3.1 Use code signing: The organization uses code signing for software published across trust boundaries. Code signing is particularly useful for protecting the integrity of software that leaves the organization's control, such as shrink-wrapped applications or thick clients.
11.4.12 Deployment: Configuration Management and Vulnerability Management

1.4.12.1 Configuration Management and Vulnerability Management Level 1

1.1 *Create or interface with incident response.* The SSG is prepared to respond to in incident. The group either creates its own incident response capability or interfaces with the organization's existing incident response team.

1.2 *Identify software defects found in operations monitoring and feed them back to development.* Defects identified through operations monitoring are fed back to development and used to change developer behavior.
2.1 *Have emergency code base response.* The organization can make quick code changes when an application is under attack. A rapid-response team works conjunction with the application owners and the SSG to study the code d the attack find a resolution, and push a patch into production.

2.2 *Track software bugs found during ops through the fix process.* Defects found during operations are fed back to development and tracked through the fix Process.
2.3 Develop operations inventory of applications. The organization has a map of its software deployments.
11.4.12.3 Activities Configuration Management and Vulnerability Management Level 3

3.1 *Fix all occurrences of software bugs from ops.* In the code base. The organization fixes all instances of software bugs found during operations and not just the small number of instances that have triggered bug reports.

3.2 *Enhance dev processes (SSDL) to prevent cause of software bugs found in ops.* Experience from operations leads to changes in the development process (SSDL). The SSDL is strengthened to prevent a repeat of bugs found during operations.
11.5 Helpful Resources For Implementing BSIMM

To help you get started with BSIMM website for collecting information in MS Excel and developing a project implementation plan in MS Project. The spreadsheet will help you study, slice, and dice the activity info within the BSIMM, while the Project file will enable you to copy or click-click-drag the activities to arrange them in the phases or groupings you need.
11.6 Applying BSIMM to the Financial Services Domain

In 2009, the Financial Services Sector Coordinating Council (FSSCC) for Critical Infrastructure Protection and Homeland Security and the Financial and Banking Information Infrastructure Committee (FBIIC) Cyber Security Committee formed a working group called the Supply Chain Working Group. It was composed of leading security and risk-management practitioners who agreed to work together to create a deliverable that is useful to IT managers and information security officers interested in improving the resiliency of their organization's supply chains. The outcome of this work, entitled "Cyber Security Committee Supply Chain Working Group Toolkit,"
Supply Chain

- Suppliers
- Procurement
- Production
- Distribution
- Customers

Internet supply chain

Summer Workshop on Cyber Security August 12-16, 2013 – Software Security, TTU
The Toolkit is divided into four channels:

1. Internally developed software
2. Software developed by a third party
3. Software purchased off the shelf
4. Hardware, firmware, appliances
11.6.1 Working Group

Methodology

In each channel, the deliverables are divided into two sections:

1. A summary of survey results from four surveys (one per channel) of members of the Financial Services Information Sharing and Analysis Center (FS-ISAC) and members of the BITS/Financial Services Roundtable.

2. From the survey results, identification of leading practices to improve supply chain resilience was based on input from recognized subject-matter experts, including reference information for the growing body of information available on supply chain resilience.