University of Alberta

A Vulnerability Modeling Approach for Certifying Security in Components for E-commerce

by

Zhixiong Li ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirement for the degree of Master of Science

Department of Electrical and Computer Engineering

Edmonton, Alberta

Spring 2008

Your file Votre référence
ISBN: 978-0-494-45844-0
Our file Notre référence
ISBN: 978-0-494-45844-0

Abstract

Today most of the e-commerce applications are component-based and a security breach in any one of the components that comprise an e-commerce application may destroy the whole application. However, commerce components are commonly delivered in black box. End-users often suspiciously question the quality of these components. Thus demands for software certifications regarding the quality and security from third-party-independent agencies are becoming stronger and stronger. This, in turn, increases the demands for new certification technologies and methodologies.

This thesis proposes a product-based security certification process, Vulnerability Modeling Certification Process (VMCP). It works on design specifications and source code using white box technologies to identify software vulnerabilities and evaluate risk associated with these vulnerabilities. The security certification, which indicates the security level of the component, is then generated based on the identified and rated vulnerabilities. VMCP can be used as a basis for certifying components regarding security.

Acknowledgements

I am greatly indebted to my supervisor Prof. Dr. James Miller whose help, stimulating suggestions and encouragement helped me in all the time of research for and writing of this thesis.

Especially, I would like to give my special thanks to my wife Qi Wang whose patient love enabled me to complete this work.

Table of Contents

List of Abbreviations

ACL	access control list
API	application programming interface
CERT	computer emergency response team
CMMI	capability maturity model integration
COTS	commercial off-the-shelf
DAO	data access object
DFD	data flow diagrams
DoS	denial of service
DREAD	damage potential, reproducibility, exploitability, affected users and discoverability
EDFD	enhanced data flow diagram
GUI	graphical user interface
HTML	hypertext markup language
HTTP	hypertext transfer protocol
NASA	national aeronautics and space administration
RPC	remote procedure call
SCSP	software certification services provider
SQL	structured query language
SSL	secure sockets layer
STRIDE	spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege
URL	uniform resoure locator
VMCP	vulnerability modeling certification process
V&V	validation and verification
XML	extensible markup language
XSS	cross-site script

1 Introduction

Electronic commerce means doing business online, using the power of the Internet to gather and understand the needs and preferences of each customer and partner, to customize products and services for them, and then to deliver the products and services as quickly as possible. Personalized, automated services offer businesses potential benefits such as increasing revenues, lower costs, and good customer satisfaction, and good partner relationships. Electronic commerce is a realistic way to achieve these benefits, and thus many companies today are engaging in direct marketing, selling, and customer service; online banking and billing; secure distribution of information; value chain trading; and corporate purchasing.

Although the benefits of electronic commerce systems are enticing, there are still many aspects that keep companies from moving to electronic commerce. First, developing, implementing, and managing these systems is not always easy. Second, besides adopting new technology, many companies will need to reengineer their business processes to maximize the benefits of electronic commerce. Usually companies approach from traditional commerce to electronic commerce in three steps. At the beginning, companies slowly place some of their advertising onto the web, online versions of printed brochures. Gradually, they add other services, such as pricing and product information. This quickly leads to online ordering, procurement, and customer service systems. Now many of these so-called staid, old companies are fully web integrated. They have faced the Internet challenges, adapted to the changes, and replaced many of their business methods to facilitate web-based customer and vendor transactions.

Nowadays E-commerce is mainly used to automate many mundane, labor-intensive processes, including:

- Product research
- Request for quotes
- Automated customer inquiry
- Electronic order entry
- Outbound and inbound logistics
- Electronic payments
- Customer support and communications

Besides, the way business transactions are performed has also been changed by E-commerce. You can now shop online for insurance, loans, real estate, and even a local dentist. If you want to trade equities, then the web-enabled trading systems are there to execute the orders. If you want to buy a car, you can shop around online for the best price from different dealerships. Online banking is almost a must for any commercial bank. Customers pay bills, transfer money, and monitor their investments online. Tickets for sports and entertainment events are also sold on the web. You can even reserve your theater tickets online and see a view of the stage from the seat you are purchasing.

1.1 Why Security Is Important to E-commerce?

The Internet's very openness implies that all communication traveling over it is inherently difficult to secure. Companies are also suffering from the openness, the lack of security of web-based transactions, and the ease with which the privacy of online communications can be violated. To make matters worse, hacking is an epidemic that is on the rise. The security issues are not limited to e-commerce, but rather are part of much broader issues affecting computer and information systems throughout the world. Organizations are losing millions of dollars each year because of the security-related crimes ranging from virus attacks to business fraud, including the theft of sensitive business information and confidential credit card information.

The cost of damages related to security breaches is estimated as billions of dollars. However, the situation is becoming worse and worse because of the fact that with the ever increasing number of users of information systems, easy access to information, and the increasing number of knowledgeable users, one can easily assume that the number of technology misuses and security threats will increase proportionally. Ira Winkler, president of the Internet Security Advisors Group in Severna Park, Md., and author of "Corporate Espionage" (Prima Publishing, 1999) succinctly states the average e-commerce business's security dilemma: "To a hacker, you're just an IP address. You get hit because you let yourself be an easy mark." Here are some eye-opening figures to contemplate: A study by Gartner Inc. indicates that 50 percent of all small to midsize enterprises were hacked in 2005, with almost 60 percent of those not even knowing they had been hacked. According to the Computer Emergency Response Team (better known as "CERT," www.cert.org), a total of 137,529 incidents were reported in 2005. But incidents are rapidly increasing — there were 173,521 reported incidents in 2006.

What makes things worse is that in their haste to get on the web, many organizations do not take security into account. As a result, the required controls often were not put in place. Moreover application developers who build the web-based applications are not experienced at web security. When they design and implement applications, developers focus on functionality. Thus hackers can attack an organization at will. New exploits enable them to shut down or seriously disrupt business processes.

Regrettably, as many businesses are reluctant to reveal that their systems have been infiltrated, and to share knowledge about their incident and the extent of the damage, the true extent of damages incurred by businesses related to e-commerce security crime cannot truly be known. The reason why businesses are reluctant to share information regarding their security breaches is because they fear that once the public learns about their incident, their customers will lose confidence in the business's ability to protect its assets, and as a result the business will lose customers in turn losing their profitability. Nobody is willing to provide financial information to an online shopping website after the company has reported that they were attacked by hackers. Businesses have nothing to gain by voluntarily admitting to having been victimized by security-related crimes. To maintain the business's survivability and competitive stance companies have to maintain a positive image regarding e-commerce security in today's media madness regarding the Internet.

1.2 Security – What Is A Secure System?

Security is fundamentally about protecting assets from illegal access. Assets may be tangible items, such as a file on your hard driver or the application database, or they may be intangible, such as your company's reputation or the availability of your web site. Security is a path rather than a destination. It is all about risk assessment and management, and implementing effective countermeasures. As you analyze and design your applications, you identify potential threats and assess the corresponding risks of each threat. Then you decide the most effective and economical approaches to mitigate these threats, making tradeoffs between the cost and benefit. A secure system should have powerful mitigation strategies and implementations against both external and internal threats of the following aspects:

- **Authentication**

  Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services by validating the user to whom they claims to be, typically through credentials, such as a user name and password. These clients may be end users, other services, processes, or servers.

- **Authorization**

  Authorization addresses the question: what can you do? It is all about granting or denying access to the resources and operations for which access is requested by the authenticated client. Authorization is usually accomplished based on user identity and role membership. Resources include files, databases, tables, rows, and so on, together with system-level resources such as registry keys and memory data. Operations include performing transactions such as purchasing a product, depositing and withdrawing money from one account, or updating a user profile.

- **Auditing**

  Auditing addresses the question: who has done what? It is all about logging events that have happened, which is the key to non-repudiation. Non-repudiation guarantees that a user has to admit what they have done, such as performing an operation or initiating a transaction. For example, in an online banking system, non-repudiation mechanisms are required to make sure that a consumer cannot deny transferring $1000 from one account to another account.

- **Confidentiality**

  Confidentiality, also referred to as *privacy*, is all about keeping secrets. It is the process of keeping private confidential data from unauthorized users or eavesdroppers who monitor the flow of traffic across a network. Two common means of enforcing confidentiality are encryption and access control lists (ACLs).

- **Integrity**

  Integrity is all about protecting data from accidental or deliberate (malicious) modification. For data passed across networks integrity is a key concern as serious as privacy (confidentiality). Hashing techniques and message authentication codes are typically used to protect the integrity of data in transit.

- **Availability**

  Availability is all about keeping systems operational for legitimate users. Denial of service attacks are the most common threats to applications regarding availability.

1.3 Modern Approaches in Constructing Web and E-commerce Applications – COTS based

The world software and services market has grown into a huge industry, about 500 billion US dollar per year. It is proven that one of the most efficient ways to reduce the cost of software design and development is to use software components, either COTS or those built in house. Component-based Internet technologies such as J2EE and .NET are making the use of software components easier and more pervasive than ever before. Nowadays, the Internet is being harnessed by mainstream businesses of all sizes for group collaboration, communication, and inexpensive dissemination of information. Component-based technologies such as Java applets, JavaBeans, and ActiveX controls make it possible for businesses to build faster and cheaper Web-based applications. The next step in the evolution of business on the Internet is electronic commerce.

Today both government and commercial organizations are already being prepared for systems that employ COTS functionality. For example, guidelines or standards have already been put forth by the US Federal Aviation Administration, US Department of Defense, and the US Food and Drug Agency [22].

1.4 The Need for a Security Assessment of COTS Component

Today when people talk about e-commerce security most of them refer to encryption technology and protocols for securing the data transaction. However it is easy to understand that a weakness in any one of the components that comprise an e-commerce system may lead to a security breach. For example, a flaw in the server side validation may allow a criminal access to the data stored in the database without forcing the criminal to break any cipher text at all. Similarly, vulnerabilities in security models for online banking may allow insecure behavior to originate from client-side interaction. Therefore the first thing for e-commerce to obtain mass market acceptance is to adequately address the security issues of component based commerce systems.

Today we are still in the early stages of building secure components and there are many aspects responsible for the poor security in these components. The most important reason is that building secure components is a hard, a complex task with no well-established processes to follow. Design and implementing security-critical components require development teams to have strong skills within at least three areas: cryptography, computer security, and software engineering. Complexity comes from the intersection and coupling of these three area and the conflicting interests. An example is usability versus authentication mechanisms. Also, in developing web-based applications, the choice of communication models, multi-thread programming, and session management add to the complexity. Recently, a number of Web-application vulnerabilities have been discovered and all of them are related to the same problem: improper *input validation* [2].

Another reason contributing to the poor security is the shortage of people with security background and training. Because of the shortage a large number of unqualified technicians are developing secure software systems. The situation in today's software industry much likes that of the British banking industry in the 80's and early 90's, where implementations by untrained personnel were the cause for the majority of security breaches. A central issue of secure software is the definition of security. Infrastructure security people often associate security with systems such as firewalls and intrusion detection systems while developers of web applications tend to think of the Secure Sockets Layer (SSL) protocol when talking about web security. However, security is an emergent system property, not only a feature, but the sum of a set of non-functional goals, which include procedures for prevention, traceability, auditing, monitoring, privacy, confidentiality, anonymity, authentication, and integrity. Experts who are able to perform an extensive security evaluation are a rare breed.

Due to the complexities involved in designing and implementing secure systems, most of application developers and end users will have to rely on *trusted* software provided by third parties as they cannot fully understand a security-critical program's inner workings. On the other side, besides building trustworthy systems, the providers of secure components face another challenge to find ways to gain trust from consumers. Software components are often delivered as executable objects ("black-boxes") with licensing agreements that forbid decompilation back to source. It is easy to think that a more expensive component is more reliable and it might have received more testing. But it is not always the fact. A less expensive component that has experienced more usage may actually have higher quality. Component providers need a way to show the real quality of their components. Objective and scientific security assessment is essential to this step.

1.5 Software Certification

Today, commerce components are commonly delivered in black box (executable format). End-users often suspiciously question their quality, thinking they do not meet their security requirements, do not match the functionality described in the manual, and are not tested thoroughly. The component provider cannot do much to change this situation, without some form of independent third-party validation. There are several reasons accounting for this: the liability associated with making claims concerning quality, end-user distrust of providers, software publishers are likely to not test components to the levels that would justify software warranties because of the cost and rush to put the components on the market, and publishers are likely to make incorrect assumptions about how users will use the software. Therefore, in the software world a new demand or business is being raised by both the component producers and consumers that independent agencies certify that programs meet certain criteria. Vendors want proof of the quality of their products and consumers want unbiased assessments from a third party. By hiring neutral agencies to do assessments on components and grant certificates, publishers get official testimony of their product's quality while end-users will also benefit from these unbiased assessments.

In recent years the business case for creating independent agencies to certify software quality has become stronger and stronger. Agencies that perform third-party-independent software certification are referred to as Software Certification Services Providers (SCSPs). In addition to all the advantages mentioned above, another advantage of having independent SCSPs is that they provide a "fair environment" for each publisher, assuming that each product of a certain field receives equal treatment. Though the demands for SCSPs are strong there are not many SCSP today. A key reason of this situation is the liability of being a certifier. When certified software fails in the field in a manner that was claimed to not be possible, the certifier bears some level of liability. This requires that the certifiers must be very confident about their assessment results before issuing a certificate. Although SCSPs have not become widespread there are several relatively unkonwn SCSPs in existence today. KeyLabs is such an organization; it handles applications for 100% Pure Java. Other than these small specialized labs, the next closest organization is Underwriters Laboratory (UL). UL certificates electrical product designs to ensure that safety concerns are mitigated.

To certify software, the first and also one of the most important steps for the

SCSPs is to setup the criteria against which the software is validated and assessed. Both subjective and objective software criteria can be tested for by SCSPs, spanning the spectrum from guaranteeing functionality to counting the lines of code. Subjective criteria tend to be imprecise and prone to error while objective criteria are precise and less prone to error. For example, deciding whether a component's behavior is correct or not is subjective as what "correctness" really means for a piece of software depends on how the certifier defines the rules against which the component is tested. SCSPs should be more careful when rendering professional opinions for criteria that are as contentious as this. But it is easy for SCSPs to assess characteristics such as whether a program has input data validation in it and how many lines of code a program has. Testing for these criteria is not very hard. However when an SCSP tries to estimate a criterion such as software security troubles will begin as how hackers attack a component and how the component in question defenses itself is quite subjective and greatly based on the testers' experience, if there is no well-defined, relatively-experience-free process to follow.

Today, the number of approaches and standards for certifying software quality is increasing. The most popular approaches are process-based (e.g., ISO9000 and SEI-CMMI). These approaches either validate the integrity of the software product development processes or personnel. NASA is one of the most successful organizations in adopting process-based certification approaches, though it is not a commercial SCSP. NASA requests independent certification both for the software they write as well as the software they purchase and built its own SCSP -- the Independent Verification & Validation facility in Fairmont, WV. Intermetrics is hired as the prime contractor by the facility to oversee the certification process and provide the necessary independent assessment. Intermetrics is a real commercial SCSP and provides NASA with a common software assessment process over all software projects.

With the development of software industry, particularly the widespread use of COTS, process-based certification approaches appear more and more inefficient and limited. Today components of one software product may come from several publishers. How can we know the quality of this software? If only a subset of these publishers are process-based certified. In addition, process-based certification is not for a certain software product but for the ability of a company to produce a high quality product. For example, CMMI is a certification which focuses on improving software quality by improving the software development process. It is a certification for the company but not for the individual products. Additionally, CMMI is extremely time- and cost- consuming, as it impacts not only the software development process but also the culture and organization of companies. So far, there are still very few companies that have passed the highest level of CMMI, Level 5.

Thus new approaches of independent "product-based" software certification are getting more and more interesting. "Product-based" approaches certify the quality of software products, based on the certain criteria and information provided by publishers. This is normally accomplished via verification and validation approaches. This kind of certificate is very much helpful when a customer determines whether a candidate component is dependable enough for their needs. It also enables customers to decide which component to purchase amongst several competing options given components' dependability scores and licensing costs (if alternatives exist). These "product-based" certificates also provide small and medium publishers with an opportunity to increase consumer confidence. The National Security Computer Association (NCSA) is a for-profit SCSP that has taken product-based approaches in firewall certification program. They use only objective criteria that specific known problems are not present in an applicant's system.

There is a group of industry representatives who meet periodically to decide what known problems should be checked for. NCSA uses their opinions as the base of their certification program and gradually introduces additional criteria into the certification process. This adaptive certification process adds rigor to the firewall certification process and thus produces a steady stream of business for the NCSA. To further reduce liability. NCSA makes a disclaimer that their firewall certificate does not cover firewall security.

Currently the biggest issue for "product-based" certificate is still the liability issue. When a software component fails in the field that an independent party has provided an assessment for, what responsibility should the independent party take for the failure and/or the loss caused? This is the main reason why "process-based" certificates are overwhelming "product-based" certificates in business.

In term of technology "product-based" procedures are much difficulty. A single "product-based" certification methodology is probably not going to "fit all" types of software but a single process-based methodology is good for most companies. Numerous certification methodologies will be needed and the basic requirement is that these methodologies are able to provide assessment in two areas: (1) what the component does, and (2) what level of integrity is guaranteed by the certificate's "seal of approval."

Despite of all these difficulties, it is believed that the stage has been set for independent "product-based" software certificate. With the emergence of new "product-based" technologies, and the maturity of these technologies, "product-based" certification will become more and more popular. In this thesis, I will propose a new technology to certify the security of components for e-commerce.

1.6 Related Work

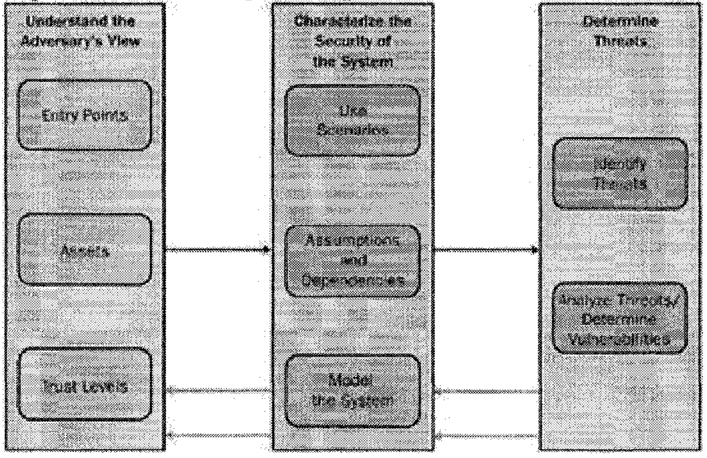
Microsoft's Threat Modeling Process

Threat modeling is a fairly new technique used to develop more secure applications. It is, in essence, the act of creating a security design specification and later testing that design. By assessing and documenting the security risks associated with an application, the threat modeling methodology first creates a threat profile of the application being developed, which is an enumeration of the entire adversary's goals for the system. Then it analyzes every threat to find out the best way to mitigate the threat.

Although threat modeling is mainly used in application development, it provides us with new concepts on how to build a product-based certification process. That is creating an attack profile of a given component, which contains all potential attacks to the component, and then verifying how the component defends these attacks; in other words, how the threats are mitigated. This is the basic concept of this thesis.

So far the most mature version of a threat modeling process is presented by Swiderski and Snyder [5]. process is comprised of three high-level steps: understanding the adversary's view, characterizing the security of the system, and determining threats. Each of these steps has logical sub-steps. Figure 1-6-1 illustrates the process.

Figure 1-6-1: The high-level process of threat modeling.


Understand the Adversary's View
Entry Points
Assets
Trust Levels
Characterize the Security of the System
Use Scenarios
Assumptions and Dependencies
Model the System
Determine Threats
Identify Threats
Analyze Threats/ Determine Vulnerabilities

The following briefly explain each of the steps.

Understanding the Adversary's View
Threat modeling takes an outside-in approach to understand the adversary's view of the system. That is enumerating entry points and assets, as well as cross-referencing them with trust levels.

- Entry Points

Entry points are any location where data or control transfers between the system being modeled and another system. They show all the places where the adversary can attack the system, including transfer points such as open sockets, remote procedure call (RPC) interfaces, Web services interfaces, and data being read from the file system.

- Assets

Assets are the resources the component or system has that an adversary might try to modify, steal, or otherwise access or manipulate. Assets can be tangible, such as a process token, or more abstract, such as data consistency (for example, a string class that maintains a length field).

- Trust Levels

Trust levels define how external entities are characterized for the system. They define the privilege that an external entity should have to legitimately use an entry point or functionality at the entry point, and they dictate which assets external entities should normally be allowed to access or affect in some way.

Characterizing the Security of the System

Characterizing the security of the system involves bounding the threat model, gathering information about dependencies that are critical to security, and understanding the internal workings of the system.

- Define usage scenarios.

Development teams must ask themselves how the component or system will be used. Conversely, the teams can ask themselves how the component or system will not be used.

- Identify assumptions and dependencies.

Development teams should collect information such as external dependencies, external and internal security notes, and implementation assumptions.

- Model the system.

Data flow diagrams (DFDs) or other diagrams, such as process models, are used to understand the actions a system performs at a given entry point. DFDs are visual representations of how a system processes data.

Determining Threats

Enumerating threats creates a threat profile for a system, describing all the potential attacks that architects and developers should mitigate against. The security of a system can be expressed in terms of threats with appropriate mitigation vs. total threats, taking into account the severity of the threats with insufficient mitigation.

- Identify threats.

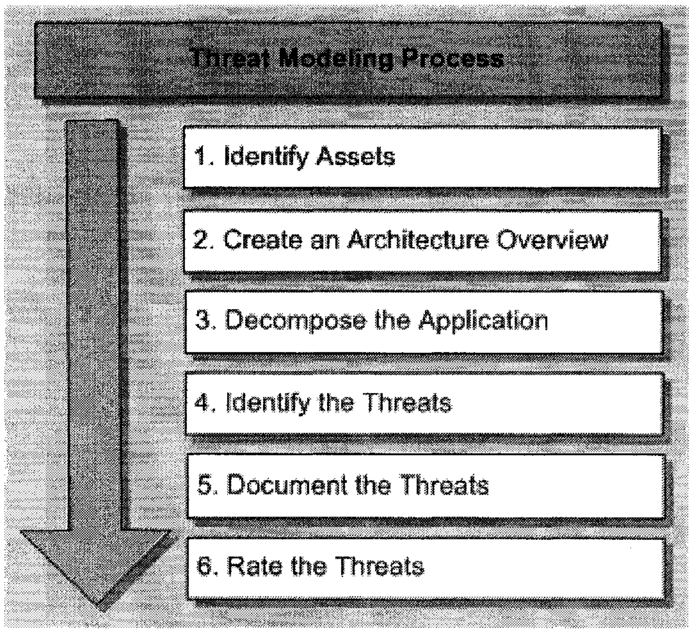
For each entry point, the development team determines how an adversary might try to affect an asset. Based on what the asset is, the team predicts what the adversary would try to do and what his goals would be.

- Analyze threats.

Development teams model threats to determine whether they are mitigated. Using threat trees, a development team can decompose a threat into individual, testable conditions. Threats that are not mitigated become vulnerabilities—security bugs that must be remedied.

There are other 2 articles published at Microsoft MSDN website addressing threat modeling. Though processes proposed by these articles are much less mature and complete than the one presented above, they are still worth being read. The first one is [20]. Figure 1-6-2 represents the process proposed.

Figure 1-6-2. An overview of the threat modeling process



The six threat modeling steps are:

Step 1: Identify assets.
Identify the assets that you need to protect. This could range from confidential data, such as your customer or orders database, to your Web pages or Web site availability.

Step 2: Create an architecture overview.
Use simple diagrams and tables to document the architecture of the component, including subsystems, trust boundaries, and data flow.

Step 3: Decompose the application.

Decompose the architecture of the component, including the underlying network and host infrastructure design, to create a security profile for the application. The aim of the security profile is to uncover vulnerabilities in the design, implementation, or deployment configuration of the component.

Step 4: Identify the threats.

Keeping the goals of an attacker in mind, and with knowledge of the architecture and potential vulnerabilities of the component, identify the threats that could affect the application.

Step 5: Document the threats.

For each threat, document each threat using a common threat template that defines a core set of attributes to capture.

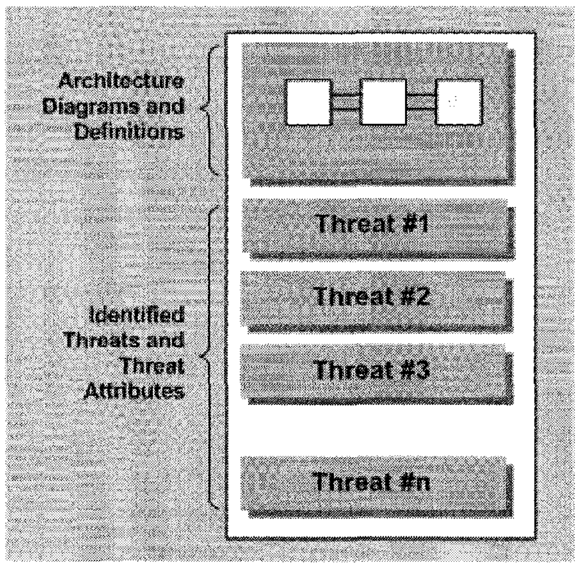
Step 6: Rate the threats.

Rate the threats to prioritize and address the most significant threats first. These threats present the biggest risk. The rating process weighs the probability of the threat against the damage that could result should an attack occur.

Output

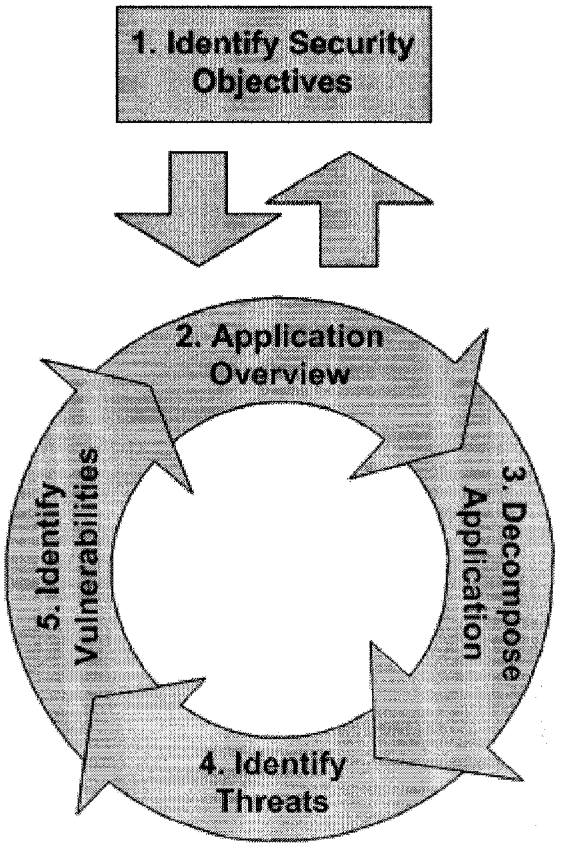
The output from the threat modeling process is a document for the various members of your project team. It allows them to clearly understand the threats that need to be addressed and how to address them. Threat models consist of a definition of the architecture of the component and a list of threats for the component scenario, as Figure 1-6-3 shows.

Figure 1-6-3. Components of the threat model


Architecture Diagrams and Definitions
Identified Threats and Threat Attributes
Threat #1
Threat #2
Threat #3
Threat #n

The other article is [6]. Figure 1-6-4 represents the process proposed.

Figure 1-6-4. The iterative threat modeling process



The five threat modeling steps are:

- Step 1: Identify security objectives. Clear objectives help you to focus the threat modeling activity and determine how much effort to spend on subsequent steps.
- Step 2: Create an application overview. Itemizing your application's important characteristics and actors helps you to identify relevant threats during step 4.
- Step 3: Decompose your application. A detailed understanding of the mechanics of your application makes it easier for you to uncover more relevant and more detailed threats.
- Step 4: Identify threats. Use details from steps 2 and 3 to identify threats relevant to your application scenario and context.
- Step 5: Identify vulnerabilities. Review the layers of your application to identify weaknesses related to your threats. Use vulnerability categories to help you focus on those areas where mistakes are most often made.

Trike Methodology

Trike [21] is a unified conceptual framework for security auditing from a risk management perspective through the generation of threat models in a reliable, repeatable manner. It is supposed to be used by a security auditing team to completely and accurately describe the security characteristics of a system from its high-level architecture to its low-level implementation details. In general, Trike uses the following four models to identify and assess threats in a system.

Requirements Model
Trike believes all threat models must begin with an understanding of what the system is intended to do. It uses an Actor-Asset-Action matrix to represent the requirement of a system. Trike looks at who interacts with the system (actor), what things the system acts upon (asset), and the actions taken by actors that the system is intended to support. Also Trike looks at what rules exist in the system to constrain those actions, and ties all of this information up in a convenient tabular format.

Implementation Model
Trike starts the implementation model by looking at those actions in the system which do not fit into the intended actions framework and how actions interact with the state of the system. It then looks at how the different software and hardware components of the system fit together in the data flow diagram. Finally Trike maps from the actions and state of the system into the data flow diagram.

Threat Model
Trike identifies threats from the full model for both the requirements of the application and the implementation of the application. Then it proceeds by building the attack graph and examining the actual system to verify all weaknesses in the system. This done, Trike can determine the vulnerabilities to the system and apply mitigations.

Risk Model
Trike calculates a threat risk value by multiplying the exposure for the threat by the probabilities associated with the vulnerabilities that implement that threat. This provides users with a set of values, which take into account the technical security issues and relate them to the business impact of those issues.

Although by the time of this thesis being written, Trike was under heavy developed and only published a draft version, it is still a valuable paper on threat modeling and security assessment.

1.7 Objectives of This Thesis

This thesis is supposed to propose a new product-based certification process to certify the security in components used by e-commerce applications. The new process will work on design specifications and source code using white box technologies to identify software vulnerabilities and evaluate risk associated with these vulnerabilities. The security certification, which indicates the security level of the component, is then generated based on the identified and rated vulnerabilities.

2 Process Overview and Framework

Process Overview

The Vulnerability Modeling Certification Process (VMCP) is a product-based security certification process. It assesses and documents the security vulnerabilities associated with a component. VMCP can help SCSPs (Software Certification Services Providers) identify vulnerabilities and assess the risk of these identified vulnerabilities and thus it serves as a basis for certifying components regarding security.

VMCP is based on white box technologies. Its key concept is very simple and straightforward: identifying architecture and implementation vulnerabilities by reviewing design documentations and performing code review. First VMCP tries to figure out the potential attacks based on the component's architecture and design and then VMCP tries to see how the component defends these attacks. Vulnerabilities are expected to be exposed during the procedures. VMCP consists of a series of steps that are structured and practiced enough to be used in real life projects. The rest of this thesis will describe each step in details.

VMCP involves understanding an adversary's goals in attacking a system based on the system's assets of interest. It looks at a component from an adversary's perspective to anticipate attack goals. VMCP is based on two premises:

- **An adversary will not attack the system without assets of interest.** This first principle is very easy to understand. An adversary's goals are always based on the system's assets. The adversary has no reason to attack the system unless it contains something of value to them—for example, corporate or personal data, processing resources, or financial information. Assets are resources that the component or system possesses that an adversary might try to modify, steal, or manipulate. Assets can be tangible, such as data stored in the database or more abstract (intangible), such as the availability of services. Assets are the basis for attacks. It is impossible to have an attack without a corresponding asset because assets are essentially attack targets.
- **An adversary cannot attack a system without entry points—interfaces the system has with the outside world.** Entry points are any location where data, or control, transfers between the system being modeled and another system. In most cases, the adversary must actively jeopardize the application's security via entry points. Entry points show all the places where the adversary can attack the system from, including transfer points such as open sockets, remote procedure call (RPC) interfaces, Web services interfaces, and data being read from the file system. Entry points are not only the places where data or commands flow into the system but include points where data or information flows out of

the system. For example, some wireless applications broadcast information that is unsolicited. A passive attacker who listens for this information could discover valuable information about the application.

VMCP bases on a system's entry points (in other words, interfaces the system has with the outside world) to determine the functionality that an adversary can exercise on the system and what assets he can affect. This allows SCSPs to enumerate potential attacks and vulnerabilities are discovered when attacks are analyzed. VMCP involves the following main steps:

- **Develop and understand a common taxonomy of security attacks and vulnerabilities.** VMCP certifies components by assessing how components defend known attacks. A common taxonomy of security attacks together with vulnerabilities that make these attacks possible should be developed and deeply understood. This taxonomy covers the most common application level attacks and vulnerabilities that plague web-based enterprise applications.
- **Understand components from an adversary's view.** Check the component in question to see

    1. How it accepts data from outside providers and how it interacts with the outside environment;
    2. What might be the interesting assets to attackers;
    3. Where are the weakest points that attackers might conduct their attack through; and
    4. Where are the attack paths through which attackers are able to reach assets of interest and how attackers break the security rules on these paths?

- **Identify potential attacks.** Having a good understanding from an adversary's view and a common taxonomy of security attacks in mind, assessors are able to identify potential attacks on the component. Attacks are identified along the attack paths by thinking about how attackers reach the assets that are connected to these attack paths.
- **Discover and rate vulnerabilities.** Using techniques such as threat (attack) trees to analyze attack vectors, VMCP can find architectural vulnerabilities as well as direct code reviews to find implementation vulnerabilities. Rating is done by evaluating potential impact of each of the vulnerabilities on the system using the component. VMCP considers five aspects when analyzing impact: Damage potential, Reproducibility, Exploitability, Affected users and Discoverability (DREAD) [67].
- **Certify components.** Draw a conclusion on the component in question regarding the security according to the rated vulnerabilities and risks caused.

Process Framework

VMCP is a structured activity for identifying and evaluating attacks and vulnerabilities that are most likely to affect the component's security. First from the perspective of an adversity, VMCP tries to figure out how the component can be attacked and then disclose the vulnerabilities through verifying how the component defends the attack. Figure 2-1 shows the framework of VMCP. Basically, there are 3 phases and 8 steps in the VMCP process.

Phase I: Preparation Phase. In this phase, we do some pre-work before modeling vulnerabilities of a certain component.

- **Step 1: Develop a common taxonomy of security attacks and vulnerabilities**. This taxonomy is used as the start point in later steps identifying attacks and vulnerabilities. It covers the most common application level attacks and vulnerabilities that plague web-based enterprise applications. For each category within the taxonomy, VMCP first identifies the attacks and then the related vulnerabilities that may pose the risks by using some effect-cause analysis techniques.

Phase II: Modeling Phase. This phase is iterative. By iterating the modeling phase, we can gradually refine our modeling result when we become more and more familiar with the component.

- **Step 2: Gain an architecture overview from the security perspective.** In this step, we try to understand the component from the perspective of security. We focus these materials that will help us to identify relevant risks during latter steps, such as the end-to-end deployment scenario, roles, key usage scenario, technologies used and application security mechanism. Usually we derive them from the specifications coming with the component.

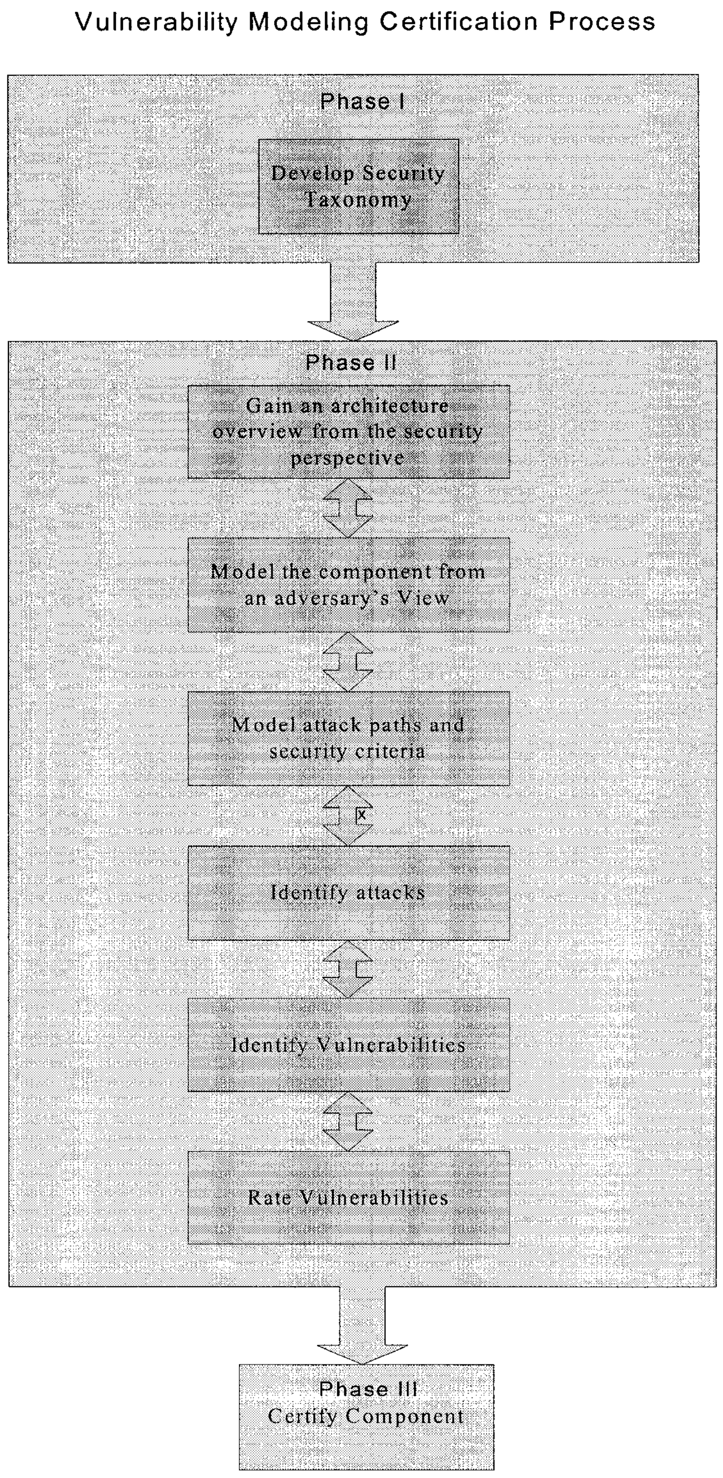
- **Step 3: Model the component from an adversary's View.** VMCP looks at a component from an adversary's perspective to anticipate attack goals. VMCP is based on the following two premises:

  1. An adversary will not attack the system without assets of interest.

  2. An adversary cannot attack a system without entry points, interfaces the system has with the outside world

  Here we break down the component in question to identify all assets of interest and entry points, together with trust boundaries where trust levels change.

Figure 2-1: VMCP Framework



Vulnerability Modeling Certification Process
Phase I
Develop Security Taxonomy
Phase II
Gain an architecture overview from the security perspective
Model the component from an adversary's View
Model attack paths and security criteria
Identify attacks
Identify Vulnerabilities
Rate Vulnerabilities
Phase III
Certify Component

- **Step 4: Model attack paths and security criteria.** An attack path in VMCP is a business logic path within the component that shows how information or control goes through an entry points and reaches the assets.

Attackers can only perform attacks along these attack paths, as they cannot invade any assets isolated from any entry points. VMCP uses DFD (Data Flow Diagram) to model attack paths and document for each identified attack path the passengers (the legal users of a path), actions (actions that can be done on each of the assets connected to the path), and rule (security criteria).

- **Step 5: Identify attacks.** With the documented attack paths and security criteria tied to them, attack analysis turns to check each of these paths and see how to break the security criteria. Actually some of the paths are very similar in nature, and, from the perspective of adversaries, they are the same and can be broken the same way. So when analyzing attack, we do not need to analyze every attack path. We can group some similar paths together according to certain criteria and pick up one or several typical paths to do an attack analysis. As for each individual path in the group we only pay attention to the particular rules. The output of this step is list of possible attacks along the chosen attack paths.

- **Step 6: Identify Vulnerabilities.** Given an attack path and a set of potential attacks, we can begin to look at ways in which those attacks may be realized. An attack is a threat-specific, implementation-specific, or technology-specific step an attacker could take to realize or help to realize exploiting a system. Through thoroughly analyzing how an identified attack may be realized and how the application defends itself, we can figure out the application's vulnerabilities. VMCP adopts attack tree to analyze attacks and then identify vulnerabilities.

- **Step 7: Rate vulnerabilities**. That is done by evaluating potential impact of each of the vulnerabilities on the system which uses the component. We consider five aspects when analyzing impact: Damage potential, Reproducibility, Exploitability, Affected users and Discoverability (DREAD).

Phase III: Certification Phase. Draw a conclusion on the component in question regarding the security according to the rated vulnerabilities and risks caused.

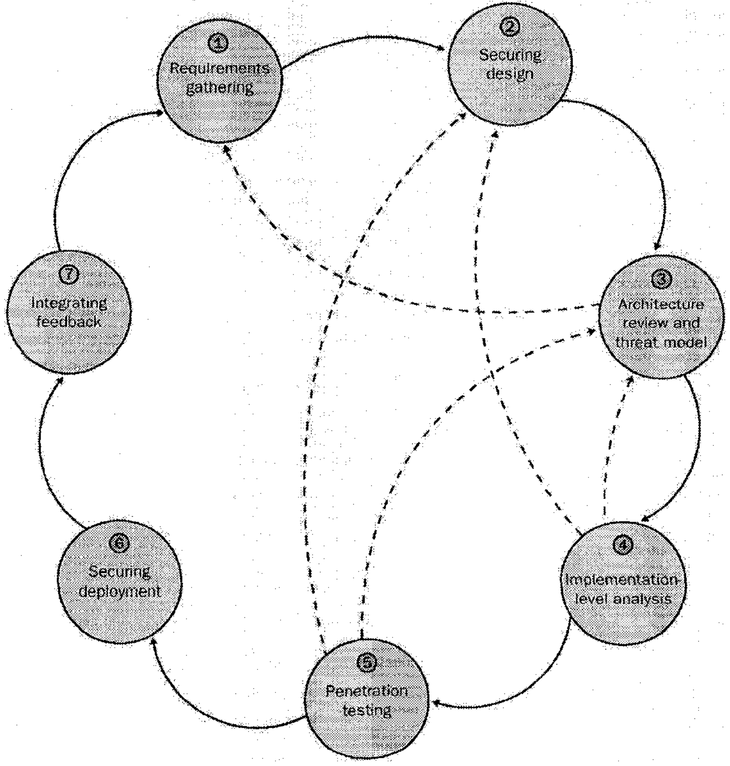
3 Comparison between VMCP and Microsoft's Threat Modeling Process

Although the basic concept of VMCP is derived from Microsoft's threat modeling methodology and some technologies and steps used by Microsoft also appear in VMCP, VMCP is totally different from Microsoft's threat modeling on both the high level concepts and the low level implementation. In this chapter I will explicitly compare and contrast VMCP and Microsoft's threat modeling from these two levels.

3.1 Difference on High Level Concepts

Microsoft's threat modeling is not an independent process. It is only one phase of application security life cycle, which is integrated into the development life cycle to develop a secure application. Figure 3-1 is from [5] and illustrates the 7 phases in the application security life cycle.

Figure 3-1: Phases in the application security life cycle.



In this security life cycle threat modeling is used to identify and analyze the security threats that are always present for the system. Generally threat modeling has two main steps:

1. It first creates a threat profile of the application being developed by assessing and documenting the security risks associated with the application. The threat

profile is an enumeration of the entire adversary's goals for the system.
2. Then it analyzes threats to determine whether they are mitigated.

The outputs are used to refine the security design, plan for penetration testing, and drive code reviews, but how to use them for these three purposes is outside of the scope of threat modeling.

VMCP is an independent product-based security certification process. It is supposed to be used by SCSPs (Software Certification Services Providers) to identify security vulnerabilities (unmitigated threats) associated with a component and assess the risk of these identified vulnerabilities. VMCP has three main steps:

1. First VMCP identifies the potential attacks (threats) based on the component's architecture and design. This is similar to the first step of Microsoft's threat modeling.
2. Then VMCP verifies how the component defends these attacks by reviewing the detailed design and source code. Vulnerabilities are expected to be exposed in this phase.
3. Finally, VMCP analyzes identified vulnerabilities and assesses risks of these vulnerabilities.

In conclusion, VMCP emphasizes on how to identify vulnerabilities (unmitigated threats) and how to assess the risks of these vulnerabilities. VMCP can be considered an extension of Microsoft's threat modeling, as both has similar first step, identify threats (attacks), but VMCP goes much further.

3.2 Differences on Low Level Implementation

Threat modeling is a concept framework but not a practical process. What it proposes are ideas or concepts on how to create a threat profile for an application, as well as some technologies and methodologies that could be used for this process. It does not provide detailed information about how to actually operate the process and how to use those technologies and methodologies in an effective manner.

Threat modeling is built upon the assumption that users of this process are very familiar with all specifications and even source code of the application being modeled. As mentioned above threat modeling is one of the seven phases of application security life cycle and it is designed to be used by development teams, so it is built upon the assumption that these teams are very familiar with all specifications and even source code of the application. As a result all detailed steps are developed focusing on how to identify threats. As to how to determine whether or not a certain threat is mitigated by the architecture design, detailed design or powerful coding, very little is addressed, only several pages in [5].

VMCP is a practical process or even a guideline that can be used by any real life project. VMCP implements, extends and refines Microsoft's threat modeling in a practical manner. It intends to provide Software Certification Services Providers with a step-by-step manual on how to certify components on security. Though VMCP borrows some concepts, technologies and methodologies proposed by threat modeling, VMCP emphasizes on implementing those concepts and using those technologies and methodologies in an effective manner.

VMCP is built upon the assumption that users of this process know nothing about the component at the time when they start to certify the component. VMCP is based on white box technologies. Its key concept is very simple and straightforward: identifying architecture vulnerabilities and implementation vulnerabilities by reviewing design specifications and performing code review. Review design specifications and source code is very time- and effort- intensive. What VMCP really does is to develop an approach to perform these boring tasks effectively and efficaciously.

The following compares VMCP with Microsoft threat modeling step by step.

Step 1: Develop a common taxonomy of security attacks and vulnerabilities.
Threat Modeling does not mention this. It depends much on expert experience to identify threats. VMCP uses its taxonomy as the start point to identify attacks and vulnerabilities. It covers the most common application level attacks and vulnerabilities that plague web-based enterprise applications. For each category within the taxonomy, VMCP first identifies the attacks and then the related vulnerabilities that may pose the risks.

Step 2: Gain an architecture overview from the security perspective.
This step in VMCP is identical to "Step 2: Create an architecture overview" of the modeling process presented in [20].

Step 3: Model the component from an adversary's View.
The modeling process present by [5] has a similar step "Understanding the Adversary's View" where it defines three elements of a system that are interesting to any adversary whoever wants to attack the system, and briefly describes how to collect these elements.

- Entry Points. Where data or control transfers between the system being modeled and another system.
- Assets. Resources the component or system has that an adversary might try to modify, steal, or otherwise access or manipulate.
- Trust Levels. They define the privilege that an external entity should have to legitimately use an entry point or functionality at the entry point, and they dictate which assets external entities should normally be allowed to access or affect in some way.

VMCP adapts this step from threat modeling by making the following enhancements:

- Import Entry Points and Assets are still utilized, but two new features are added: "trust roles" and "category". These two features play important roles in later attack analysis.
- Replace Trust Levels with Trust Boundaries, which refer to places where trust levels change. A trust boundary can be imaged as a line drawn through a component. On one side of the line, data is un-trusted. On the other side of the line, data is trustworthy. (See charter 7 for more details).
- Provides more detailed approaches to collect these elements.

Step 4: Model attack paths and security criteria.
When modeling applications, Microsoft uses traditional DFDs. No detailed information is provided on how to use DFDs to model applications, except some basic concepts such as start from the high level and gradually descend to the desired low level.

VMCP creates and implements an innovative approach to model applications using EDFD (Enhanced Data Flow Diagram, a term introduced by VMCP). The following new concepts and approach have been defined and implemented:

- **Attack Path.** Traditionally attack paths refer to an unmitigated path from the root node to a leave in an attack tree or threat tree. In VMCP, an attack path is redefined as a logical path that connects entry points and assets, through which external or internal data or controls flow from the entry points to the assets.

- **Security Criteria.** Security rules associated with an attack path. It defines who the legal passengers of an attack path are and what actions these passengers can perform on this attack path.

- **EDFD** (Enhanced Data Flow Diagram). The traditional DFD has been enhanced for the benefit of vulnerability modeling by replacing External Entities with Entry Points, replacing Data Stores with Assets and adding Trust Boundaries into the diagram.

- **A new approach.** VMCP develops and implement a new, concrete approach to model attack paths and security criteria using EDFD.

Step 5: Identify attacks.
Threat modeling proposes the concepts of identifying threats by correlating threats and assets, and starting with known vulnerabilities. Again it emphasizes the concepts and no concrete approach is proposed. Threats identified are classified using STRIDE [67].

VMCP develops and implements another innovative approach for identifying attacks. This approach is a combination of inside-out and outside-in approaches based on the attack paths, security criteria and the taxonomy developed in step 1. The basic idea is to check attack paths and see if and how to break the security criteria with the help of the taxonomy. As to attack clarification, VMCP adopts STRIDE too, similar to threat modeling.

Step 6: Identify Vulnerabilities.
Threat modeling uses threat trees to analyze threats and determine if there are unmitigated paths existing. These unmitigated paths are vulnerabilities. Threat modeling proposes performing code and design review on the most potential places to identify vulnerabilities but nothing is discussed on how to determine and find out where are these most potential places, assuming that you are a member of the development team and should know where these potential places are

.

VMCP uses the same technologies to identify vulnerabilities, attack trees (threat trees), code and design reviews, as they are the commonly used V&V technologies. The difference is VMCP explains and provides guidance with regard to the process, e.g. it discusses where are the most likely locations of threats and how to identify both architecture and implementation vulnerabilities using attack trees, attack paths, security criteria and the taxonomy. In order words, VMCP tells you how to perform code and design review in an effective and efficacious manner, specifically for vulnerability determination rather than the more traditional defect removal. Also VMCP has something new in this step:

- Enhancing the traditional attack tree by adding a pre-condition branch into the diagram; and
- Importing the concept of an Attack Pattern Library

These two points are very helpful in identifying vulnerabilities, and will be the corner stones of computationally automatic analyzing of attacks.

Step 7: Rate vulnerabilities.
Threat modeling use DREAD method to assess risks associated with identified threats. It gives a brief definition of different security levels for each metric of D.R.E.A.D, but no information on how to use exist tools or methods to assist the assessment. Threat modeling uses continual numbers such 1 to 10 to score threats, which does not consider the qualitative nature of threats. The problem is how you can tell the exact difference between 2 and 3 when estimating potential damage.

VMCP also uses the DREAD method to assess risks associated with identified vulnerabilities. Differences are:

- VMCP uses discrete numbers such as 5, 10 and 15 to score vulnerabilities, combining the advantages of both the quantitative method and the qualitative method. This not only takes into account the qualitative nature of vulnerabilities but makes it possible to perform quantitative analyst on them.
- VMCP uses attack trees and assets to assist in assessing risks. Detailed information is provided.

Step 8: Certify the component.
Obviously this step is exclusive to VMCP. VMCP certifies the security level of a component based on the score calculated in step 7.

4 Security Attacks & Vulnerabilities Taxonomy

This taxonomy will be used as the start point in later steps to assist with identifying attacks and vulnerabilities. It focuses on the most common application level attacks and vulnerabilities that plague web-based enterprise applications. Effect-cause analysis technologies were used to develop this taxonomy, thinking of the attacks as the effects and the vulnerabilities as the causes whose existence in the application makes the attacks realizable. Some attacks are very similar in nature, for example SQL injection and cross-site scripting are both execute malicious code in the backend of applications. Moreover, these attacks are caused by the same vulnerabilities, say using non-validated input or relying upon only client-side validation. Therefore we grouped these attacks and related vulnerabilities together as one category. So far this classification is by no means thorough and detailed enough for real business projects because developing a thorough taxonomy is complex enough for a separate research topic and is out of the scope of this thesis. At this stage I classified security attacks and vulnerabilities into nine groups. Appendix D is a sample taxonomy which has detailed information about these nine groups.

1. **Input and data validation**
2. **Authentication**
3. **Authorization**
4. **Session Management**
5. **Insecure Data Storage**
6. **Insecure Configuration Management**
7. **Cryptography**
8. **Parameter Manipulation**
9. **Exception handling, Auditing and Logging**

Input and data validation

Input and data validation requires applications not to blindly trust any input or data before they pass the validation of the type, length, format, range or even the content. It is a must to validate the input or data before processing them. An attacker can compromise your application if any such vulnerability is identified. Applications that do not perform input and data validation are susceptible for following attacks.

- Buffer Overflow
- Cross-site scripting
- SQL injection
- Canonicalization
- Format string attacks

Authentication

Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services by validating the user with whom they claims to be. This is typically achieved through credentials, such as a user name and password. These clients may be end users, other services, processes, or servers. Following are the possible attacks that an attacker can conduct to exploit failures in an application.

- Brute force attacks
- Dictionary attacks
- Cookie replay attacks
- Credential theft

Authorization

Authorization addresses the question: what can you do? It is all about granting or denying access to the resources and operations for which the authenticated client requests access. Authorization is usually accomplished based upon user identity and role membership. Resources include files, databases, tables, rows, and so on, together with system-level resources such as registry keys and memory data. Operations include performing transactions such as purchasing a product, depositing and withdrawing money from one account, or updating user profile. Top attacks that exploit authorization are

- Elevation of privilege
- Disclosure of confidential data
- Luring attacks

Session Management

In order to provide a friendly environment to the users, web-based applications often use sessions to maintain states through user's subsequent requests. Sessions are stored on servers and linked to users by session IDs. Session IDs are an attractive target for hackers as they can act as the associated users once they get their session ID. Moreover, sometimes applications store sensitive information in the session objects managed by the application layer. The attractive session ID and sensitive information stored in the session objects lead to potential attacks. They include:

- Session hijacking
- Session fixating
- Session forging
- Session replay

Insecure Data Storage

There is a misunderstanding that if the encryption is strong enough no sensitive data will be stolen. However encryption may be totally compromised by a single vulnerability. This answers the question why devastating thefts of sensitive data continue to occur even though enterprises worldwide spent approximately $20 billion per year on IT security. Sensitive data is always at great risk as it is always the target of malicious attacks. Most of the security cost and effort are usually spent on protecting sensitive data. Common attacks regarding data storage are:

- Unauthorized access to data in storage
- Unauthorized access to data in memory
- Network eavesdropping
- Data tampering

Insecure Configuration Management

Today web applications frequently use services provided by the application server and/or web server such as data storage, directory services, mail and so on. However the component development group (provider) is separate from the group using the component (consumer). Very often a wide gap between those who write the component and those responsible for the operations environment (consumers) is created by the improper assumptions made by the writers that how consumers will configure their server. Web application security concerns often span this gap. In addition, Most of the web applications are configurable and store the configuration parameters in files or databases. To facility management of configuration, applications normally provide configuration management interfaces to allow users with high privileges, say administrators, to change configuration parameters and perform maintenance. This makes the situation even worse. The following are common attacks due to insecure configuration management.

- Unauthorized access to configuration management interfaces
- Unauthorized access to configuration stores
- Retrieval of plain text configuration secrets

Cryptography

Today most web-based applications use cryptography to protect sensitive information when transmitted and stored. Basically cryptographic systems can provide four services: authentication, non-repudiation, confidentiality and integrity. Cryptography is one of the most advanced topics of application security and there are many approaches to encryption, each with advantages and disadvantages. Very often expert experience is needed when architects and developers try to choose a cryptography approach and implement it correctly and accurately. A small mistake in configuration or coding may result in a useless cryptography. Typical cryptographic attacks are:

- Cryptographic key attacks

Parameter Manipulation

Manipulating the data sent between the browser and the web application is a simple but effective way to change application behaviors. Information captured from the browser is usually sent to the server in one of these four formats: URL query string, form fields, cookies and HTTP headers. In a badly designed and developed web application, malicious users can modify data before it is be transmitted so even cryptographic protection in the transport layer (SSL) is insufficient. Parameter tampering can often be done with:

- URL Query String
- HTML Form field
- Cookie
- HTTP header

Exception handling, Auditing and Logging

Exception handling, auditing and logging are three different aspects of the same topic: how to track events within an application. Applications should always fail safe. When an application fails to an unknown state, the exception information shown might not be making sense for the end user but might be a very interesting message for an attacker. Motivated attackers may be able to exploit this indeterminate state to access unauthorized functionality, or worse manipulate data. Well-written applications enable auditing and logging to easily track or identify potential fraud or anomalies end-to-end. This helps to identify which user is trying to exploit and what actions have been done. With this kind of information necessary actions can be taken to prevent the system from such attacks. The following attacks are related to this area.

- Detailed error message attacks
- Repudiation
- Escape from being traced
- Cover tracks

5 Gain an architecture overview from the security perspective

Basically all security vulnerabilities fall into two types: implementation and architecture vulnerabilities. Implementation vulnerabilities are related to how the component is implemented. For example, no server side data validation will make injection attacks possible. Architecture vulnerabilities are related to how the component is designed and what kinds of technologies are used. For example, any session-based component is prone to the following architectural-level vulnerabilities: session identifier replay, unsecured session identifiers and injection attacks.

Today there are many mature technologies addressing implementation security problems. One common method that has become prevalent is using code reviews. However even a thorough code review will not discover architecture-level vulnerabilities. The reasons are obviously. Code review, conducted either by humans or by automatic scanners, focuses on a very limited context, reviewing the source code line by line. Moreover, a code review can only discover security problems that have been written into the applications. Many security problems are caused by the technologies used and how the component is deployed, rather than bad code.

Currently, security experts rely on their experience to recognize architecture-level security issues in components. Teams that are trying to understand the risk their applications face have broad guidelines to work with and use methodologies that often center on brainstorming for analyzing component. The vulnerability modeling process described in this thesis enables anyone trying to examine the security architecture of a component to work with a procedural approach to identify commonly known architecture flaws, and to identify new issues—including those specific to a particular component. VMCP makes the application security analysis less reliant on intuition and allows people with less experience in security analysis to evaluate a component's security strength.

In this chapter, I will present a procedure of how to obtain an architectural overview of the component in question from the security perspective. This is the base stone of analyzing architecture vulnerabilities. To understand the component from the perspective of security, VMCP tries to:

- **Draw the end-to-end deployment scenario.**
- **Identify roles.**
- **Identify key usage scenarios.**
- **Identify security mechanisms.**
- **Identify technologies.**

Draw the End-to-End Deployment Scenario

How will the component be deployed determines the component's architecture at the high level. For example if the component will be deployed in a distributed fashion, the client-server architecture pattern is the best choice. To obtain a good understanding of the component, the first step is thinking of how it is deployed. Moreover some attacks and vulnerabilities are related to a particular architectural pattern. Cookie manipulating and HTML Form field manipulating will never happen to non-web-based components. When drawing the deployment scenario, we first draw a big picture that includes the sub-composition and structure of the component, its subsystems, and its deployment characteristics. Then add details about the authentication, authorization, and communication mechanisms to the big picture.

In general, the deployment diagram should include the following:

- **End-to-end deployment topology.** The topology should indicate the layout of the servers and the access of intranet, extranet, or Internet. We can start with logical network topologies, and then refine them to more detailed physical topologies.
- **Logical layers.** These layers are the software layers of the component, indicating where and how the presentation layer, business layer, and data access layers reside. Logical layers need to be refined to include physical server boundaries.
- **Key sub-components.** It is not possible and necessary to include every sub-component in the deployment scenario diagram. Only the important components within each logical layer should be included.
- **Communication ports and protocols.** We need to know which servers and components communicate with each other and what protocols they use, e.g. HTTP, or HTTPS. The specifics of inbound and outbound information packages should be included.
- **External dependencies.** If the component has dependencies on external systems they should be indicated. Later in the modeling process, this will help us identify attacks and vulnerabilities that can arise if any assumptions the component makes about the external systems are false or if the external systems do not work as expected.

Identify Roles

Identify who can do what and cannot do what within the component. For each role, we need to figure out what are the legal and illegal activities. In other words, what is supposed to and not supposed to happen. Particular attention should be paid to higher-privileged groups of users. Below is an example from appendix A: Sample I – JSPCART. JSPCART is an online Shopping Cart which has three roles:

Anonymous user, Authenticated user and Administrators. Each role has legal and illegal activities.

Identified Roles of JSPCART

Anonymous user:

Legal activities:

- Browse product or catalog list
- Create a cart, and browse or modify items in the cart
- Create a user account

Illegal activities:

- Checkout items in a cart created by his/her own.
- Browse, modify or checkout items in carts created by others
- Browse or modify user profile
- Change products or catalogs information
- Browse or modify orders

Authenticated user:

Legal activities:

- Browse product or catalog list
- Create a cart, and browse, modify or checkout items in the cart.
- Create a user account
- Browse or modify his/her own profile
- Browse or modify his/her own orders

Illegal activities:

- Browse, modify or checkout items in carts created by other users
- Browse or modify profiles of other users
- Change products or catalogs information
- Browse or modify orders created by other users

Administrators:

Legal activities:

- Full control on user accounts, carts, products, catalogs and orders.

Illegal activities:

- None

Identify Key Usage Scenarios

Key usage scenarios can be derived from use cases that come with other specifications. At least they should cover all important features of the component and identify the dominant application functionality and usage, especially in the Create, Read, Update, and Delete aspects. This kind of information helps us understand how the component is intended to be used and how it can be misused.

When discovering key usage scenarios, we need to avoid attempting to analyze every possible use case. Instead, we focus on the main use cases that exercise the predominant Create, Read, Update, and Delete functionality of the component. For example, the important usage scenarios of the JSPCART are:

- Anonymous user browses the product pagers.
- Anonymous user adds and/or removes items to the shopping cart, modify the item quantity.
- Anonymous user logs in to authenticate prior to placing an order.
- Anonymous user creates a new account prior to placing an order.
- Authenticated user places, browses and modifies an order.
- Authenticated user browses and/or modifies his/her user profile.
- Administrator manipulates user profiles, products, catalogs and orders.

Identify Security Mechanisms

The purpose of this effort is to gain high level knowledge of the security mechanisms used by the component. For example, we might know how the component is authenticated by the database or how users are authorized. We might know what mechanisms are used to perform authentication and authorization and where they are performed. We might also know certain details about if there is server side data validation and how it is to be performed.

Basically we need to identify any key points that we can derived from the specifications about the following:

- Authentication
- Authorization
- Input and data validation
- Configuration management
- Sensitive data
- Session management
- Cryptography
- Parameter manipulation
- Exception handling, Auditing and logging

Identify Technologies

Technologies used by the component contribute much to the security. For example if the component is developed in C, you have to pay special attention to potential buffer flows. Identifying technologies also helps us to focus on technology-specific attacks and vulnerabilities later in the modeling activities. When identifying technologies, we focus on the following aspects:

- Operating systems
- Web server software
- Database server software
- Technologies used in the presentation, business, and data access layers
- Development languages

6 Model the component from an adversary's View

VMCP uses a defensive approach to certificate security. For a specified component, VMCP first figures out all potential attacks that an adversary might use to exploit the component. For each attack, VMCP verifies how the component defends against the attack. To model the adversary view VMCP is based on the following three premises:

- **An adversary will not attack the system without assets of interest.** This first principle is very easy to understand. An adversary's goals are always based on the system's assets. The adversary has no reason to attack the system unless it contains something of value to him/her—for example, corporate or personal data, processing resources, or financial information. Assets are the resources the component or system has that an adversary might try to modify, steal, or manipulate. Assets can be tangible, such as data stored in the database or more abstract (intangible), such as such as the availability of services. Assets are the basis for attacks. It is impossible to have an attack without a corresponding asset because assets are essentially attack targets.

- **An adversary cannot attack a system without entry points, interfaces the system has with the outside world.** Entry points are any location where data or control transfers between the system being modeled and another system. In most cases, the adversary must actively jeopardize the application's security via entry points. Entry points show all the places where the adversary can attack the system from, including transfer points such as open sockets, remote procedure call (RPC) interfaces, Web services interfaces, and data being read from the file system. Entry points are not only the places where data or commands flow into the system but includes the points where data or information flows out of the system. The remaining passive attacks (those attacks where the adversary simply consumes data from the application) are information disclosure attacks. In this case, the adversary is still interacting with the application by listening on the appropriate channel, which might be a network, an event, or another message channel. For example, some wireless applications broadcast information that is unsolicited. A passive attacker who listens for this information could discover valuable information about the application.

- **An adversary always attacks the system at the weakest points, the trust boundaries.** By identifying all assets of interest and entry points, adversaries gather the basic information for undertaking attacks, what are the attack targets and where they can enter the system to reach the assets. Then what is left is to figure out where is the most possible points they can break the system, in other words the weakest points of the system. In great part these are the trust boundaries, where privilege of an external or internal entity to access system's assets changes. A login page is an example of external trust boundary as an external user's privilege changes after the authentication procedure. The interface between application and back-end database is an example of internal trust boundary. Database usually has its own authentication procedure for each application instance trying to connect to it.

VMCP use a systematic method to model the component from these three aspects, assets of interest, entry points and trust boundaries, to get enough information for later attack analysis.

6.1 Identify Assets

Definition of an Asset

Assets are abstract or concrete resources that the system must protect from incorrect or unauthorized use by adversaries. Assets can be tangible, say user account, or intangible like the availability of service.

- **Tangible assets.** Tangible assets are physical assets such as data stored in database or configure file stored on a disk. Tangible assets also include non-persistent assets. Tangible assets are generally easy to understand and identify.
- **Intangible assets.** Intangible assets are abstract assets that cannot be seen, touched or physically measured, such as the availability of services or processes running on a server. Intangible assets are easy to ignore.

Assets, either tangible or intangible, can be transitive through their relationship with other assets. A component within a system usually is not independent but interacts with other components. If a component acts as a gateway to the functionality and assets of the other components, the assets of the other components are called the transitive assets of the gateway component. Let's think about an access control component in a file system. The component implements an authorization mechanism to check entries on a resource to see whether a user has access rights to that resource. We can consider the resources as a transitive asset of the access control component because other components determine whether to grant access based solely on the information returned by the access control component, even though it does not interact directly with that resource and simply checks the access control list on behalf of others.

How to identify Assets

Identifying assets can be one of the more difficult parts of the VMCP, and if you are going to miss something, it is probably an asset. VMCP uses security objectives to guide the identification of assets. Security objectives are goals the component is developed to achieve, or constraints the component is developed under. Security objectives are often described in terms of constraints. For example, an unauthorized user must not change account information. VMCP classifies security objectives into four catalogues:

- **Confidentiality.** How the component protects against unauthorized information disclosure.
- **Integrity.** How the component prevents unauthorized information changes.
- **Availability.** How the component protects provide the required services even while under attack.

- **Traceability and auditing.** How the component find out who did what and when they did.

Security objectives are very helpful when we try to identify assets. One place to start is by looking at what are the objects of each security objective. Also, look at what nouns are used repeatedly in security objectives. For example, if one of the security objects is to protect customer account details as sensitive data, then we can identify customer account details as assets as this kind of information is interesting to the attackers. To identify the security objectives, consider the following questions:

- **What is the sensitive data does the component needs to protect?** Sensitive data, like use user credential, customer credit card numbers, financial history and transaction records, are always interesting to attacks. They are the most critical asset.
- **Are there any compliance requirements?** Is the component developed under any compliance requirements such as security policy, privacy laws, regulations, and standards?
- **Are there intangible assets that you need to protect?** The most common intangible asset is the quality of service requirements including availability and performance requirements.
- **What kinds of activity should be recorded for further auditing?** Activities on sensitive data should be recorded, such as update, delete, and store records in production price table.

The following are examples of some common security objectives:

- Prevent attackers from obtaining sensitive customer data, e.g. profile information.
- Meet service-level agreements for application availability and performance.
- Record activities on sensitive customer data.

How to record assets

When identify assets VMCP gathers the following information.

- **ID** A unique number assigned to the asset, which is used to cross-reference assets with attacks and vulnerabilities later in the modeling process.
- **Name** A short title for the asset, which should be descriptive enough to identify the asset—for example, User credentials, order data, and availability of service.
- **Description** The brief description of the asset.
- **Trust Roles** The trust roles are those who are normally allowed to

access or otherwise interact with the asset. The allowed access (authorization) of each role to the asset should also be document. VMCP checks four types of access, C: Create, R: Retrieve, U: Update, D: Delete.

- **Category** The category to which the asset belongs to. Currently we classify the assets into three categories
    - **Application data** Transient or persistent data meaningful to business logics such as Order and Account.
    - **Non-Functionality** Assets related to non-functionality implementation. Besides business logics components usually implement some general non-functional requirement say auditing and monitoring.
    - **System resource** Assets of the environment where the component is running, e.g. files stored in the file system.

In some systems, it is easier to identify assets if they are grouped according to the part of the system they belong to. This is particularly useful in Web-based applications and other multi-user systems. For example, the assets of a Web commerce application that processes user orders could include the users' credit card numbers, the purchase invoices, and the website availability. These assets might belong to the application data, and non-functionality categories, respectively. Categorizing assets in this manner can ensure that significant types of assets are not missed altogether.

Examples

This section contains identified assets from the three samples: an online shopping cart Jspcart, Duke's Bank Application and Credit Card Payment Component. Appendix A, Appendix B and Appendix C have the complete information of these three samples.

Table 6-1-1: Asset Table of Jspcart

ID	Name	Description	Trust Roles	Category
A1	User credentials	Username and password	Authenticated user (CRUD)	Application data
A2	User profile	User profiles stored in back-end database	Authenticated user (CRUD)	Application data
A3	Cart	Cart information stored in back-end database, e.g. the name and quantity of items in a cart.	Authenticated user (CRUD) Anonymous user (CRUD)	Application data
A4	Order	Orders stored in back-end database	Authenticated user (CRUD)	Application data
A5	Product	Products stored in back-end database	Anonymous user (R) Authenticated user (R) Administrator (CRUD)	Application data
A6	Shopping	Shopping information for orders.	Authenticated	Application

	information		user (CRU) Administrator (CRUD)	data
A6	Catalog	Catalogs stored in back-end database	Administrator (CRUD)	Application data
A7	Process	Processes running within the same machine where the component is running	N/A	System resource
A8	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource
A9	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A10	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

C: Create, **R**: Retrieve, **U**: Update, **D**: Delete.

Table 6-1-2: Asset Table of Duke's Bank Application

ID	Name	Description	Trust Roles	Category
A1	User credentials	Username and password	BankCustomer, BankAdmin (CRUD)	Application data
A2	Customer Data	Customer information stored in back-end database	BankAdmin (CRUD)	Application data
A3	Account Data	Account information stored in back-end database.	BankAdmin (CRUD) BankCustome (RU)	Application data
A4	Transaction Data	Transaction information stored in back-end database	BankCustome (CR)	Application data
A5	Process	Processes running within the same machine where the component is running	N/A	System resource
A6	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource
A7	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A8	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

C: Create, **R**: Retrieve, **U**: Update, **D**: Delete.

Table 6-1-3: Asset Table of Credit Card Payment Component

ID	Name	Description	Trust Roles	Catalogue
A1	Credit Card Information	The cardholder, card number and expire date, stored in memory.	Invoker(R)	Application data
A2	Order	Order information stored in memory.	Invoker(R)	Application data
A3	Configuration Data	Configuration information stored in back-end database.	Invoker(CRUD)	Application data
A4	Geographic Zones	Geographic Zones and countries.	Invoker(R)	Application data
A5	Process	Processes running within the same machine where the component is running	N/A	System resource
A6	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource
A7	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A8	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

C: Create, **R**: Retrieve, **U**: Update, **D**: Delete.

6.2 Identify Entry Points

Definition of Entry Points
Entry points are interfaces where a component interacts with the outside world, and where control and data crosses the boundary of the component. Entry points include all junctions between the system and the external environment. They are also called attack points from the perspective of adversaries, as entry points represent a means of interacting with the system. Entry points are used by VMPC to determine the functionality that an adversary can exercise on the component and what assets s/he can affect. Basically VMCP treats entry points as two types: external entry points and internal entry points.

- **External Entry Points.** These types of entry points are intended to be exposed to the external environment of the system within which the component is used. Clients or attackers have direct access to these Entry Points. The following are common external entry points.
  - **User interfaces**. Where users interact with the component. Components use user interfaces to accept use input. If the input and data validation mechanism is not secure enough, lots of attacks can be conducted via user interfaces such as buffer overflow, cross-site scripting, SQL injection and Format string attacks.
  - **Network interface**. Where component interact with network. For example, web server listening port is a network interface where the front-end Web application listening for HTTP requests. This entry point is intended to be exposed to clients. Attackers can bypass the web browser and set data directly to the server. Also attackers can exploit sensitive data using network monitoring software that can capture traffic leading to host which is on the same network.

- **Internal Entry Points.** These types of entry points are intended to be exposed to the internal environment of the system within which the component is used. Clients or attackers do not have direct access to these entry points but they can indirectly interact with them. The following are common internal entry points:
  - **System interface.** Where the component interact with the environment it residents in. Components may read data from the file system or configuration store, such as a registry. If attackers can manipulate these data by attacking the file system or registry then they can change the behaviour of components.
  - **Component interface**. Where the component interact with other components. These entry points exposed by subcomponents across the layers of a component may exist only to support internal communication with other components. Components usually have APIs through which other components or systems can invoke functions provided. If attackers can control the way a component or system invokes the APIs they can indirectly control the

behaviour of the invoked components.

How to identify entry points

Entry points are actually the interfaces between a component and external and/or internal environment so the most efficient way to identify entry points is to identify the component's interfaces. As mentioned above, most entry points fall in one of these four types: user interfaces, network interfaces, system interfaces and component interfaces. When identifying entry points, VMCP focuses on these four types of interfaces of a component, which usually cover almost all the entry points. Also, VMCP checks the following three aspects.

- **Exit points.** Exit points are where a component sends data or message to clients or to external systems. VMCP treat exit points as entry points because they share similar characteristics and both are refer to locations where control or data moves between the component and the external world. Although exit points only pose threats of information disclosure, hackers might gather useful data at an exit point to perform other types of attacks. Thus, it's necessary to include exit points in the entry point enumeration, particularly these where a component outputs data that includes client input or includes data from entrusted sources, such as shared databases.

- **Layered Entry Points**. In web-based applications (components), entry points can be layered. Basically each Web page can be considered an entry point as it might be used by an attacker to interact with the application. However a certain page can provides multiple disparate functions based on the parameters carried by the URL string or Form items. For example, an online shopping system has a Web page called MyOrder.jsp. This page depending on the Action parameter may perform different actions. A request for / MyOrder.jsp?Action=View might show the order details, whereas / MyOrder.jsp?Action=Delete might delete the order. In this example, the View and Delete functions are layered entry points on the MyOrder.jsp page.

- **Level of Granularity.** When identifying entry points we have to make trade off between the completeness of entry points and the effectiveness of the modeling process. Entry points should be identified to a level that is enough to cover all component functionality but not so granular that they overwhelm the modeling process. In general, the entry points outlined will be detailed enough to identify all unique potential attacks. Initially, VMCP focuses on higher-level entry points, and over time, include more granular entry points.

How to record entry points

When identify assets VMCP gathers the following information.

- **ID** A unique number assigned to the entry point, which is used to cross-reference the entry point with attacks and vulnerabilities later in the modeling process.
- **Name** A short title for the entry point, which should be descriptive enough to identify the entry point—for example, Web server listening port or Login Page.
- **Description** A brief description of the entry point.
- **Trust Roles** The trust roles are those who are normally allowed to access or otherwise interact with the entry point.
- **Category** The category to which the entry point belongs to. Currently we classify the assets into four categories
  - User interface. Where users interact with the component. Components use user interfaces to accept use input.
  - Network interface. Where component interact with network, say web server listening port.
  - System interface. Where the component interact with the environment it residents in.
  - Component interface. Where the component interact with other components.

  The classification of entry points is very important for later attack analysis. Each type of entry points has its own unique characteristics and vulnerabilities. For example, the way an adversary attacks an entry point of user interface type is totally different from that he/she attacks an entry point of system interface type.

Examples

This section contains identified entry points from the three samples: an online shopping cart Jspcart, Duke's Bank Application and Credit Card Payment Component. Appendix A, Appendix B and Appendix C have the complete information of these three samples.

Table 6-2-1: Entry Point Table of Jspcart

ID	Name	Description	Trust Roles	Category
EP1	Web server listening port	The port on which the Web server listens. All web pages are layered on this entry point.	All	Network Interface
EP2	Login Page	Get user credentials and passed them to server side for authentication	All	User Interface
EP3	Signup Page	Create a user profile and pass it back to server side	All	User Interface
EP4	Cart Page	Users modify quantity of items and remove items from cart.	All	User Interface
EP5	ChangePas	Authenticated users change password	Authenticated	User Interface

	sword page		user	
EP6	ChangeProfile page	Authenticated users change profile	Authenticated user	User Interface
EP7	Shipping page	Authenticated users input shipping information	Authenticated user	User Interface
EP8	MyOrder Page	Authenticated users enquiry order details and/or cancel orders	Authenticated user	User Interface

Table 6-2-2: Entry Point Table of Duke's Bank Application

ID	Name	Description	Trust Roles	Category
EP1	Web container listening port	The port on which the Web container listens. All web pages are layered on this entry point.	All	Network Interface
EP2	EJB container listening port	The port on which the EJB container listens. Remote application client communicates with application server via this port.	All	Network Interface
EP3	Logon Page	Get user credentials and passed them to server side for authentication	All	User Interface
EP3	Account List Page	List account details of a customer and transaction history of a account	BankCustomer	User Interface
EP4	Transfer Funds Page	Transfer funds between accounts	BankCustomer	User Interface
EP5	ATM Page	Withdraw and deposit funds	BankCustomer	User Interface
EP6	Customer Info GUI (Application Client)	Manipulate customer information	BankAdmin	User Interface
EP7	Account Info GUI (Application Client)	Manipulate account information	BankAdmin	User Interface

Table 6-2-3: Entry Point Table of Credit Card Payment Component

ID	Name	Description	Trust Roles	Catalogue
EP1	update_status()	Determine if the credit card payment is enabled.	Invoker	Component interface
EP2	javascript_validation()	Return a snippet of input validation javascript program.	Invoker	Component interface
EP3	selection()	Return Forms which accept user's input	Invoker	Component interface
EP4	pre_confirmation_check()	Validate the credit card accepted.	Invoker	Component interface

EP5	confirmation()	Return masked credit card information	Invoker	Component interface
EP6	process_button()	Return Forms with hidden fields of credit card information.	Invoker	Component interface
EP7	before_process()	Store card number to order	Invoker	Component interface
EP8	after_process()	Email order information to customer.	Invoker	Component interface
EP9	install()	Store configuration information to the configuration table.	Invoker	Component interface System interface
EP10	remove()	Remove configuration information from the configuration table.	Invoker	Component interface

6.3 Identify Trust Boundaries

Definition of Trust Boundaries
Trust boundaries refer to places where trust levels change. A trust boundary can be imaged as a line drawn through a component. On one side of the line, data is un-trusted. On the other side of the line, data is trustworthy. When data or control wants to cross the trust boundary (to move from un-trusted to trusted), it must pass a security check just as what we do at the airport. The security check in a component is validation logic that prevents bad data or control from entering the component. Take an online banking system for example, the trust level changes after a user logs into the application. Before this he cannot complete any transactions. Once they successfully log into the system, they can do any thing that the logon ID has privilege to achieve. In this case we think there is a trust boundary between the logon page and the web application. To facility identifying trust boundaries, VMCP classifies trust boundaries into four categories.

- **User Interface Boundary.** This is a trust boundary between a UI and the component. The data users input through a GUI or Web page becomes trusted by the component once it goes cross the boundary.

- **Service Boundary.** This is a trust boundary between the component and an external service provider or requestor. A component might call other components to request a service or it might provide services to others. Components must have an effective mechanism to ensure that the data returned is valid and is from the appropriate service providers, or only the appropriate callers are allowed access to request service from it.

- **Network Boundary.** This is a trust boundary between the component and Internet or intra-net. The firewall is a typical network trust boundary. It moves qualified information from the un-trusted Internet to the web application. The listening port of an http server or application server is another typical network boundary of web applications. Data or requests from Internet become trustworthy while they cross this boundary.

- **System Boundary** This is a trust boundary between the component and the system it resides on or connects to. Components usually interact with a file system or database to store or retrieve information. Vulnerabilities might be found here if components trust the information without a good reason.

How to identify Trust Boundaries
When identifying trust boundaries VMCP uses the following 3 steps:

- Start by identifying boundaries between a component and the external environment or systems. For example, a component may read configuration files from a file system, it may make calls to the database server, or it may call a remote service provider for services. There must be trust boundaries between a component and the external environment or systems systems.

- Check high privileged places where access requires additional privileges. If users want to access to these places they have to obtain prevalent privileges, which means the trust level changes. In other words, there are trust boundaries keeping users from directly going to these places. For example, an administration page is restricted to managers. The page requires high privileges and also requires that the user is a member of an administrator role.

- Check data or process flow to identify trust boundaries. We do not have to dig into the detailed diagrams deeply but just to the extent that we can tell the places where the upstream data flow or user input becomes trusted from un-trusted and how the data flow and input is authenticated and authorized.

How to record Trust Boundaries

When identify assets VMCP gathers the following information.

- **ID** A unique number assigned to the trust boundary, which is used to cross-reference the entry point with attacks and vulnerabilities later in the modeling process.
- **Name** A short title for the trust boundary, which should be descriptive enough to identify the entry point.
- **Description** The brief description of the trust boundary.

Examples

.This section contains identified trust boundaries from the three samples: an online shopping cart Jspcart, Duke's Bank Application and Credit Card Payment Component. Appendix A, Appendix B and Appendix C have the complete information of these three samples.

Table 6-3-1: Trust Boundary Table of Jspcart

ID	Name	Description	Category
TB1	Client Boundary	The boundary between remote clients and backend applications in server side.	Network Boundary
TB2	Login Boundary	The boundary between the login model and other models	User Interface Boundary
TB3	Database Boundary	The boundary between the application and database.	System Boundary

Table 6-3-2: Trust Boundary Table of Duke's Bank Application

ID	Name	Description	Category
TB1	Web Clinet Boundary	The boundary between remote web client and backend application on the server side.	Network Boundary
TB2	Application Clinet Boundary	The boundary between remote application client and backend application on the server side.	Network Boundary
TB3	Database Boundary	The boundary between the application and database.	System Boundary
TB4	Logon Boundary	The boundary between the logon model and other models	User Interface Boundary

Table 6-3-3: Trust Boundary Table of Credit Card Payment Component

ID	Name	Description	Category
TB1	API Boundary	The boundary between outside invokers and the component.	Service Boundary
TB2	Database Boundary	The boundary between the component and database.	System Boundary
TB3	Memory Data Boundary	The boundary between processes and Memory Data	System Boundary

7 Model Attack Paths and Security Criteria

7.1 Attack Path and Security Criteria

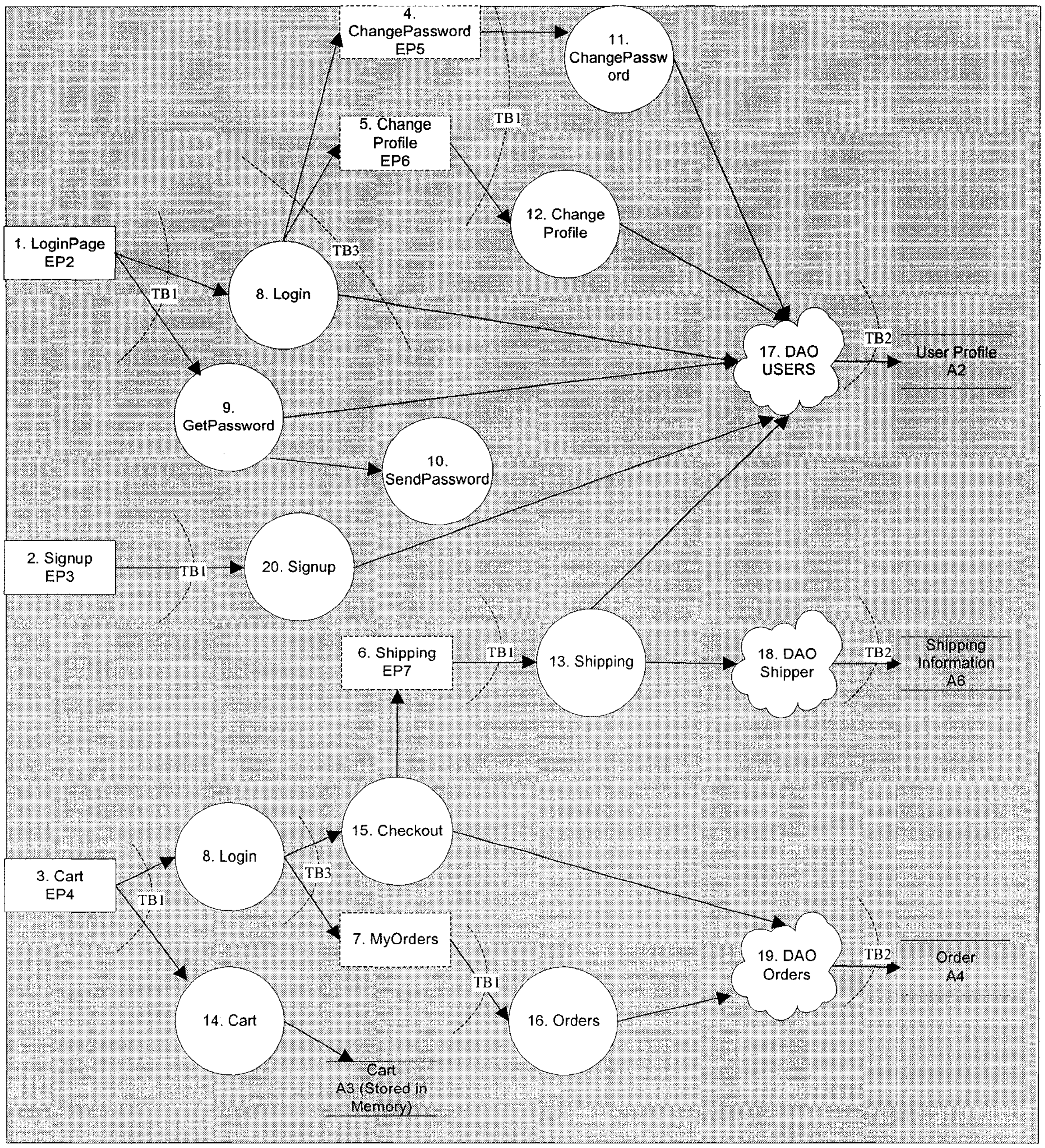
Attack Path

In VMCP, an attack path is defined as a logical path that connects entry points and assets, through which data or controls flow from entry points to assets. In other words, an attack path represents the computational logic within a component that connects discrete entry points to assets.

When performing attacks, hackers feed malevolent data or controls to the system being attacked and try to cheat the system to treat the malevolent data or controls as trusted data or controls. Then these malevolent data or controls are able to flow within the system along paths that trusted data or controls usually take and disasters occurs when these bad data and control are processed by the system. Before attackers begin to attack a system, they must know that there are something valuable (assets) and there are places (entry points) where they can access the system. But there is still something uncertain to them. That is whether or not the malevolent data or controls they input can reach the assets and bring back whatever they expect. Therefore most of their "attacking work" is to figure out whether or not there are logical paths within the systems that their malevolent data or controls can take to reach the assets. These paths are attack paths. Attack paths can be presented in form of data flow diagram or other flow diagrams like process or work flow diagrams. VMCP enhances traditional data flow diagram to present the computational logic of a component by adding assets, entry points and trust boundaries to the data flow diagram. We will detail enhanced data flow diagram in next section. Figure 7-1-1 is the enhanced data flow diagram of JSPCart, an online shopping cart. In this diagram, rectangles with label "EP#" represent entry points, pairs of horizontal parallel lines with label "A#" represent assets, and arc lines with label "TB#" represents trust boundaries.

For example, a hacker tries to attack Order data of the JSPCart. He can start the attack from any of the entry points such as the Login Page, Signup Page, or Cart Page. However the malevolent data or controls he inputs via the Login Page and Signup Page will never reach the Order data because there is no logic path connecting Login Page and Signup Page with the Order data. Thus attacks originate from these two points will never hurt the Order data. However attacks initialized from the Cart Page are likely to be successful as it is possible for the malevolent data or control to reach the Order data through paths 3→8→15→19 or 3→8→7→16→19 if the hacker can cheat the JSPCart to trust the data or controls fed by him. These two paths are attack paths.

Figure 7-1-1 Enhanced DFD of JSPCart



Security Criteria

Security Criteria are rules associated with an attack path, VMCP considers security criteria of an attack path from two aspects, authentication and authorization.

- **Authentication:** prescribes who are permitted to use an attack path, or whose data can pass through an attack path. Any unauthenticated passenger is forbidden to use the attack path for any purpose. In the JSPCart example, the legal passenger of attack path 3→8→15→19 or 3→8→7→16→19 is an authenticated user.

- **Authorization:** prescribes what kind of actions a legal passenger can perform via a certain attack path. The passenger can only perform the actions that he is authorized to do. In the JSPCart example, an authenticated user is authorized to create, update, retrieve or delete the Order data of this own, but not to touch the Order data of others.

When a hacker attacks a system via a chosen attack path, what he needs to do is to break the security criteria by either cheating the system that he is the right person to use the attack path or cheating the system that he is doing what he is allowed to do. Contrarily, the security level of a system depends on how the system defends these kinds of tactics.

Why Attack path and Security Criteria

Basically VMCP is a procedure based on white box technologies. It detects architectural and implementation vulnerabilities by reviewing design documentations and performing code review. First VMCP tries to figure out the potential attacks based on the component's architecture and design and then VMCP tries to see how the component defends these attacks by reviewing the detailed design and code. Vulnerabilities are expected to be discovered during this procedure. This procedure would be very time- and cost-consumed if we spend too much effort on the places where attacks are not possible. By using attack paths and Security Criteria, we can greatly:

- **Reduce the effort to find out potential attacks.** Actually attack paths are where the potential attacks will occur. By modeling all attack paths VMCP only focus on the right place to find out potential attacks.

- **Reduce the effort to review how a system defends potential attacks.** When reviewing the defending mechanism, VMCP knows where to check, that is along attack paths, and what to check, that is how the system implements the security criteria. This will greatly reduce the time and cost.

7.2 Enhanced Data Flow Diagram

Data flow diagrams (DFDs) provide a logical depiction of the implementation of a system and show the large-scale architecture of the system. It shows what entities exist in the implementation of the system, and along what paths these entities exchange information. VMCP uses DFDs to gain better understanding of the operations of a component. They provide a visual representation of how the component processes data and control. This representation allows the component to be modeled based on transformations and processes applied to data and controls an adversary might supply.

The data flow approach in VMCP follows the adversary's data and controls as they are processed by the system, analyzing how they are parsed and acted upon, as well as noting which assets they interact with. Because an adversary can attack only the parts of an application that they can exercise in this manner, the data flow approach provides an ideal way to show where the application could be susceptible to security failures. To let the DFD serve VMCP better, we enhanced the traditional DFD in the follow aspects:

- **Replace External Entity with Entry Point.** In a traditional DFD, the external entity shape represents an inter-actor that exists outside the system being modeled. It shows WHO interacts with the system. However, rather than WHO, VMCP is more interested in WHERE. Entry points have all information of where an external entity can interact with the systems.

- **Replace Data Store with Asset.** In traditional DFD, the data store shape represents a repository for data—such as the registry, file system, or database—WHERE data is saved or retrieved. Contrary to the external entity, here VMCP is more interested in WHO rather than WHERE. Assets represent particular data that is interested to attackers, e.g. Order, Transaction Records and so on.

- **Add Trust Boundary into the diagram.** VMCP extends traditional DFDs to include trust boundaries, which can help us identify attacks. Trust boundaries separate two processing nodes (or a processing node and an entity point or asset) that have different trust levels associated with them, or nodes that perform actions with different trust levels.

The enhanced data flow diagram (EDFD) provides a systematic approach to identify potential attacks. Rather than simply brainstorming attacks, the certification team is able to follow data and controls through the system along attack paths to identify potential attacks. Any transformation or action on behalf of the data could be susceptible to attacks. Thus, the team members are better able to enumerate attacks because at any point they are dealing with a specific processing action on specific data from a specific entry point.

7.3 How to model attack paths and Security Criteria

Enhanced data flow diagrams (EDFDs) are the preferred method of diagramming components in VMCP, because they show processes that occur based upon data input. An adversary cannot attack a software system without supplying it with data. Therefore, failures caused by attacks to a system can occur only at these places where the system transforms, takes action based on, or otherwise processes data that is ultimately supplied by the adversary. These places are process nodes in a data flow diagram. An attack path consists of a sequence of process nodes that process data flowing from entry points to assets—that is, any of the aforementioned nodes that process data or controls from a potentially malicious source.

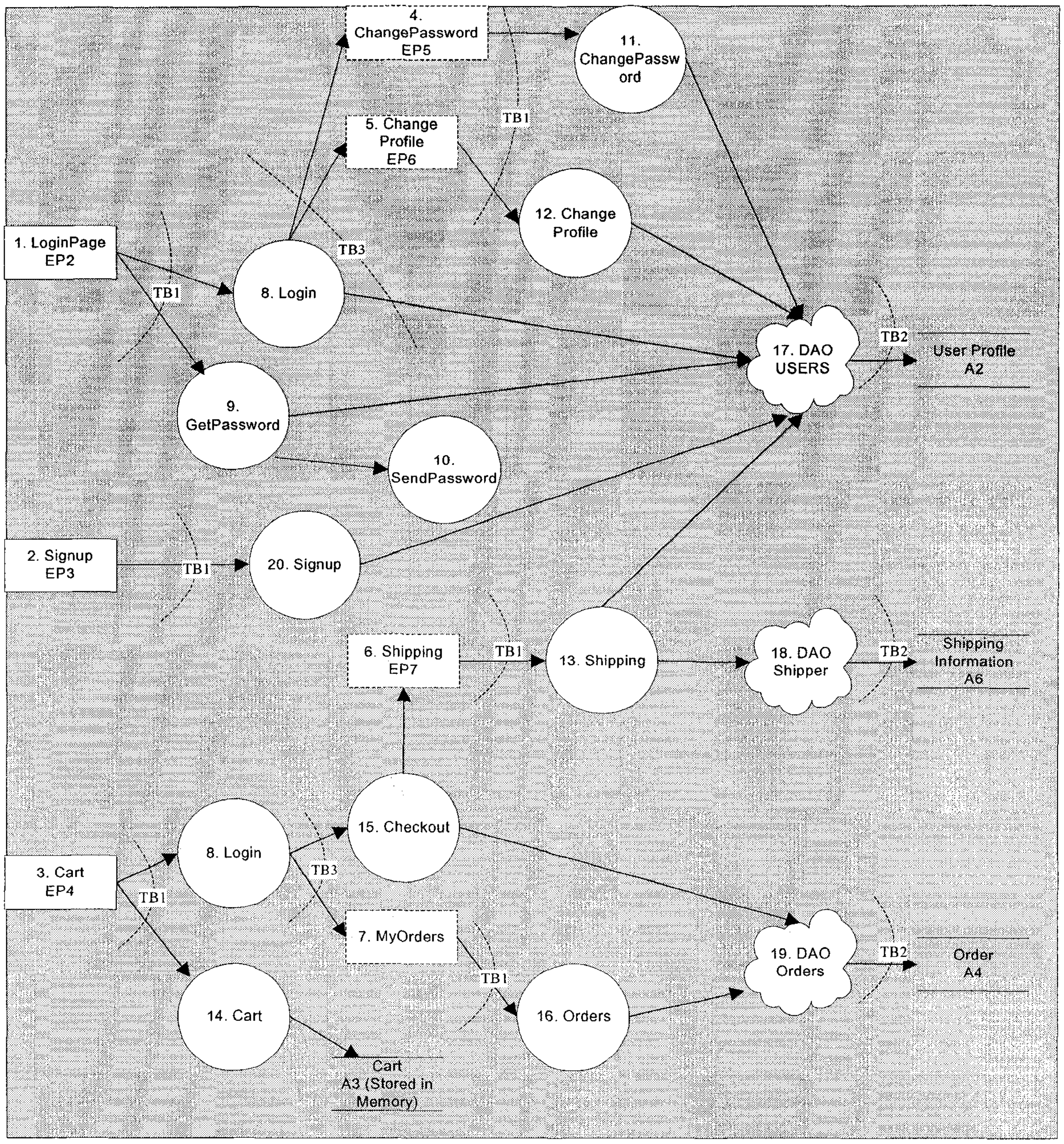
Modeling attack paths is actually modeling EDFDs because an attack path is a particular path in EDFDs, a series of process nodes starting from an entry point and ending at an asset. When we model EDFDs, it is not necessary to draw out all branches. We only enumerate the connections between entry points and assets. VMCP follows below steps to model EDFDs, a collections of attack paths.

1. Uses all identified entry points as start points and then extends them to anywhere the data or controls that enter the component by these entry points can go. By this way we can draw out all paths interacting with these entry points and find out all possible assets the attacker can reach through the entry points. We may also find out some new assets that have not been identified before.

2. Use all identified assets as end points and figure out all paths through which data or controls flow into these assets. Then go backward along these branches until the starting points of the data or controls. By this way we can get all possible paths leading to the assets and all entry points connected to paths. We may also find out some new assets that have not been identified before.

3. Integrate identified trust boundary into the flow chart. After step1 and step 2 we must have found out all paths connecting entry points and assets. These are the attack paths. Add identified trust boundary into the flow chart to indicate the most possible place a successful attack may occurs. Figure 7-3-1 is the EDFD of an online shopping cart, JSPCart.

Figure 7-3-1 EDFD of JSPCart



When modeling attack paths we must gather the following information. Figure 7-3-1 is the EDFD of appendix A: sample I Jspcart and table 7-3-1 is the documented attack paths.

- **ID** A unique number assigned to the attack path, which will be used to cross-reference attack paths with attacks and vulnerabilities later in the modeling process.
- **Description** A brief description of the attack path. Write down all nodes of the paths and briefly describe what data is passed on the path and what processes it interacts with.

- **Passenger** The role/roles who create the data or control passing through the path. For instance, the username and password passed on the path P1 of table 7-3-1 are created by an anonymous user, so the passenger of this path is anonymous user.
- **Action on Asset** A pair of asset and actions, indicating what assets interact with the path and what kind of actions can be done on a certain asset. VMCP decomposes any (virtual) action into one of "Create", "Read", "Update", and "Delete" (CRUD), or a compound action based on one or more of these actions. Action on Asset is presented in form of Asset (Action) pair, e.g. Anonymous User(R) and Authenticated User (CRUD).
- **Rule** The security criteria of each action. Rules define the circumstances within which an action can occur. The rules for an action are a set of declarative sentence fragments, connected by logical connectives (and, or, and not). Actually the *Passenger* and *Action on Asset* are also the rules. We list them separately because they are the most basic rules that almost all actions will have. Rules other than these two are expected here, like the frequency that actions can be taken or when they may or must occur, what portions of an asset can be affected by an action

Table 7-3-1 Attack Paths of JSPCart

Path ID	P1
Description	1→8→17 Anonymous user inputs username and password at webpage. Login model compares the credential passed by login pager with that got from data access object (DAO) users.
Passenger	Anonymous user
Action on Asset	User profile (R)
Rules	Action is only allowed to the profile of user's own.

Path ID	P2
Description	1→8→4→11→17 1. Authenticated users input new password on ChangePasswrod page 2. ChangePasswrod page passes the new password to server side

	model ChangePassword. 3. ChangePassword pass it to DAO users. 4. DAO users update the user profile.
Passenger	Authenticated user
Action on Asset	User profile (U)
Rules	Action is only allowed to the profile of user's own.

Path ID	P3
Description	1→8→5→12→17 1. Authenticated users change profile on ChangeProfile page 2. ChangeProfile page passes the changed profile to server side model ChangeProfile. 3. ChangeProfile pass it to DAO users. 4. DAO users update the user profile.
Passenger	Authenticated user
Action on Asset	User profile (U)
Rules	Action is only allowed to the profile of user's own.

Path ID	P4
Description	1→9→17 Anonymous user input username. Getpassword model retrieves the password using the username through DAO users.
Passenger	Anonymous user

Action on Asset	User profile (R)
Rules	Action is only allowed to the profile of user's own.

Path ID	P5
Description	2→20→17 Anonymous user input signup information. Signup model create the profile through DAO users.
Passenger	Anonymous user
Action on Asset	User profile (C)
Rules	NA

Path ID	P6
Description	3→8→15→6→13→17 Authenticated user check out the items in the shopping cart, and input shopping information. Shopping model retrieves some basic personal information from user profile through DAO users
Passenger	Authenticated user
Action on Asset	User profile (R)
Rules	Users can only retrieve their own profile.

Path ID	P7
Description	3→8→15→6→13→18 Authenticated user check out the items in the shopping cart, and

	input shopping information. Shopping model stores shopping information to database though DAO Shipper.
Passenger	Authenticated user
Action	Shipping information (CRU)
Rules	Users can only check out the cart created by themselves

Path ID	P8
Description	3→8→15→19 Authenticated user checkout the shopping cart. Checkout model creates an order through DAO orders.
Passenger	Authenticated user
Action on Asset	Orders(C)
Rules	Users can only check out the cart created by themselves

Path ID	P9
Description	3→8→7→16→19 Authenticated user updates orders.
Passenger	Authenticated user
Action on Asset	Orders (CRUD)
Rules	User can only update orders under their identifier.

8 Identify Attacks

8.1 Identify attacks by Outside-In and Inside-Out

Identifying attacks is often the most difficult part of any vulnerability modeling process. However, --in VMCP it can be straightforward as all of the required information is at hand: the pre-defined security attacks and vulnerabilities taxonomy, the architectural overview of the component from the security perspective, collected entry points and assets, and the enhanced data flow diagrams (EDFDs). VMCP uses a combination of tow complementary approaches, outside-in and inside-out, to perform attacks analysis.

- **Outside-In**

  The Outside-In approach begins with a set of potential, well-known and pre-defined attacks, and matches them to the details of the situation. With this approach, we consult a list of attacks and determine whether they apply to a certain environment. The list VMCP uses is the Security attacks & vulnerabilities taxonomy defined in chapter 4, which focuses on the most common application level attacks and vulnerabilities that plague web-based enterprise applications.

  Outside-In is a general and easy approach, however it does have its limitations. First, every component is unique. Although two components might share some features, a certain amount of these features will be system specific. Thus portions of their potential attacks will differ. Working backward from well-known attacks typically yields only common attacks, and system-specific attacks require a deeper analysis of the unique qualities of the component being modeled. Second, when we determine whether a common attack applies to a certain component, other issues will occur. Where to check? How much to check? Do we have to check anywhere in the component? Definitely we need a guideline pointing out where are the possible and necessary places to check. The Inside-Out approach resolves these questions.

- **Inside-Out**

  Inside-Out begins with the details about the situation and identifies attacks associated with them. With this approach, we study a component and repeatedly ask ourselves what might go wrong here? More particularly, for each part of the component, we need to consider what weakness or possible failures exists in the component, could there be any inputs or situations that might exploit a vulnerability and trigger a failure in this component, and who or what would be impacted by potential attacks and how bad would the damage be.

  The Inside-out approach is a direct form of attack analysis. It requires substantial technical insight and expert experiences. If we do not have expert knowledge of attacks and how these attacks exploit systems, we won't find anything even if we check the component multiple times. This also reveals the biggest disadvantage of this approach, depending too much on personal experience and knowledge.

Approach used by VMCP

VMCP combines these two approaches when identifying attacks, enhancing the advantages and counteracting the disadvantages. The overwhelming advantages of the combination are:

- VMCP not only has a list of common attacks to check against, but also specifies a collection of places to check for these attacks and component-specific attacks. In chapter 7, we defined the attack paths and security criteria. The security criteria are component-specific, so component-specific attacks can be identified when we check if attackers can and how they break these security criteria. Attack paths in VMCP are places where potential attacks will occur. When identifying attacks, VMCP checks and only checks attacks along these attack paths. Other places outside the attack paths are not necessary to check, because they are not security-related.
- The security attacks & vulnerabilities taxonomy defined by VMCP focuses on common application level attacks and vulnerabilities that plague web-based enterprise applications. This taxonomy not only lists well-known attacks but also lists the reasons (causing vulnerabilities) to each listed attack. If we used it to perform inside-out attack analysis, the dependencies on personal experience and knowledge will be reduce greatly.

8.2 Steps to identify attacks in VMCP

VMCP assumes that to attack an application what the attackers have to and can only do is taking over one of the attack paths and breaking the security criteria. With the documented attack paths and security criteria tied to them, identifying attacks turns into checking each of these paths to see how to break the security criteria. Actually some of the paths are very similar in nature, for example in table 7-3-1 the attack paths for JSPCart, paths P2 and P3 both get user input via Webpager and update the user profile. From the perspective of adversaries, they are the same and can be broken the same way. So when identifying attacks, we do not need to analyze every attack path. We can group some similar paths together according to certain criteria and pick one or several typical paths to do a common attack analysis. As for each individual path in the group, we only pay attention to the particular security requirements listed in the Rules line of the Attack Path table. To perform attack analysis, VMCP uses the following three steps:

1. **Group attack paths**
2. **Choose attack paths to be analyzed from a group**
3. **Identify Possible Attacks along chosen attack paths**

Group attack paths

Attack paths with the same category of entry point and asset can be grouped together, as they are similar in term of attack analysis. VMPC classifies entry points into four categories: User interface, Network interface, System interface and Component interface. Each of these categories are tied with some exclusive common attacks, for example, buffer overflows, cross-site scripting and format string attacks are user interface related attacks while network eavesdropping occurs only when an application has a network interface. The same situation applies to assets also. VMPC classifies assets into 3 categories: application data, non-functionality, and system resource. Command insertion is a very common attack to system resource however it does nothing to application data stored in databases, which is suffering from SQL injection attacks. Therefore, it is possible and efficient to perform a common analysis on a group of attack paths with entry points and assets from the same category.

Take Table 7-3-1 as an example. Consider the paths P1, P3, P6 and P8. The entry point of P1 and P3 is EP2 (Login Page) and that of P6 and P8 is EP4 (Cart Page). From Table 8-2-1 below we know EP2 and EP4 both belong to the user interface category. The asset of P1, P3, and P6 is User Profile and that of P8 is Order. Both the User Profile and the Order belong to the category of application data. As entry points of these four paths belong to the same category and assets of them also belong to the same category, we group these 4 paths together.

Table 8-2-1: Entry Point Table of Jspcart

ID	Name	Description	Trust Roles	Category
EP1	Web server listening port	The port on which the Web server listens. All web pages are layered on this entry point.	All	Network Interface
EP2	Login Page	Get user credentials and passed them to server side for authentication	All	User Interface
EP3	Signup Page	Create a user profile and pass it back to server side	All	User Interface
EP4	Cart Page	Users modify quantity of items and remove items from cart.	All	User Interface
EP5	ChangePassword page	Authenticated users change password	Authenticated user	User Interface
EP6	ChangeProfile page	Authenticated users change profile	Authenticated user	User Interface
EP7	Shipping page	Authenticated users input shipping information	Authenticated user	User Interface
EP8	MyOrder Page	Authenticated users enquiry order details and/or cancel orders	Authenticated user	User Interface

Table 8-2-2: Asset Table of Jspcart

ID	Name	Description	Trust Roles	Category
A1	User credentials	Username and password	Authenticated user (CRUD)	Application data
A2	User profile	User profiles stored in back-end database	Authenticated user (CRUD)	Application data
A3	Cart	Cart information stored in back-end database, e.g. the name and quantity of items in a cart.	Authenticated user (CRUD) Anonymous user (CRUD)	Application data
A4	Order	Orders stored in back-end database	Authenticated user (CRUD)	Application data
A5	Product	Products stored in back-end database	Anonymous user (R) Authenticated user (R) Administrator (CRUD)	Application data
A6	Shopping information	Shopping information for orders.	Authenticated user (CRU) Administrator	Application data

			(CRUD)	
A6	Catalog	Catalogs stored in back-end database	Administrator (CRUD)	Application data
A7	Process	Processes running within the same machine where the component is running	N/A	System resource
A8	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource
A9	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A10	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

C: Create, **R**: Retrieve, **U**: Update, **D**: Delete.

Choose attack paths to be analyzed from a group

When identifying potential attacks on a group of attack paths we only need to pick up one or several paths as the analysis target. How the target is chosen significantly affects the completion and efficiency of the analysis. We must choose these that represent the group the most and the analysis result covers the most attacks applying to this group. VMCP uses the following criteria:

Choose a collection of paths that has the least number of paths and

1. *has at least one path whose passengers are in the highest privileged user group*
2. *has at least one path whose passengers have the lowest access privilege to the category of asset*
3. *covers all trust boundaries in the group.*

Criterion 1 selects paths where most attacks regarding authentication occur. As the passenger with the highest privilege, the authentication mechanism of this path should have the strictest constrains and attacks on this path will cause the most valuable loss. Therefore attacks identified on this path will possibly cover all potential attacks regarding authentication. Criterion 2 selects paths where most attacks regarding authorization occur. To get control of an asset, a hacker has two options:

- one is to get a credential that is powerful enough to control the asset; and
- the other is to get enough authorization for a normal credential that is

very easy to obtain such as a anonymous user.

Criterion 1 covers option 1 and criterion 2 is for option 2. Paths whose passengers have the lowest access privilege must be places where hackers perform most attacks regarding authorization. Criterion 3 is supposed to include all places where attacks are most likely to occur.

Let us proceed with the previous example. In the group of paths P1, P3, P6 and P8, the passenger of P1 is Anonymous User and that of P3, P6, and P8 is Authenticated User. So the final collection must include at least one of P3, P6, and P8. Then it comes to the second criterion, P1 and P6 only have Read access to User Profile. P3 has Update access to User Profile. P8 has Create access to Order. According to the criterion 2, we will pick up at least one of P1 and P6. As for trust boundaries this group has three trust boundaries TB1, TB2, and TB3 and any of the four paths covers all of them. Therefore, there are many sets of paths that meet the criteria 1 to 3, say {P3, P6, P8}, {P6, P8}, {6} or even the full set {P1, P3, P6, P8}. What we need is the one with least number of members. That is {6}.

What is beneath the scene is that VCMP tries to choose a collection of attack paths along which we can discover most of the potential attacks and then vulnerabilities. Analyzing P6 can disclose more attacks than analyzing P1 as the passenger of P1 is Anonymous User and anybody can take over the path, while you have to break the authentication before take over the P6. That is why there is criterion 1. After you have broken the authentication of P6 all that you can do is retrieving information from the User Profile. If you want to change something in the User Profile extra work is needed. P3 is another story; once you have broken this authentication you can update the User Profile without any extra effort. This is the rationale of criterion 2. The trust boundary is where the trust level changes and the elevation of privilege occurs. Almost all attacks are done through trust boundaries; so the final collection must cover all trust boundaries.

Identify Possible Attacks along chosen attack paths

From the attack path table, we can see security criteria consist of three sections: passenger, action on asset and rules. We can identify possible attacks from these three aspects:

- Can and how hackers break authentication and acting as the expected passenger?
- Can and how hackers elevate privilege and do what is not allowed?
- Can and how hackers break rules, particular security requirement?

Basically most attacks fall into one of two categories, either authentication or elevation of privilege. An authentication attack occurs when a passenger breaks the authentication mechanism and does what are allowed by the application to another passenger. An elevation of privilege attack occurs in one of three situations:

- when an passenger performs an action which no passenger is intended to perform on an asset (an entirely disallowed action); or
- when an passenger performs an action on an asset despite the rules for that action (specifically disallowed action); or
- when an passenger uses the component to perform an action on some other component's asset.

To identify attacks, VMCP starts from the common attack patterns presented in chapter 4 Security Attacks & Vulnerabilities Taxonomy, and then examines the attack paths tier by tier, layer by layer, and process by process. By focusing on attack and vulnerability categories, we focus on the areas where security mistakes are most frequently made. The attacks identified at this stage do not necessarily indicate vulnerabilities. Vulnerabilities are identified when VMCP tries to verify low the component defends all these identified attacks. Next chapter will details this.

8.3 Classify and document attacks

The high-level classification of attacks helps verification team understand what an attack allows a hacker to do. Furthermore, it helps verification team assign priority to attacks. In most applications, elevation of privilege attacks carry the most risk because they allow an attacker to perform normally restricted functionality. However, in many other applications, repudiation is also critical (such as in financial applications where failure to properly audit actions could have legal and monetary implications). VMCP uses the STRIDE model to classify identified attacks. STRIDE is a classification of the effects of realizing an attack: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege.

Spoofing
Spoofing allows an adversary to cheat a system that he is another user who has an identity in the system. Applications that have many users but use a single execution context at the application and database level are fragile to spoofing attacks. Applications must implement a strong mechanism to prevent users from acting as any other user, or becoming that user.

Tampering
Tampering refers to the malicious modification of data within the system. Equipped with proper tools, hackers are able to change any data delivered to client/browser such as client-side validation, GET and POST data, cookies, HTTP headers, and so on. Sensitive data such as account information, which are available only within the application itself should not be sent to the user. Applications should never trust any data received from the user without careful validation.

Repudiation
Repudiation refers to an adversary denying performing some malicious activity because the application does not have sufficient evidence to prove otherwise. Sufficient traceability and auditing of user activity have to be implemented to keep users from disputing transactions. For example, if a user says, "I didn't transfer money to this external account", and you cannot track their activities from front to back of the application, it is extremely likely that the transaction will have to be written off.

Information Disclosure
Information disclosure refers to the exposure of sensitive data to a user that does not have access to that data. For example, the user's browser can leak information if the browser does not correctly implement the no-caching policies requested by the HTTP headers. A secure application should minimize the amount of information stored by a browser because it might leak information and be used by an attacker to learn more about the user or even become that user.

Denial of Service
Denial of service refers to attempts to make an application unavailable to its intended users. It is among the most common attacks to applications regarding availability that is all about keeping systems available for legitimate users in the certain period. Denial of service crashes an application or ensures that it is sufficiently overwhelmed so that other users cannot access the application. Not all applications are aware that they could be abused by a denial of service attacks and open expensive resources such as large files, complex calculations, heavy-duty searches, or long queries to anonymous users.

Elevation of Privilege
Elevation of privilege refers to a process or an attack by which a malicious user tries to become a member of the group with higher privilege than the group for which they have been authorized. This kind of attack could enable hackers to compromise or destroy a system, or to access unauthorized information. For example, if an application provides user and administration roles, it is vital to ensure that the user cannot elevate themselves to any higher privilege roles. Applications sometimes fail to gate actions through an authorization matrix to ensure that only the right roles can access privileged functionality.

Attacks often fit into multiple categories of the STRIDE model. Certification teams need to understand the underlying effect an attack has on the system when classifying threats with the STRIDE model. For example, some attacks are pure spoofing in nature; others might enable tampering but arise because of the elevation of privilege. Certifiers must ensure they understand the root effect of an attack; otherwise they might not classify the attack correctly.

When compiling an application's vulnerability model, the modeling team must gather the following information about the attacks they want to verify the component against:

- **Numerical ID** A unique number assigned to the attack for reference.
- **Name** A short title for the attack, which should be descriptive enough to identify the attack as well as the target asset. An example of this is, "The adversary views another user's personal information."
- **Description** The description should provide additional details about the nature of the attack.
- **STRIDE classification** The STRIDE model is used to help understand the effect of realizing a specific threat.

Examples

This section contains identified attacks from appendix C: Sample III - the Credit Card Payment Component.

ID	A1	Name	Buffer overflow
Description	Buffer overflow and Integer overflow.		
STRIDE	T. D. E		

ID	A2	Name	SQL injection
Description	SQL injection occurs, enabling an attacker to exploit an input validation vulnerability to execute commands in the database and thereby access and/or modify data.		
STRIDE	T. I. E		

ID	A3	Name	Cross-Site Scripting
Description	Cross-site scripting occurs when an attacker succeeds in injecting script code.		
STRIDE	T. I. D. E		

ID	A4	Name	Information Disclosure
Description	Information is disclosed and sensitive exception details are revealed to the client.		
STRIDE	I		

ID	A5	Name	Server Attack
Description	An attacker manages to take control of the servers, gain unauthorized access to the database, and run commands against the database.		
STRIDE	T. D. E		

ID	A6	Name	Configuration Attack

Description	Retrieval of clear text configuration secrets
STRIDE	T. D

ID	A7	Name	Individual Accountability
Description	Lack of individual accountability		
STRIDE	R		

ID	A8	Name	Session Attack
Description	Session hijacking and replaying.		
STRIDE	S. T. E		

ID	A9	Name	Form Manipulation
Description	Query string and form field manipulation		
STRIDE	T. E		

9 Identify Vulnerabilities

Given an attack path and a set of potential attacks, we can start to look at how those attacks may be realized. An attack is a series of threat-specific, implementation-specific, or technology-specific steps that a malicious user takes to realize or help to realize exploiting a system. Through thoroughly analyzing how an identified attack may be realized and how the application defends itself, we can evaluate the application's vulnerabilities. VMCP adopts attack tree to analyze attacks and then identify vulnerabilities.

9.1 Analyze attacks using attack tree

Attack tree and Attack tree analysis

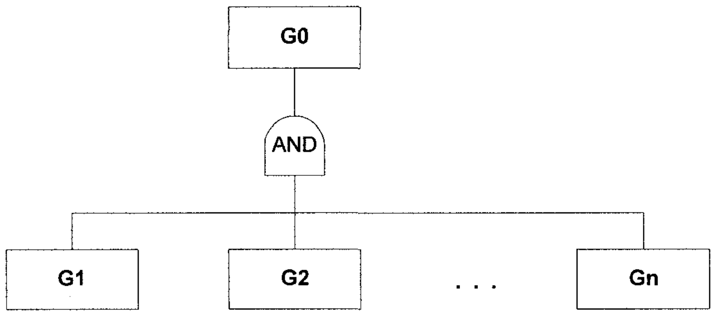
Attack trees were defined by Bruce Schneier [68] based on the earlier work by Nancy Leveson [69]. An attack tree is a way of analyzing and documenting the potential attacks on a system in a structured and hierarchical manner, which describes how an attacker could realize a specific attack to the system. A traditional attack tree is made up of tasks and subtasks. An attack is represented as the root node of each tree and tasks that need to be done to realize the attack are represented as nodes other than root node. Children of each node describe in increasing detail how an attacker could accomplish the task in the parent node. Children of each node are sub-goals needed for the node, and together, all the children should specify every way that this node could occur. In addition to goal and condition type nodes, attack graphs can contain logical connectives. Some nodes may require all of their children to be accomplished in order to be accomplished themselves, while others may require only a single node to be accomplished. A node of an attack tree is decomposed either as

- a set of attack sub-goals, all of which must be achieved for the attack to succeed, that are represented as an AND-decomposition, or

- a set of attack sub-goals, any one of which must be achieved for the attack to succeed, that are represented as an OR-decomposition.

Attack trees can be represented graphically or textually. An AND-decomposition is represented as follows:

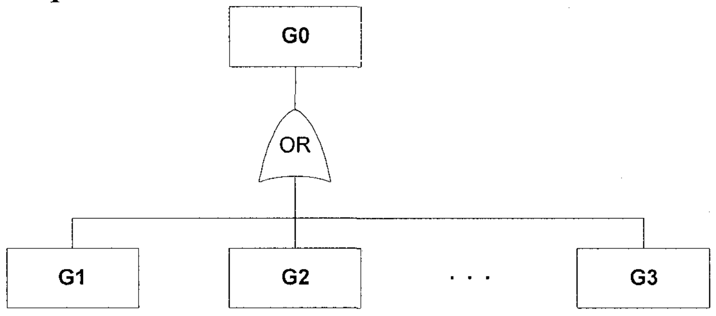
Graphical:



Textual: **Goal** G0
AND G1
G2
. . .
Gn

This represents a goal G0 that can be achieved if the attacker achieves each of G1 through Gn.

An OR- decomposition is represented similarly:
Graphical:



Textual: **Goal** G0
AND G1
G2
. . .
Gn

This represents a goal G0 that can be achieved if the attacker achieves any one of G1 through Gn.

Attack trees can consist of any combination of AND- and OR-decompositions. Individual attack scenarios can be generated from an attack tree by traversing the tree in a depth-first manner. In general, leaf tasks are added onto the end of scenarios as they are generated. OR-decompositions generate new scenarios. AND-decompositions extend existing scenarios. Intermediate nodes of an attack tree do not appear in the attack scenarios because they are elaborated by lower level sub-nodes. For example, attack tree A of figure 9-1-1 generates the attack scenarios {G3, G5, G6} and {G4, G5, G6}, and attack tree B of figure 9-1-2 generates the attack scenarios {G4}, {G8, G9}, {G2} and {G6, G7}.

Figure 9-1-1: Attack Tree A

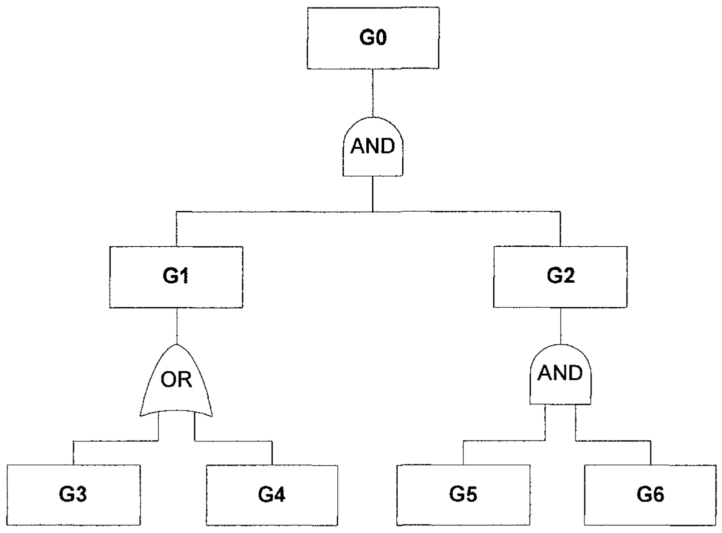
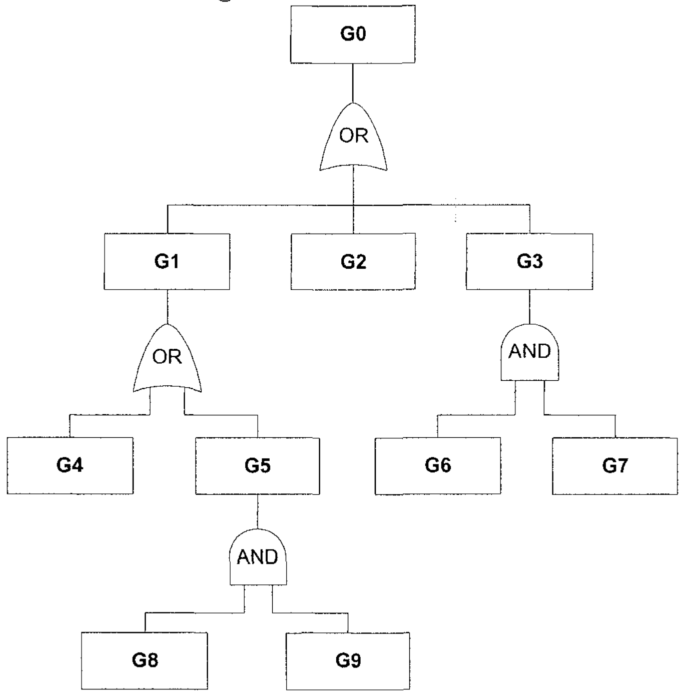


Figure 9-1-2: Attack Tree B



Attack trees allow the refinement of attacks to a level of detail chosen by the analyst. Prowell [70] addressed the property of referential transparency of an attack tree as:

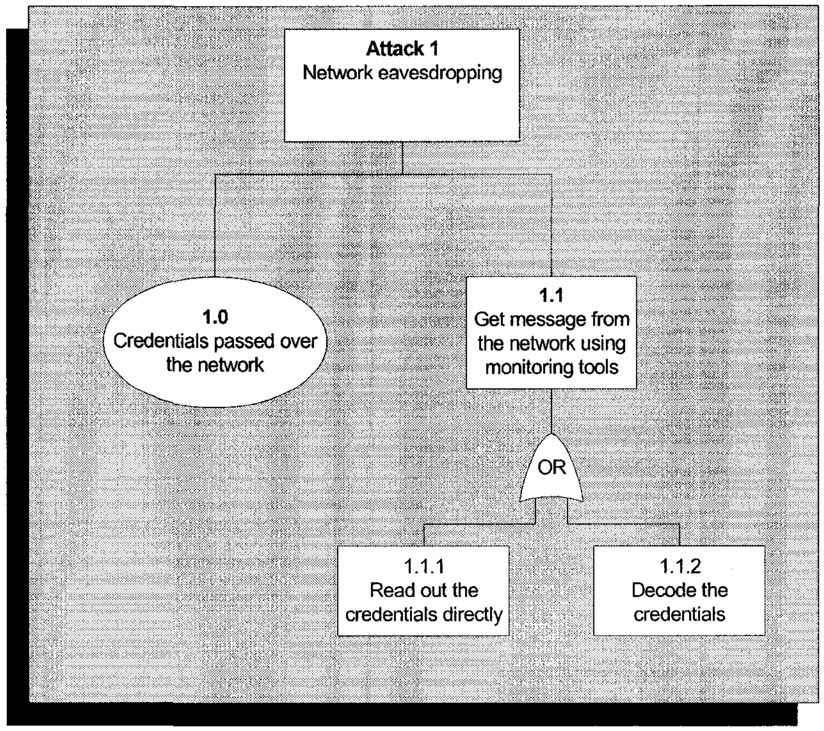
> *"Referential transparency implies that the relevant lower level details of an entity are abstracted rather than omitted in a particular system of higher level description, so that the higher level description contains everything needed to understand the entity when placed in a larger context"*

Analyst may take advantage of this property to explore certain attack paths in more depth than others, while still generate attack scenarios that make sense. In addition, refining the branches of the attack tree generates new branches, resulting in attack scenarios at the new lower level of detail.

Enhancement made by VMCP

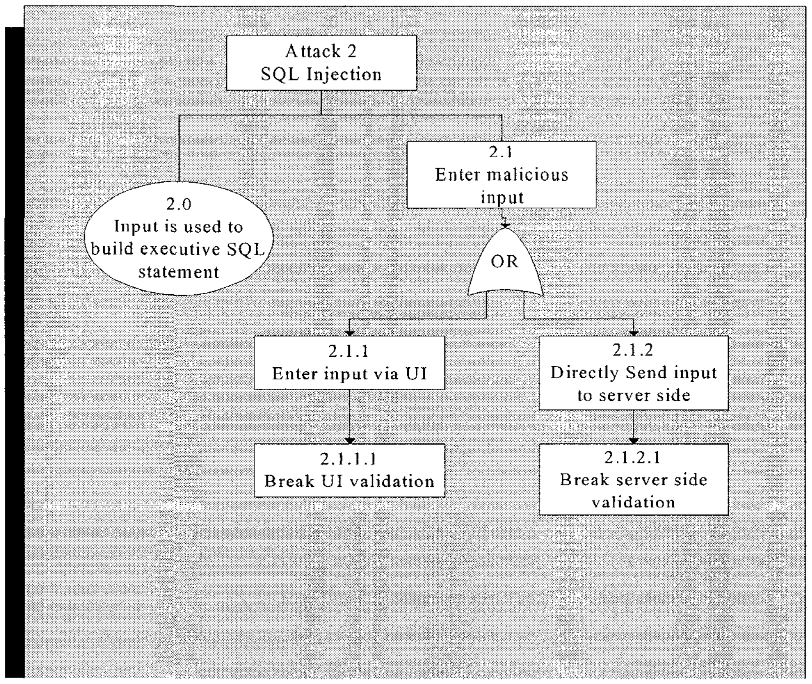
VMCP identifies vulnerabilities by checking how components defend potential attacks identified in previous steps. To make attack analysis more efficient VMPC introduces pre-condition branches into attack trees. Pre-condition is the specific situation or environment without which the attack is not possible to realize. The pre-condition is presented using an ellipse while task and subtask using a rectangle. Figure 9-1-3 is an attack tree for obtaining authentication credentials over the network. The pre-condition for this tree is that the credentials are passed over the network. If the credentials are not passed over the network, it is impossible for an attacker to obtain them over the network. Figure 9-1-4 is another sample attack tree – SQL Injection. Its pre-condition is that the user input is used to build the SQL executive statement. In VMPC, when it comes to performing an attack tree analysis, we first figure out if the pre-condition is met based on the information obtained so far. If the pre-condition is not met, we proceed to the next attack. In this way, we can filter out some identified attacks which are impossible to the component in question at the very beginning, and focus on these really exist.

Figure 9-1-3 Attack tree –Network eavesdropping



Attack 1
Network eavesdropping
1.0
Credentials passed over the network
1.1
Get message from the network using monitoring tools
OR
1.1.1
Read out the credentials directly
1.1.2
Decode the credentials

Figure 9-1-4 Attack tree – SQL Injection



9.2 Identify vulnerabilities by verifying attack trees along attack paths

In terms of attack tree, a vulnerability is defined as a special path through an attack tree from one or more leaves to the root node. On this path no or insufficient defenses are implemented, wherein all conditions for the attack are met. This path specifies a weakness or a collection of weaknesses in the component that allow an attacker to implement an attack. Generally, all vulnerabilities fall into two types: architecture vulnerabilities and implementation vulnerabilities. VMCP seeks to identify both of these two types of vulnerabilities.

Identify architecture vulnerabilities

Architecture vulnerabilities are defined as inherent vulnerabilities in the design of the application. In other words, architecture vulnerabilities are problems that are designed into the application, and thus are hard to identify. These vulnerabilities cannot be identified by normal methods. For example code review is one of the most common methods that have become prevalent in V&V and is very useful in identifying coding problems, however even a thorough code review will not dig out architecture-level vulnerabilities. The reasons are obviously. Code review, conducted either by humans or by automatic scanners, focuses on a very limited context, reviewing the source code line by line. Moreover, a code review can only find out security problems that have been written into applications. However, architecture vulnerabilities are security problems that have been designed into applications. They are often caused by the technologies used and how the component will be deployed, rather than bad code.

VMCP uses attack trees to identify vulnerabilities. It investigates every node in an attack tree to see if and how the component mitigates the node by designs and implementation. This makes the application security analysis process less reliant on intuition and allows the process to more systematically enable people with less experience in security analysis to evaluate application's security strength.

When identifying architecture vulnerabilities, VMCP emphasizes on the mid-nodes, instead of leaves, and the pre-condition nodes of attack trees, verifying how the component mitigates them. We have discovered that mitigation of the mid-nodes (nodes of a tree except the leaves) and the pre-condition nodes are always related to the architecture and design, and mitigation of the leaves is done by implementation. Therefore, it is an efficient and feasible way to identify architecture vulnerabilities by verifying how the component mitigates the mid-node and the pre-condition nodes of attack trees.

On the SQL Injection attack tree (Figure 9-1-4 Attack tree – SQL Injection), there is one pre-condition node and 3 mid-nodes:

- **PC-node**: 2.0 Input is used to build executive SQL statements

- **Mid-node1**: 2.1 Enter malicious input
- **Mid-node2** 2.1.1 Enter input via UI
- **Mid-node3** 2.1.2 Directly send input to server side

To verify the PC-node, we need to check the design document and information collected in the previous steps. If the component does not use the input data to build executive SQL statements, there will not be any SQL injection attacks; else we have to check other mid-nodes. For sub-tasks described in the 3 mid-nodes there are two options to defense: one is suppressing any input, the other is using validation on both the client and server sides. Obviously, the option 1 is not possible for web applications. Thus, what we only need to do is to verify whether the component uses server side input data validate. If not, one of the architecture vulnerabilities will be "User input is used to build executive SQL statements and there is no input data validation on the server side".

Identify implementation vulnerabilities

Implementation vulnerabilities are defined as inherent vulnerabilities in the implementation of the application. In other words, implementation vulnerabilities are problems that are coded into the application. VMCP adopts code review as the verification method.

Although there are tools available that can automatically identify some kinds of implementation errors, they are currently very limited, both in number and functionality. These tools produce a lot of output much of which can be regarded as false positives. Moreover most of these tools are only useful in identifying unsafe code constructions, and not adept at identifying which instances of a construct will result in vulnerability and at finding security vulnerabilities that occur with different signatures. For example, that a piece of code that uses a string function that does not perform bounds checking can be easily identified. However, a similar buffer overflow that occurs because a function is looping through some section and copying to a fixed buffer with either user-specified or incorrect completion criteria is very hard to find out. Though both cause overflows, the latter type of buffer overflows still require human effort for identified. Even if tools are strong enough to identify these buffer overflows, they would not necessarily be able to determine whether or not the construction creates security vulnerabilities. Another problem with using automatic tools is that it is difficult for people to decide which potential vulnerabilities are worth further investigation, as these tools produce "mountains" of warnings with equal priority. Usually many of these constructions might be legitimate implementations.

Code review is a time-intensive process but VMCP creates a process that allows a team to check places (attack paths) of highest risk within a component for vulnerabilities. This allows the team to identify which sections of component would be best served by a code review. VMCP also allows a team to decide which bugs reported by code analysis tools are most likely to result in vulnerabilities and enables the team to reduce the volume of investigation required.

As mentioned in the last section, mitigation of the leaves of an attack tree is always related to implementation. VMCP identifies implementation vulnerabilities by verifying how the component mitigates the leaves of an attack tree. On the SQL Injection attack tree (Figure 9-1-4 Attack tree – SQL Injection), there are 2 leaves:

- 2.1.1.1 Break UI validation
- 2.1.2.1 Break server side validation

By reviewing corresponding code blocks on the attack paths, we can tell if the component is healthy enough to resist any SQL injection attacks.

9.3 Attack Pattern Library

Patterns exist everywhere today. When building attack trees we do not have to start from scratch each time, as certain patterns appear repeatedly and we can use them as sub-trees in our graphs. Creating attack trees is very time and effort consuming, but by using attack pattern libraries this becomes easier. An attack pattern library for VMCP includes two types of patterns, architecture and implementation patterns, corresponding to architecture and implementation vulnerabilities. Architecture patterns should be re-used in a concrete attack tree as high level nodes rather than leave node. Either an implementation pattern or system-specific nodes should extend all or some of leave nodes in an architecture pattern. Implementation patterns only appear in the lower levels of an attack tree. All the leave nodes of an implementation pattern should be the leave nodes of the attack tree that uses it as a sub-tree. These nodes can be used directly as part of the actual steps to attack a common well-known vulnerability.

Attack libraries are the most critical step to realize automatic attack analysis. With an attack library, the high layers of an attack tree or even the whole tree can be generated automatically by mapping between the kind of attack that is being analyzed and the technologies and design patterns used by relevant Data Flow Diagram elements. In many cases, it may be possible for the sub-trees from the attack library to simply be copied into the system-specific attack graph. In other cases, they may need to be customized for the specific system. Either way, attack libraries make the generation of in depth attack trees much more rapid and easier. Furthermore, the more certification an organization has completed, the better their library of attack patterns will become.

At this stage, I only introduce the concept of attack pattern library into VMCP methodology and have not done much concrete research on how to build and how to use an attack library, as this topic is wide and complex enough to be separate thesis. However, it is on the list of my further work.

10 Rate Vulnerabilities

To certify the security level of a given component, we need to know not only what vulnerabilities it has, but also the impact and risk that each vulnerability has on the customers if malicious users exploit it. VMCP uses a quantitative method DREAD to rate vulnerabilities.

DREAD

The DREAD method was first introduced by Michael Howard and David LeBlanc [67] to characterize the risk associated with a vulnerability. Later, Frank Swiderski and Window Snyder [5] used it to rate threats. In this thesis, I adapted it to rate vulnerabilities in VMCP by redefining the definition of each rating level, making it more practical to operate. The term *DREAD* stands for **D**amage potential, **R**eproducibility, **E**xploitability, **A**ffected users and **D**iscoverability. When using the DREAD method, a certification team calculates security risk/impact as an average of numeric values assigned to each of these five categories.

- **Damage potential** Measures the extent of the damage that occurs if a vulnerability is exploited.
- **Reproducibility** Measures how difficult it is to reproduce a successful exploitation of a vulnerability. A race condition attack is very hard to reproduce but a URL manipulation is easy to reproduce.
- **Exploitability** Measures how difficult it is to exploit a vulnerability. One of the most common methods to mitigate an attack is to increase the exploitability of vulnerabilities. If a user credential has to be transferred through network, then a strong cryptography may protect the data by increasing the exploitability.
- **Affected users** Measures how high is the ratio of installed instances of the system that would be affected if an exploit became widely available. Here we consider the ration but not the absolute amount.
- **Discoverability** Measures how high is the likelihood that a vulnerability will be discovered by external security researchers and hackers.

Microsoft uses continual numbers such 1 to 10 to score vulnerabilities [67] and threats [5]. They try to propose a completely quantitative method to measure and analyze vulnerabilities and threats. However these things are qualitative in nature and are impossible to be exactly measured by continual numbers. For example, when scoring metric R (Reproducibility), we usually estimate how difficult it is to reproduce a successful exploitation of a vulnerability, based on the personal experience and knowledge. The problem is how you can tell the difference when John scores a 5 to a vulnerability and Mike scores a 6 to the same one. Even if the estimates are done by the same person, it does not make any sense to differentiate a 5-point vulnerability from a 6-point one.

VMCP uses discrete numbers to score vulnerabilities, combining the advantages of both the quantitative method and the qualitative method. This not only takes into account the qualitative nature of vulnerabilities but makes it possible to perform quantitative analyst on them. VMCP uses only 3 numbers 5, 10, and 15 for each category, which means the risk is low, moderate, or high. Then the overall score of a vulnerability is determined by averaging the numbers (adding the numbers and dividing by 5, in other words).

Vulnerability Score = (DAMAGE + REPRODUCABILITY + EXPLOITABILITY + AFFECTED USERS + DISCOVERABILITY) / 5

Damage Potential
If a vulnerability is exploited how great can the damage be? Damage potential measures the extent of actual damage possible with the vulnerability. Typically, the worst (15) indicates that a vulnerability allows the attacker to circumvent all security restrictions and do virtually anything. For example, elevation of privilege vulnerabilities are usually scored 15. The following table is the definition of the three security level.

5	10	15
Leak trivial information	Leak sensitive information ; Compromise or affect individual user data.	The attacker can subvert the security system, get full trust authorization, run as administrator, or upload content.

To assess the potential damage of a vulnerability, VMCP provides a easy and straightforward way. At this stage, we already have a list of identified attacks and assets tied to attacks (chapter 9). Each identified vulnerability is derived from one of these attacks using attack tree analysis. Therefore it is easy to know which attacks will be realized by exploiting the vulnerability being assessed and then to know which assets will be impacted if the vulnerability is exploited. By assessing the value of all of these assets, we can score the potential damage. In some cases, a vulnerability might cause several attacks then we have to put together all the attacks and the assets to do the assessment. For example, no server side validation is the reason for many attacks such as SQL injection, buffer flow and so on. When assessing the damage of this vulnerability we have to think about all assets impact by all these attacks.

Reproducibility
Reproducibility looks at how difficult it is to reproduce an attack exploiting a given vulnerability. A complicated race condition which requires a very specific system state might be a five (5), if there was no straightforward way to produce that state, while an URL manipulation would be a fifteen (15). High

reproducibility is critical for most attackers to benefit. The following table is the definition of the three security level.

5	10	15
The attack is very difficult to reproduce, even with knowledge of the security hole.	The attack can be reproduced, but only with a timing window and a particular race situation.	The attack can be reproduced every time and does not require a timing window.

How a given vulnerability can be exploited is clearly described by the attack tree where the vulnerability derives from. With all attack trees we have it is quiet easy to determine the reproducibility of a given vulnerability.

Exploitability
Exploitability looks at how technically difficult to perform an exploit. The first thing to consider is what degree of authentication and authorization is required to perform an attack. For example, if an anonymous remote user can attack the system with just a browser, it should be assigned 15, while a local user attack requiring strong credentials is possibly a 5. In addition, expert experience and cost needed to perform an exploit are important factors too. For example, if a junior attacker with a home PC can start the exploit, that is a 15, but an exploit that needs an expert group working together and an expense of $100,000,000 is only a 5. In addition, if there are automatic tools existing is also to be considered.

When assessing the exploitability, it is critical to know the security trends, as they are changing all the time; for instance, an expert-experience-needed exploit could change from 15 to 5 if malware is developed and becomes widely used. It is worth being somewhat conservative here, especially when dealing with well-known software. The following table is the definition of the three security level.

5	10	15
advanced programming and networking skills, advanced or custom attack tools	Malware exists, or easily performed using normal attack tools	Just a browser

The same as reproducibility, exploitability can also be easily determined using attack trees.

Affected Users
Affected users looks at the number of users or the percentage of users that would be affected if the vulnerability were exploited. VMCP prefers percentage to absolute number. This measures approximately what percentage of users would be impacted by an attack: 71–100 percent (15), 31-70 percent (10) and 0-30 percent (5). An attack on a server indirectly affects a larger number of clients and

probably other networks, while a client attack impacts only a few people. Besides the percentage, VMCP takes into account the market size and absolute numbers of users. As you know, one percent of 100 million is still a very large number. The following table is the definition of the three security level.

5	10	15
0-30 percentage of users, obscure feature; affects anonymous users	31-70 percentage users, non-default configuration	71-100 percentage, default configuration, key customers

When scoring affected users, VMCP takes into account two groups of users, one is the group that are affected as assets tied to them are attacked, for example a user profile attacking affects all users. The other group is these who are affected because some functionalities or modules of the system become unavailable due to attacks. To find the number of users affected by assets, we will do the same as what we do when determining the damage:

- ❖ Find attacks from which the vulnerability derives, and then find all assets related to these attacks. To find the number of users affected by unavailability of functionalities, we have to check attack paths. In VMCP each attack is tied to one or more attack paths, from these attack paths we can identify which modules (processes) will be affected by a certain attack.

Discoverability

Discoverability looks at how difficult it is to discover a vulnerability. This element depends upon expertise and personal judgment. VMCP does not have assistive tools to help determine it. Moreover, discoverability might change dramatically from a 5-score vulnerability to a 15-score one right after the vulnerability is publicly published. The following table is the definition of the three security level.

5	10	15
The bug is obscure, and it is unlikely that users will work out damage potential.	The vulnerability is in a seldom-used part of the product, and only a few users should come across it. It would take some thinking to see malicious use.	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable.

Examples

This section contains identified vulnerabilities from the three samples: an online shopping cart Jspcart, Duke's Bank Application and Credit Card Payment Component. Appendix A, Appendix B and Appendix C have the complete information of these three samples.

Description	User password is stored as plain text in database				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	10	5	15	10	11

Description	Lack of password complexity enforcement, say password retry logic				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	10	5	15	15	12

Description	Missing or weak input validation at the server				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	15	15	15	15	15

Description	Failure to validate cookie input				
Rate					
DAMAGE	REPRODUCABILIT	EXPLOITABILIT	AFFECTE	DISCOVERABILIT	OVERAL

	Y	Y	D USERS	Y	L
15	15	15	15	10	14

Descriptio n	Failure to validate cookie input				
Rate					
DAMAGE	REPRODUCABILIT Y	EXPLOITABILIT Y	AFFECTE D USERS	DISCOVERABILIT Y	OVERAL L
15	15	15	15	10	14

Descriptio n	Failure to encode output leading to potential cross-site scripting issues				
Rate					
DAMAGE	REPRODUCABILIT Y	EXPLOITABILIT Y	AFFECTE D USERS	DISCOVERABILIT Y	OVERAL L
15	10	10	15	5	11

Descriptio n	Exposing an administration function through the customer-facing Web page				
Rate					
DAMAGE	REPRODUCABILIT Y	EXPLOITABILIT Y	AFFECTE D USERS	DISCOVERABILIT Y	OVERAL L
15	10	5	15	15	12

Descriptio	Exposing exception details to the client

n					
Rate					
DAMAGE	REPRODUCABILIT Y	EXPLOITABILIT Y	AFFECTE D USERS	DISCOVERABILIT Y	OVERAL L
10	15	10	15	15	13

11 Certify the component

Based on the rated vulnerabilities, the component can be certified with a security level that indicates the degree to which the users of the component assume risk by using the component in their application. In addition, a certification report should be granted to the component vender. The certification report includes:

- Security level
- Identified and rated vulnerabilities

Security Level
Security level states how secure a given component is. According to the rated vulnerabilities, VMCP rates component into three degrees: high, moderate and low.

- **High** Although there are vulnerabilities existing they are not critical and will not cause serious problem and/or losses to the users.
- **Moderate** The vulnerabilities are serious and might cause serious problems or losses to the user but the risks are still acceptable.
- **Low** The result is catastrophic and not acceptable once the vulnerabilities are exploited by attackers to implement an attack successfully.

When determining security level, VMCP looks at the weakest vulnerability (vulnerability with the highest score) instead of the overall rating of all identified vulnerabilities. VMCP believes a hacker might take over a system once he/she can break the weakest point. Thus the security degree of a system is determined by the weakest point of the system. VMCP divides the 15-ponit vulnerability score into 3 sections respectively representing the three security levels. Table 11-1 defines the three security levels in details.

Table 11-1: Component Security Levels

Security Level	Description	Criteria
High	Security problems are moderately Serious	The highest score of all vulnerabilities rated from 5 to 7
Moderate	Security problems are serious but still acceptable	The highest score of all vulnerabilities rated from 8 to 11
Low	Security problems are catastrophic	The highest score of all vulnerabilities rated from 12 to 15

12 Conclusion

The objective of this thesis is to propose a new product-based certification process to certify the security in components used by e-commerce applications. Within this scope, VMCP has been developed. It works on design specifications and source code using white box technologies to identify software vulnerabilities and to evaluate risk associated with these vulnerabilities. The security certification, which indicates the security level of the component, is then generated based on the identified and rated vulnerabilities.

In order to check how the whole process works on real business project, I applied VMCP to three sample components: Jspcart(an online shopping cart), Duke's Bank Application and Credit Card Payment Component. Detailed information can be found in appendix A, appendix B, and appendix C. Thereinafter I will address several significant points that have come out from the three samples.

Iteratively vs. Sequentially

VMCP is iterative rather than sequential. By iterating the modeling process, we can gradually refine our modeling result when we become more and more familiar with the component. Since VMCP is built upon the assumption that users of this process know nothing about the component at the time when they start to certify the component, the first part of this process is to understand the component and to obtain useful information regarding identifying attacks. The second part is to verify if and how the component in question defends itself again these identified attacks. Vulnerabilities are supposed to be revealed in the verification procedures. Apparently the second part and even the modeling result depend greatly on the output of the first part, the information gathering phase. Information gathering is always iterative. For example, when modeling attack paths, we start with the identified entry points and assets to figure out the logical paths within the component that connect entry points and assets. It is very possible that new entry points and assets may be found in this procedure. The second part is also iterative, when verifying how a component defends itself, we spend much time on code and design review and thus get more and more familiar with the component. As a result, we are very likely to find out new attacks.

Learning Curve

Regarding learning curve, two points were observed. The first point is that time is reduced greatly for the same type of components. When I applied VMCP for the first to the online shopping cart, everything seemed new to me even though it is me who creates this process. I had to learn how to see the component from the security perspective, how to identify entry points and assets, how to use attack tree and so on. When I came to the second sample, Duke's Bank, the certification time was reduced dramatically because of two reasons. First, I was used to the process and technologies used. Second, Duke's Bank and Jspcart belong to the same type of component. Both are web-based. They share lots of common feathers regarding security. The third sample, Credit Card Payment Component, took me more time than the second did, as it is totally different from the other two samples. The Credit Card Payment Component is not web-based. It has no user interfaces but only provides APIs. There are not too many security feathers shared between these two types of components.

The second point is that the effect of the learning curve varies for each steps. Steps such as gain an architecture overview, model the component from an adversary's view and rate vulnerabilities do not depend too much on the component-specific situation and thus time is reduced greatly once the users get used to the process. For component-specific steps such as identifying vulnerabilities and modeling attack paths, time changes slightly, as they totally determined by the details of the component being certified.

Evolvement of the Taxonomy and Attack Trees

Software industry is changing very rapidly. New technologies appear almost daily. Hackers are always able to catch up with the latest technologies, either using them to attack existing systems or inventing even more advanced counter-technologies and using them to attach systems built by these so-called new, secure technologies. As a result, new attacks and vulnerabilities are publicly reported periodically. To reflect these new attacks and vulnerabilities, the taxonomy used by VMCP should also be refreshed periodically. Attack trees are the concrete implementation of attacks, so whenever there are new attacks added to the taxonomy, new attack trees should be developed. Also whenever approaches and technologies used by hackers to perform existing attacks change, attack trees should be updated to reflect these changes.

Expertise Required

VMCP greatly reduces the dependency on expertise knowledge and personal experience. VMCP achieves this by introducing the security taxonomy and attack trees. Developing a security taxonomy and attack trees are really security-knowledge-intensive, however the Software Certificate Service Providers can hire an external consulting company to help with these difficult tasks. Once the security taxonomy and attack trees are built, VMCP will become straight forward. The process itself does not need too much security knowledge and experience.

Major Advantage

VMCP performs Validation & Verification in a time- and cost- effective manner. V & V tasks, such as code review, are usually very time-intensive. However VMCP creates an approach to conduct these tasks effectively. VMCP uses attack paths to help identify vulnerabilities, which allows a team to check places (attack paths) of highest risk within a component for vulnerabilities. This enables the team to identify which sections of component would be best served by a code review. VMCP also allows a team to decide which bugs reported by code analysis tools are most likely to result in vulnerabilities and enables the team to reduce the volume of investigation required.

Weakness

VMCP depends so much on Data Flow Diagrams that the correctness of Data Flow Diagrams affects the result of the certification process. Data Flow Diagram is a very important element of VMCP. It is used to help model attack paths, and identify attacks and vulnerabilities. However, not all components to be certified will come with Data Flow Diagrams. If a component come in without Data Flow Diagrams, the certification team has to build the Data Flow Diagrams itself, which makes it possible that some portions of the Data Flow Diagrams might be missed or some portions are drawn incorrectly, because the certification team is not as familiar with the components as the development team is. If this really happens, the accuracy of the certification will be greatly discounted.

In conclusion, VMCP is a product-based security certification process, which is supposed to be conducted in an iterative manner. It has a positive learning curve, especially when it is applied to the same type of components, certification time will be reduced dramatically once the users get used to the process. The use of security taxonomy and attack trees reduce the dependency on expertise knowledge and personal experience, however they need periodical maintenance in order to reflect the most recent attacks and vulnerabilities. Though VMCP depends so much on Data Flow Diagrams that the correctness of Data Flow Diagrams affects the result of the certification process, it has its overwhelming advantage that V&V are performed in a time- and cost- effective manner.

References

1. "The Ten Most Critical Web Application Security Vulnerabilities", OWASP, 2004. (at: http://www.owasp.org/documentation/topten.html)
2. Lebanidze, Eugene. "Securing Enterprise Web Applications at the source: An Application Security Perspective", OWASP, 2005 (at: http://www.owasp.org/docroot/owasp/misc/Securing_Enterprise_Web_Applications_at_the_Source.pdf)
3. Curphey, M. "A Guide to Building Secure Web Applications", OWASP, 2005. (at: http://www.owasp.org/documentation/guide.html)
4. Vijayarahavan, Giri. "A Taxonomy of E-commerce Risks and Failures", CSTER, 2003. (at: http://www.testingeducation.org/a/tecrf.pdf)
5. Frank Swiderski and Window Snyder. "Threat Modeling", Microsoft Press, 2004.
6. J.D. Meier, Alex Mackman, Blaine Wastell. "Threat Modeling Web Applications", Microsoft Corporation, 2005. (at: http://msdn2.microsoft.com/en-us/library/ms978516.aspx#tmwa_webapplicationsecurityframe)
7. Anup K. Ghosh and Gary McGraw. "An Approach for Certifying Security in Software Components". Reliable Software Technologies, 1998 (at: http://www.cigital.com/papers/download/cert.pdf)
8. Robyn R. Lutz and Robert M. Woodhouse. "Bi-directional Analysis for Certification of Safety-Critical Software". California Institute of Technology Pasadena, CA 91109-8099, 1998.
9. David P. Gilliam, John C. Kelly, John D. Powell and Matt Bishop. "Reducing Software Security Risk through an Integrated Approach". 26th Annual NASA Goddard Volume , Issue , 2001 pp 36 – 42, 2001
10. Ramaswamy Chandramouli and Mark Blackburn. "Automated Testing of Security Functions using a combined Model & Interface driven Approach". Proceedings of the 37th Hawaii International Conference on System Sciences, 2004 (at: http://csdl2.computer.org/comp/proceedings/hicss/2004/2056/09/205690299b.pdf)
11. George Fink and Matt Bishop, "Property-Based Testing; A New Approach to Testing for Assurance", ACM SIGSOFT Software Engineering Notes 22(4) pp. 74–80, 1997
12. George Fink and Karl Levitt. "Property-Based Testing of privileged Programs". Department of Computer Science, University of California. 1994 (at: http://seclab.cs.ucdavis.edu/papers/pdfs/gf-kl-94.pdf)
13. Constance Heitmeyer. "SCR: A Practical Method For Requirements Specification". Naval Research Laboratory, Washington, DC. 1998 (at: http://chacs.nrl.navy.mil/publications/chacs/1998/1998heitmeyer-DASC98.pdf)
14. James Bach, "Heuristic Risk-Based Testing", Software Testing and Quality Engineering Magazine, November 1999, pp 96 - 98.

15. K. Jiwnani and M. Zelkowitz. "Maintaining Software with a Security Perspective", International Conference on Software Maintenance (ICSM'02) pp 194 - 203. 2002
16. Ulf Lindqvist and Erland Jonsson. "A Map of Security Risks Associated with Using COST". In *Computer,* Vol. 31, No. 6 June 1998, pp 60-66.
17. Paul Gerrard and Neil Thompson. "Risk-Based E-Business Testing". Artech House. 2002
18. Gary McGraw and John Viega. "Building Secure Software: How To Avoid Security Problems The Right Way". Addison Wesley, Reading, Mass., 2001
19. Weigang Zhao, "A study of web-based application architecture and performance measurements". School of Information Systems Curtin University, Australia. 1999 (at: http://ausweb.scu.edu.au/aw99/papers/zhao/paper.html)
20. J.D. Meier, Alex Mackman, Michael Dunner, Srinath Vasireddy, Ray Escamilla and Anandha Murukan, "Improving Web Application Security: Threats and Countermeasures". Microsoft. 2003. (at: http://msdn2.microsoft.com/en-us/library/ms994921.aspx)
21. Paul Saitta_, Brenda Larcom, and Michael Eddington. "Trike v.1 Methodology Document [Draft]". 2005. (at: http://www.octotrike.org/Trike_v1_Methodology_Document-draft.pdf)
22. Jeffrey Voas, "An Approach to Certifying Off-the-Shelf Software Components". IEEE Computer, 31(6):53–59 June 1998.
23. Jeffrey Voas, "Defensive Approaches to Testing Systems that Contain COTS and Third-Party Functionality", Reliable Software Technologies Corporation, Sterling, VA., 1998 (at: http://www.cigital.com/papers/download/ictcs98.pdf)
24. Hangkon Kim, "A Framework for Security Assurance in Component Based Development", Springer Berlin, 10.1007/11424826_5, 2004
25. Anup K. Ghosh, "Certifying Security of Components used in Electronic Commerce", Reliable Software Technologies Corporation, Sterling, VA., 1998 (at: http://www.objs.com/workshops/ws9801/papers/paper001.html)
26. Andrew P. Moore, Robert J. Ellison and Richard C. Linger, "Attack Modeling for Information Security and Survivability", CMU/SEI-2001-TN-001, 2001
27. Jeffrey Voas, "Software Certification Laboratories To Be or Not to Be Liable?", Reliable Software Technologies Corporation, Sterling, VA., 1998 (at: http://www.stsc.hill.af.mil/Crosstalk/1998/04/certification.pdf)
28. Jeffrey Voas, "User Participation-Based Software Certification", Eurovav, pp 267 - 276, 1999
29. Kamran Yapub, "Modeling Security Requirements of Target of Evaluation and Vulnerabilities in UML", Lulea University of Technology, Sweden, 2006. (at: http://epubl.ltu.se/1653-0187/2006/31/LTU-PB-EX-0631-SE.pdf)

30. Jeffrey Voas and Jeffery Payne, "Dependability certification of software components", Journal of Systems and Software, NO. 52, pp. 165-172, 2000.
31. Krishnan Rama, "Securing Your ASP.NET Web Applications", 2004 (at: http://www.c-sharpcorner.com/UploadFile/krishvr/securewebapp11262005011914AM/securewebapp.aspx?ArticleID=1a878159-02d3-4d13-90d8-fe55d6c571f6)
32. Mitja Kolšek, "Session Fixation Vulnerability in Web-based Applications", ACROS Security, 2002 (at: http://www.acros.si/papers/session_fixation.pdf)
33. John Steven and Gunnar Peterson, "A Metrics Framework to Drive Application Security Improvement", IEEE COMPUTER SOCIETY, 2007 (at: http://www.arctecgroup.net/pdf/0703-OWASPMetrics.pdf)
34. Seyit Ahmet C¸ amtepe and B¨ulent Yener, "A Formal Method for Attack Modeling and Detection", Computer Science Department, Rensselaer Polytechnic Institute, Troy, NY 12180. 2006 (at: http://www.cs.rpi.edu/research/pdf/06-01.pdf)
35. Fredrik Moberg, "Security analysis of an information system using an attack tree-based methodology", Chalmers University Of Technology, 2000 (at: http://antareja.rvs.uni-bielefeld.de/~made/Seminar/Attack-Tree/fredrik.moberg-thesis.pdf)
36. Yue Chen, Barry Boehm and Luke Sheppard, "Value Driven Security Threat Modeling Based on Attack Path Analysis", University of Southern California, Los Angeles, CA, 90089-0781, USA. 2006
37. Bruce W. Weide, Paolo Bucci, Wayne D. Heym, Murali Sitaraman and Giorgio Rizzoni, "Issues in Performance Certification for High-Level Automotive Control Software", Software Engineering for Automotive Systems, Volume 30 , Issue 4 , July 2005, pp 1 – 6.
38. S.W. Smith, "Turing is from Mars, Shannon is from Venus: Computer Science and Computer Engineering", IEEE Security & Privacy, vol. 3, no. 2, 2005, pp. 66-69
39. Paolo Donzelli, Marvin Zelkowitz, Victor Basili, Dan Allard and Kenneth N. Meyer, "Evaluating COTS Component Dependability in Context", IEEE Software, Volume 22, Issue 4, July 2005, pp 46 – 53.
40. Giri Vijayaraghavan and Cem Kaner,, "Bug Taxonomies: Use Them to Generate Better Tests", STAR EAST 2003, Orlando, FL. 2003 (at: http://testingeducation.org/a/bugtax.pdf)
41. Jeffrey Voas, "Certifying Software for High Assurance Environments", IEEE Software, Volume 16, Issue 4, July 1999, pp 48 -54.
42. John D. Howard, "An Analysis of Security Incidents on the Internet", Pittsburgh, Pennsylvania 15213 USA, 1997 (at: http://www.cert.org/research/JHThesis/)
43. Michael Howard and James Whittaker, "Violating Assumptions with Fuzzing", IEEE. Security & Privacy, pp 58 - 62, March/April 2005

44. Gary McGraw, "Software Assurance for Security", Reliable Software Technologies, Sterling, VA. 1999. (at: http://www.cigital.com/papers/download/ieee-computer-secass.pdf)
45. A UML Specification Language targeting Security Risk Assessment, 2006 (at: http://coras.sourceforge.net/)
46. M. Schiffman, "Common Vulnerability Scoring System (CVSS)". 2006 (at: http://www.first.org/cvss/)
47. Caleb Sima, "security at the next level: are you web application vulnerable?", SPI Dynamics Inc. 2004 (at: http://www.spidynamics.com/whitepapers/webappwhitepaper.pdf)
48. Kanta Jiwnani and Marvin Zelkowitz , "Security Testing using a Susceptibility Matrix", Chillarege Press, FastAbstract ISSRE 2002. (at: http://www.cs.umd.edu/~mvz/pub/sectest.pdf)
49. Gary McGraw, "Knowledge for Software Security", Security & Privacy Magazine, IEEE, Volume: 3, Issue: 2, March-April 2005, pp 74 -78.
50. M.G. Graff and K.R. van Wyk, "Secure Coding—Principles & Practices". O'Reilly, first edition 2003.
51. Yngve Espelid, Lars-Helge Netland, Khalid Mughal, and Kjell Jørgen Hole, "Simplifying Client-Server Application Development with Secure Reusable Components", Department of Informatics, University of Bergen. 2006 (at: http://www.nowires.org/Papers-PDF/SimplifyingClientServerAppDev.pdf)
52. Jane Cleland-Huang, Mark Denne, Ghazy Mahjub, and Nilesh Patel, "A Goal-Oriented Approach for Mitigating Security and Continuity Risks", DePaul University and Symantec Corp, 10.1007/978-0-387-34831-5_10, 2006
53. Thuy D. Nguyen, Cynthia E. Irvine, and Douglas R. Kane Jr., "Using Common Criteria Methodology to Express Informal Security Requirements", Proc. International Symposium on Secure Software Engineering, Arlington, VA, March 2006, pp. 75-85.
54. Amit Paradkar, Suzanne McIntosh, Sam Weber, David Toll, Paul A. Karger, and Matthew Kaplan, "Chicken & Egg: Dependencies in Security Testing and Compliance with Common Criteria Evaluations", IEEE International Symposium on Secure Software Engineering (ISSSE '06). 13-15 March 2006, Arlington, VA IEEE Computer Society. p. 65-74.
55. Suvda Myagmar and William Yurcik, "Why Johnny Can Hack: The Mismatch between Vulnerabilities and Security Protection Standards", NCSA University of Illinois at Urbana-Champaign. 2006 (at: http://citeseer.ist.psu.edu/cache/papers/cs2/496/http:zSzzSzwww.projects.ncassr.orgzSzthreatmodelingzSzissse06.pdf/myagmar06why.pdf)
56. Petr Hejda, "Architectural Model for User interfaces of Web-based Applications", Rockwell Automation Research Center Prague Czech Republic. 2000 (at: http://ui4all.ics.forth.gr/UI4ALL-2000/files/Short_papers/Hejda.pdf)
57. G. Hoglund and G. McGraw, "Exploiting Software—How to Break Code". Addison-Wesley, first edition 2004.

58. K.J. Soo Hoo, "How Much is Enough? A Risk-Management Approach to Computer Security". Ph.D. dissertation, Graduate School of Engineering, Stanford University, 2000. (at: http://citeseer.ist.psu.edu/cache/papers/cs/25847/http:zSzzSzcisac.stanford.eduzSzdocszSzsoohoo.pdf/how-much-is-enough.pdf)
59. Gerard J. Holzmann, "Trends in Software Verification", JPL Laboratory for Reliable Software California Institute of Technology Pasadena, CA. 2003 (at: http://spinroot.com/gerard/pdf/fme03.pdf)
60. T.A. Henzinger, R. Jhala, et al., "Software Verification with Blast.", Proc 10th SPIN Workshop on Model Checking Software, LNCS 2648, Springer-Verlag, 2003. pp 235 -239.
61. Steven R. Rakitin, "Software Verification and Validation for Practitioners and Managers", Artech House Publishers; 2nd edition. 2001
62. Mulcahy, G. "J2EE and .NET Security," CGI Security, 2002. (at: http://www.cgisecurity.com/lib/J2EEandDotNetsecurityByGerMulcahy.pdf)
63. Ross, R. "Guide for the Security Certification and Accreditation of Federal Information Systems," NIST SP800-37. 2004 (at: http://whitepapers.silicon.com/0,39024759,60116708p,00.htm)
64. Conklin, W., White, G., Cothren, C. "Principles of Computer Security." Career Education, first edition, 2004
65. B. Schneier, "Attack trees: Modeling security threats," Dr. Dobb's Journal, 1999,12(24):21-29
66. Japzon, Eddie Manuel, Henry L. Cummings and Jr., John W. "Managing Obsolescence in the U.S. Department of Defense Acquisition Environment". RIAC, Department of Defense. 2000. (at: http://quanterion.com/RIAC/Library/Library.asp?ArgVal=45555-049)
67. Michael Howard and David LeBlanc. "Writing Secure Code, Second Edition". Microsoft Press. 2003
68. Schneier, Bruce. "Attack Trees." 21-29. Dr. Dobb's Journal of Software Tools 24, 12 (December 1999): 21-29.
69. Leveson, N. O., "Safeware: System Safety and Computers", Addison-Wesley, Reading MA, 1995
70. Prowell, S. J., Trammell, C. J., Linger, R. C., and Poore, J. H. "Cleanroom Software Engineering: Technology and Process". Boston, MA: Addison Wesley Longman, 1999.

Appendix A: Sample I - JSPCART

Overview

JSPCART is an open source Shopping Cart developed using JSP, running on Tomcat and MySQL on any platform. (http://jspcart.neurospeech.com/). It consists of two components. One is the user component by which internet users shop (Figure A-1-1). The other is exclusively for backend administrators to maintain the products, catalogs, orders and user accounts (Figure A-1-2).

Figure A-1-1 JSPCART User Component

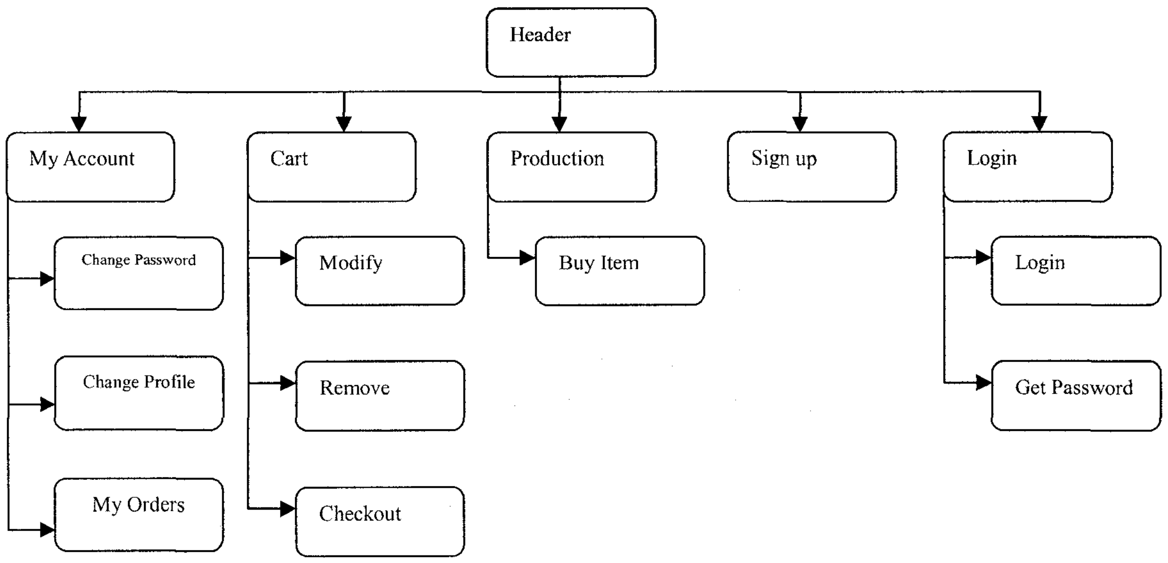
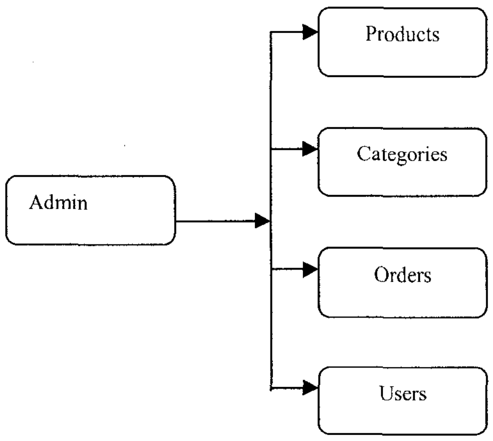


Figure A-1-2 JSPCART Admin Component



Develop a common security risks and vulnerabilities taxonomy

We use the taxonomy described in chapter 4 for this sample as it covers the most common application level attacks and vulnerabilities that plague web-based enterprise applications.

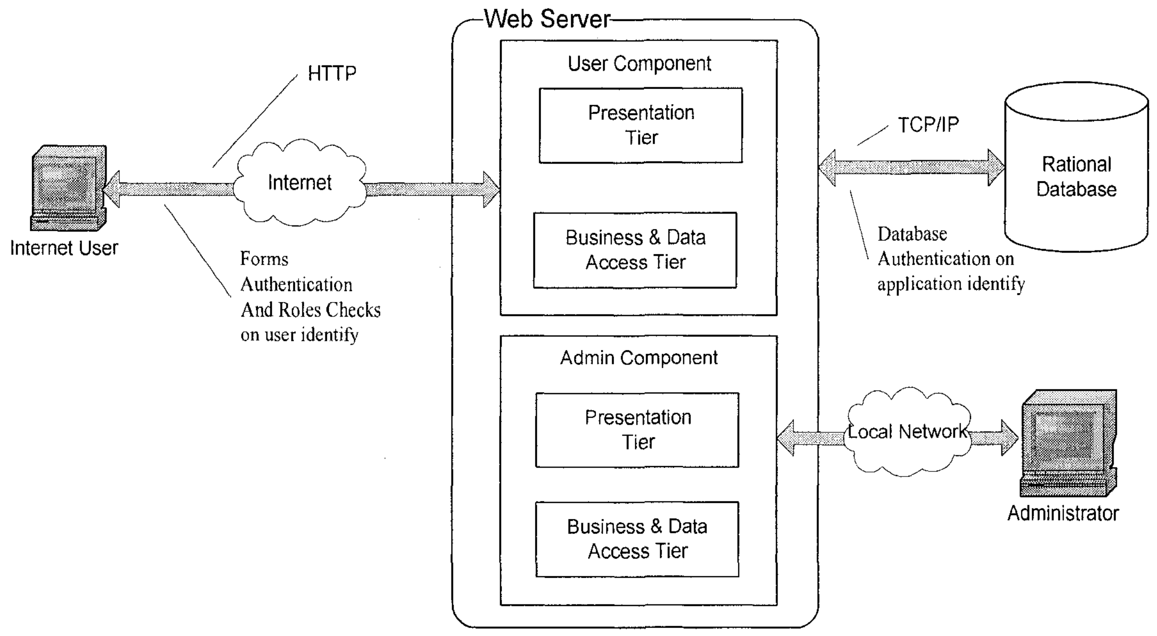
Gain an architecture overview from the security perspective

Here we try to understand the JSPCART from the perspective of security. Our goal is to identify the deployment Scenarios, the key functionality, characteristics, and roles. This will help us to identify relevant attacks later.

End-to-End Deployment Scenario

JSPCART is a web-based shopping cart with a relational database back end. Both the user and admin components are accommodated in the web server. The user component is available to Internet users via Internet. The admin component is only available to the local administrators via a local intranet. Figure 13.3 shows the deployment scenario.

Figure A-1-3 JSPCART deployment Scenario


Web Server
User Component
Presentation Tier
Business & Data Access Tier
Admin Component
Presentation Tier
Business & Data Access Tier
HTTP
Internet
Internet User
Forms Authentication And Roles Checks on user identify
TCP/IP
Rational Database
Database Authentication on application identify
Local Network
Administrator

Roles

Identify who can do what and cannot do what in the component.

Anonymous user:

Legal activities:

- Browse product or catalog list

- Create a cart, and browse or modify items in the cart
- Create a user account

Illegal activities:

- Checkout items in a cart created by his/her own.
- Browse, modify or checkout items in carts created by others
- Browse or modify user profile
- Change products or catalogs information
- Browse or modify orders

Authenticated user:

Legal activities:

- Browse product or catalog list
- Create a cart, and browse, modify or checkout items in the cart.
- Create a user account
- Browse or modify his/her own profile
- Browse or modify his/her own orders

Illegal activities:

- Browse, modify or checkout items in carts created by other users
- Browse or modify profiles of other users
- Change products or catalogs information
- Browse or modify orders created by other users

Administrators

Legal activities:

- Full control on user accounts, carts, products, catalogs and orders.

Illegal activities:

- None

Key Usage Scenarios

Important application scenarios are:

- Anonymous user browses the product pagers.
- Anonymous user adds and/or removes items to the shopping cart, modify the item quantity.
- Anonymous user logs in to authenticate prior to placing an order.
- Anonymous user creates a new account prior to placing an order.
- Authenticated user places, browses and modifies an order.
- Authenticated user browses and/or modifies his/her user profile.
- Administrator manipulates user profiles, products, catalogs and orders.

Technologies

Identifying technologies helps us to focus on technology-specific attacks. The JSPCart component uses the following technologies:

- Presentation logic: JSP
- Business logic: Java Class Libraries, JavaBean
- Data access logic: JDBC, embased SQL

Application Security Mechanisms

The major application security mechanisms are:

- Users and administrators are authenticated with Forms authentication.
- Application is authenticated at the database server by using Windows authentication.
- Roles are used to authorize access to business logic.

Model the component from an adversary's View

VMCP looks at a component from an adversary's perspective to anticipate attack goals. VMCP is based on the following two premises:

1. An adversary will not attack the system without assets of interest.

2. An adversary cannot attack a system without entry points, interfaces the system has with the outside world

By identifying all assets of interest and entry points, adversaries gather the basic information for undertaking attacks, what are the attack targets, and where they can enter the system to reach the assets. Then what is left is to figure out where is the most possible points they can break the system. These usually are the trust boundaries, where trust levels change. We have modeled JSPCART from these three aspects: assets of interest, entry points and trust boundaries, to get enough information for later attack analysis.

Identified Assets

ID	Name	Description	Trust Roles	Category
A1	User credentials	Username and password	Authenticated user (CRUD)	Application data
A2	User profile	User profiles stored in back-end database	Authenticated user (CRUD)	Application data
A3	Cart	Cart information stored in back-end database, e.g. the name and quantity of items in a cart.	Authenticated user (CRUD) Anonymous user (CRUD)	Application data
A4	Order	Orders stored in back-end database	Authenticated user (CRUD)	Application data
A5	Product	Products stored in back-end database	Anonymous user (R) Authenticated user (R) Administrator (CRUD)	Application data
A6	Shopping information	Shopping information for orders.	Authenticated user (CRU) Administrator (CRUD)	Application data
A6	Catalog	Catalogs stored in back-end database	Administrator (CRUD)	Application data
A7	Process	Processes running within the same machine where the component is running	N/A	System resource
A8	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource
A9	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A10	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

Identified entry point

ID	Name	Description	Trust Roles	Category
EP1	Web server listening port	The port on which the Web server listens. All web pages are layered on this entry point.	All	Network Interface
EP2	Login Page	Get user credentials and passed them to server side for authentication	All	User Interface
EP3	Signup Page	Create a user profile and pass it back to server side	All	User Interface
EP4	Cart Page	Users modify quantity of items and remove items from cart.	All	User Interface
EP5	ChangePassword page	Authenticated users change password	Authenticated user	User Interface
EP6	ChangeProfile page	Authenticated users change profile	Authenticated user	User Interface
EP7	Shipping page	Authenticated users input shipping information	Authenticated user	User Interface
EP8	MyOrder Page	Authenticated users enquiry order details and/or cancel orders	Authenticated user	User Interface

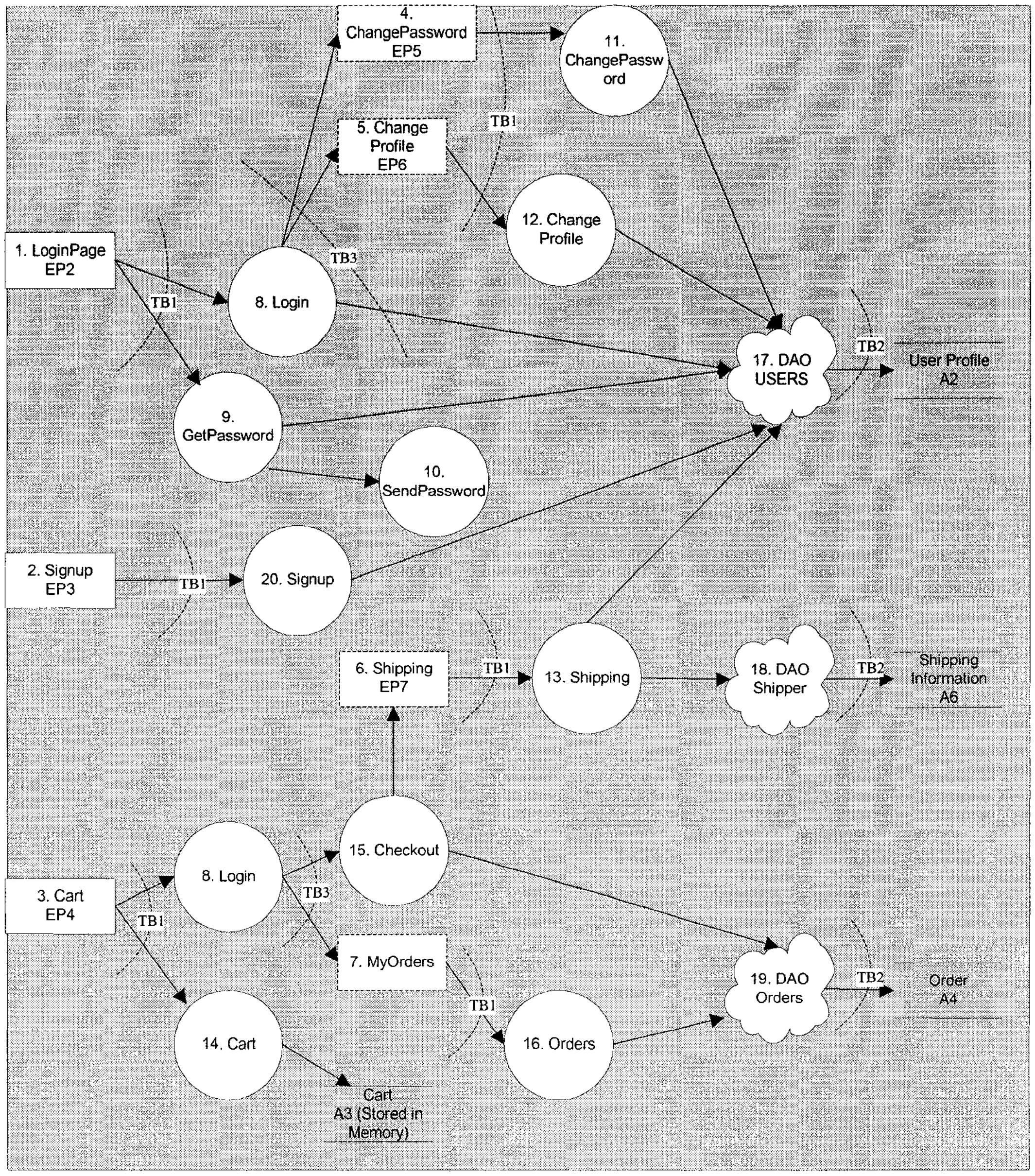
Identified Trust Boundary

ID	Name	Description	Category
TB1	Client Boundary	The boundary between remote clients and backend applications in server side.	Network Boundary
TB2	Login Boundary	The boundary between the login model and other models	User Interface Boundary
TB3	Database Boundary	The boundary between the application and database.	System Boundary

Model attack paths and security criteria

Using the theory and method presented in chapter 7, we have modeled a partial DFD of JSPCart, attack paths and security criteria.

Figure A-5-1 DFD of JSPCart Integrated with Trust Boundaries



Attack paths and security criteria

Path ID	P1
Description	1→8→17 Anonymous user inputs username and password at webpage. Login model compares the credential passed by login pager with that got from data access object (DAO) users.

Passenger	Anonymous user
Action on Asset	User profile (R)
Rules	Action is only allowed to the profile of user's own.

Path ID	P2
Description	1→8→4→11→17 5. Authenticated users input new password on ChangePasswrod page 6. ChangePasswrod page passes the new password to server side model ChangePassword. 7. ChangePassword pass it to DAO users. 8. DAO users update the user profile.
Passenger	Authenticated user
Action on Asset	User profile (U)
Rules	Action is only allowed to the profile of user's own.

Path ID	P3
Description	1→8→5→12→17 5. Authenticated users change profile on ChangeProfile page 6. ChangeProfile page passes the changed profile to server side model ChangeProfile. 7. ChangeProfile pass it to DAO users. 8. DAO users update the user profile.
Passenger	Authenticated user

Action on Asset	User profile (U)
Rules	Action is only allowed to the profile of user's own.

Path ID	P4
Description	1→9→17 Anonymous user input username. Getpassword model retrieves the password using the username through DAO users.
Passenger	Anonymous user
Action on Asset	User profile (R)
Rules	Action is only allowed to the profile of user's own.

Path ID	P5
Description	2→20→17 Anonymous user input signup information. Signup model create the profile through DAO users.
Passenger	Anonymous user
Action on Asset	User profile (C)
Rules	NA

Path ID	P6
Description	3→8→15→6→13→17 Authenticated user check out the items in the shopping cart, and input shopping information. Shopping model retrieves some basic

	personal information from user profile through DAO users
Passenger	Authenticated user
Action on Asset	User profile (R)
Rules	Users can only retrieve their own profile.

Path ID	P7
Description	3→8→15→6→13→18 Authenticated user check out the items in the shopping cart, and input shopping information. Shopping model stores shopping information to database though DAO Shipper.
Passenger	Authenticated user
Action	Shipping information (CRU)
Rules	Users can only check out the cart created by themselves

Path ID	P8
Description	3→8→15→19 Authenticated user checkout the shopping cart. Checkout model creates an order through DAO orders.
Passenger	Authenticated user
Action on Asset	Orders(C)
Rules	Users can only check out the cart created by themselves

Path ID	P9

Description	3→8→7→16→19 Authenticated user updates orders.
Passenger	Authenticated user
Action on Asset	Orders (CRUD)
Rules	User can only update orders under their identifier.

Attack analysis

In accordance with the attack analysis methodology described in chapter 8, we group together paths P1 to P9 as they have the same category of entry point and assets and choose path P6 as a candidate for detailed attack analysis because:

- its passenger is authenticated user which is has the highest privilege of the group.
- it has only 'READ' access to the asset
- it covers all three trust boundaries in the group.

Possible attacks along path P6

ID	A1	Name	Dictionary Store Attack
Description	Brute force attacks occur against the dictionary store		
STRIDE	T. I. E		

ID	A2	Name	Client Credentials Attack
Description	Network eavesdropping occurs between the browser and Web server to capture client credentials.		
STRIDE	S		

ID	A3	Name	Spoof Identity in Cookies

Description	An attacker captures an authentication cookie to spoof identity
STRIDE	S

ID	A4	Name	SQL injection
Description	SQL injection occurs, enabling an attacker to exploit an input validation vulnerability to execute commands in the database and thereby access and/or modify data.		
STRIDE	T. I. E		

ID	A5	Name	Cross-Site Scripting
Description	Cross-site scripting occurs when an attacker succeeds in injecting script code.		
STRIDE	T. I. D. E		

ID	A6	Name	Cookie Replay
Description	Cookie replay or capture occurs, allowing an attacker to spoof identity and access the application as another user.		
STRIDE	S		

ID	A7	Name	Information Disclosure
Description	Information is disclosed and sensitive exception details are revealed to the client.		
STRIDE	I		

ID	A8	Name	Server Attack
Description	An attacker manages to take control of the servers, gain unauthorized access to the database, and run commands against the database.		

STRIDE	T. D. E

ID	A9	Name	Individual Accountability
Description	Lack of individual accountability		
STRIDE	R		

ID	A10	Name	Form Manipulation
Description	Query string and form field manipulation		
STRIDE	T. E		

Vulnerabilities identified and rates

Use the methodologies presented in chapter 9 and chapter 10 we have identified and rated the flowing vulnerabilities:

Description	User password is stored as plain text in database				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	10	5	15	10	11

Description	Lack of password complexity enforcement, say password retry logic				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	10	5	15	15	12

Description	Missing or weak input validation at the server				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	15	15	15	15	15

Description	Failure to validate cookie input				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	15	15	15	10	14

Description	Failure to validate cookie input				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	15	15	15	10	14

Description	Failure to encode output leading to potential cross-site scripting issues				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	10	10	15	5	11

Description	Exposing an administration function through the customer-facing Web page				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	10	5	15	15	12

Description	Exposing exception details to the client				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
10	15	10	15	15	13

Certification

Based on the rated vulnerabilities, Security level of JSPCART is certified as low, as it has 5 vulnerabilities rated from 12 – 15.

Appendix B: Sample II - Duke's Bank Application

Overview

Duke's Bank, an online banking application, is an example application from the Java EE (Java Platform, Enterprise Edition) tutorial. It is supposed to run on a Java EE application server. Duke's Bank has two clients: an application client (standalone client) used by administrators to manage customers and account, and a web client used by customers to access account histories and perform transactions. The web client is built using JavaServer Faces technology and the application client is built using Java Swing. The server end is built using EJB and servlets. Data is stored in a database accessed through EJB.

Develop a common security risks and vulnerabilities taxonomy

We use the taxonomy described in chapter 4 for this sample as it covers the most common application level attacks and vulnerabilities that plague web-based enterprise applications.

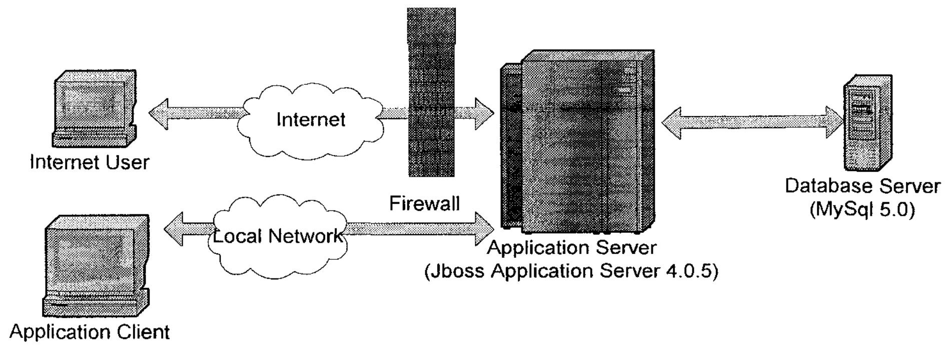
Gain an architecture overview from the security perspective

Here we try to understand the Duke's Bank application from the perspective of security. Our goal is to identify the deployment Scenarios, the key functionality, characteristics, and roles. This will help us to identify relevant attacks later.

End-to-End Deployment Scenario

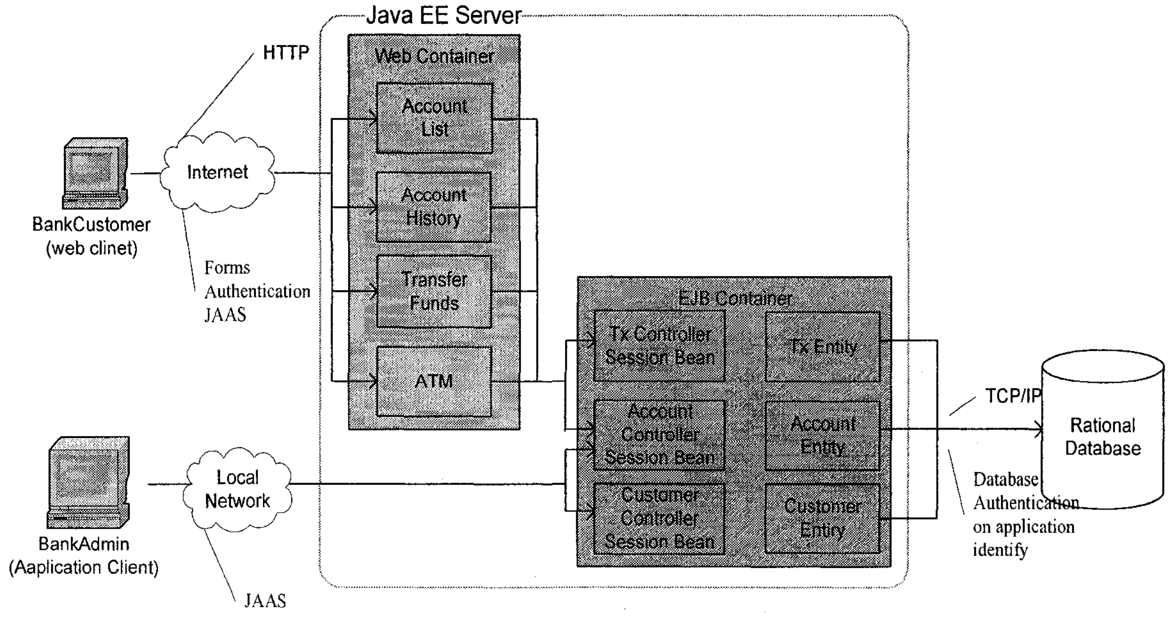
Duke's Bank is designed using standard 3-tier architecture, user-end (client), back-end (server) and database. Duke's Bank has two clients, web and application. The supposed deployment scenario is that web client and server end programs are deployed on a Java EE application server, the application client on a remote (or local) computer, and the database on a database server(physically can be the same machine as the application server). In our sample, we used JBoss Application Server 4.0.5 and Mysql 5.0. Figure B-1-1 is the network topology.

Figure B-1-1 Network topology of Duke's Bank



The network topology gives us a big picture of the environment where the application runs and how it interacts with the external environment. Though network topology uncovers some possible security issues, it is not detailed enough for a security certification. However, we can explore the detailed deployment diagram to gain more informaiton. Usually a detailed deployment diagram includes end-to-end deployment topology, logical layers, key components, key services, Communication ports and protocols, Identities and External dependencies. Figure B-1-2 is the detailed deployment diagram of Duke's Bank.

Figure B-1-2 Detailed deployment diagram of Duke's Bank



Java EE Server
HTTP
Web Container
Account List
Account History
Transfer Funds
ATM
Internet
BankCustomer (web clinet)
Forms Authentication JAAS
EJB Container
Tx Controller Session Bean
Tx Entity
Account Controller Session Bean
Account Entity
Customer Controller Session Bean
Customer Entity
TCP/IP
Rational Database
Database Authentication on application identify
Local Network
BankAdmin (Aaplication Client)
JAAS

Roles

Identify the component's roles: that is, identify who can do what and cannot do what within the application.

BankCustomer:

Legal activities:

- Browse account list
- Check account details
- Check transaction history of an account
- Transfer funds between accounts
- Withdraw and deposit funds

Illegal activities:

  - Browse account list, account details, and transaction history of other customers.
  - Perform transaction on accounts of other customers
  - Perform activities exclusive to BankAdmin

BankAdmin:

Legal activities:

  - Add, update, view and remove customer information
  - Add, view and remove account information
  - Add a new customer to an exist account
  - Remove a customer from an existing account

Illegal activities:

  - Perform transaction on customer's accounts

Key Usage Scenarios

Important application scenarios are:

- BankCustomer browse account list
- BankCustomer check account details
- BankCustomer check transaction history of an account
- BankCustomer transfer funds between accounts
- BankCustomer withdraw and deposit funds
- BankAdmin add, update, view and remove customer information
- BankAdmin add, view and remove account information
- BankAdmin add a new customer to an exist account
- BankAdmin remove a customer from an existing account

Technologies

Identifying technologies helps us to focus on technology-specific attacks. The Duke's Bankuses the following technologies:

- Web client: JavaServer Faces
- Applicaton client: Java Swing interface

- Back-end: EJB , servlets
- Data access logic: Container-Managed Entity Bean
- Authentication and Authorization: JAAS (Java Authentication and Authorization Service)

Application Security Mechanisms

The major application security mechanisms are:

- Users are authenticated with Forms authentication.
- Application is authenticated at the database server by using Windows authentication.
- Roles are used to authorize access to business logic.

Model the component from an adversary's View

VMCP looks at a component from an adversary's perspective to anticipate attack goals. VMCP is based on the following two premises:

1. An adversary will not attack the system without assets of interest.
2. An adversary cannot attack a system without entry points, interfaces the system has with the outside world

By identifying all assets of interest and entry points, adversaries gather the basic information for undertaking attacks, what are the attack targets, and where they can enter the system to reach the assets. Then what is left is to figure out where is the most possible points they can break the system. These usually are the trust boundaries, where trust levels change. We have modeled Duke's Bank from these three aspects: assets of interest, entry points and trust boundaries, to get enough information for later attack analysis.

Identified Assets

ID	Name	Description	Trust Roles	Category
A1	User credentials	Username and password	BankCustomer, BankAdmin (CRUD)	Application data
A2	Customer Data	Customer information stored in back-end database	BankAdmin (CRUD)	Application data
A3	Account Data	Account information stored in back-end database.	BankAdmin (CRUD)	Application data

			BankCustome (RU)	
A4	Transaction Data	Transaction information stored in back-end database	BankCustome (CR)	Application data
A5	Process	Processes running within the same machine where the component is running	N/A	System resource
A6	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource
A7	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A8	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

C: Create, R: Retrieve, U: Update, D: Delete.

Identified entry point

ID	Name	Description	Trust Roles	Category
EP1	Web container listening port	The port on which the Web container listens. All web pages are layered on this entry point.	All	Network Interface
EP2	EJB container listening port	The port on which the EJB container listens. Remote application client communicates with application server via this port.	All	Network Interface
EP3	Logon Page	Get user credentials and passed them to server side for authentication	All	User Interface
EP3	Account List Page	List account details of a customer and transaction history of a account	BankCustomer	User Interface
EP4	Transfer Funds Page	Transfer funds between accounts	BankCustomer	User Interface
EP5	ATM Page	Withdraw and deposit funds	BankCustomer	User Interface
EP6	Customer Info GUI (Application Client)	Manipulate customer information	BankAdmin	User Interface
EP7	Account Info GUI	Manipulate account information	BankAdmin	User Interface

	(Application Client)			

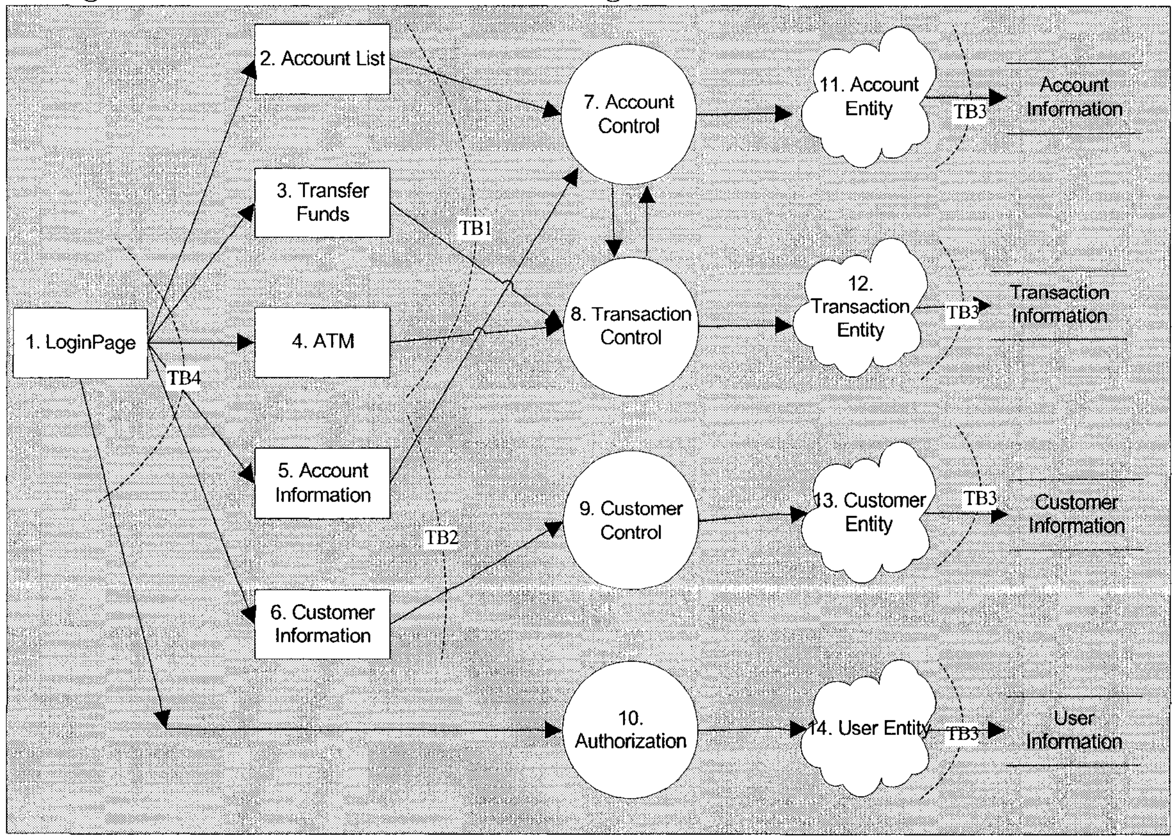
Identified Trust Boundary

ID	Name	Description	Category
TB1	Web Clinet Boundary	The boundary between remote web client and backend application on the server side.	Network Boundary
TB2	Application Clinet Boundary	The boundary between remote application client and backend application on the server side.	Network Boundary
TB3	Database Boundary	The boundary between the application and database.	System Boundary
TB4	Logon Boundary	The boundary between the logon model and other models	User Interface Boundary

Model attack paths and security criteria

Using the theory and method presented in chapter 7, we have modeled the DFD of Duke's Bank, attack paths and security criteria.

Figure B-5-1 EDFD of Duke's Bank Integrated with Trust Boundaries



Attack paths and security criteria

Path ID	P1
Description	1→10→14 Anonymous user inputs username and password at webpage. Authorization model compares the credential passed by logon pager with that stored in back-end database.
Passenger	Anonymous user
Action on Asset	User Information (R)
Rules	Action is only allowed to the user's own credential.

Path ID	P2
Description	1→2→7→11 BankCustomer retrieve account information.
Passenger	BankCustomer
Action on Asset	Account Information (R)
Rules	BankCustomer can only retrieve his/her own account information.

Path ID	P3
Description	1→2→7→8→12 BankCustomer retrieve transaction history of a certain account.
Passenger	BankCustomer
Action on	Transaction Information (R)

Asset	
Rules	BankCustomer can only retrieve his/her own transaction information..

Path ID	P4
Description	1→3→8→12 BankCustomer input instruction of transferring money from one account to the other and transactions are updated to database.
Passenger	BankCustomer
Action on Asset	Transaction Information (C)
Rules	BankCustomer can only transfer money between his/her own accounts.

Path ID	P5
Description	1→3→8→7→11 1. Retrieve account balance from database 2. Update account balance to database
Passenger	BankCustomer
Action on Asset	Account Information (RU)
Rules	BankCustomer can only retrieve/update his/her own account balance.

Path ID	P6

Description	1→5→7→13 BankAdmin maintains account information.
Passenger	BankAdmin
Action on Asset	Account Information (CRUD)
Rules	N/A.

Path ID	P7
Description	1→6→9→13 BankAdmin maintains customer information.
Passenger	BankAdmin
Action	Customer Information (CRUD)
Rules	N/A

Attack analysis

In accordance with the attack analysis methodology described in chapter 8, we choose path P2 and P7 as candidates for detailed attack analysis because:

- Their passenger are BankCustomer and BankAdmin who have the highest privilege.
- P2 has only 'READ' access to the asset (Account Information)
- They cover all four trust boundaries.

Possible attacks along path P2 and P7

ID	A1	Name	Client Credentials Attack
Description	Network eavesdropping occurs between the browser and Web server to capture client credentials.		
STRIDE	S		

ID	A2	Name	Spoof Identity in Cookies
Description	An attacker captures an authentication cookie to spoof identity		
STRIDE	S		

ID	A3	Name	SQL injection
Description	SQL injection occurs, enabling an attacker to exploit an input validation vulnerability to execute commands in the database and thereby access and/or modify data.		
STRIDE	T. I. E		

ID	A4	Name	Cross-Site Scripting
Description	Cross-site scripting occurs when an attacker succeeds in injecting script code.		
STRIDE	T. I. D. E		

ID	A5	Name	Cookie Replay
Description	Cookie replay or capture occurs, allowing an attacker to spoof identity and access the application as another user.		
STRIDE	S		

ID	A6	Name	Information Disclosure
Description	Information is disclosed and sensitive exception details are revealed to the client.		
STRIDE	I		

ID	A7	Name	Server Attack

Description	An attacker manages to take control of the servers, gain unauthorized access to the database, and run commands against the database.
STRIDE	T. D. E

ID	A8	Name	Canonicalization Attack
Description	Canonicalization attacks caused by using user name for security decisions.		
STRIDE	S. T. E		

ID	A9	Name	Configuration Attack
Description	Retrieval of clear text configuration secrets		
STRIDE	T. D		

ID	A10	Name	Individual Accountability
Description	Lack of individual accountability		
STRIDE	R		

ID	A11	Name	Session Attack
Description	Session hijacking and replaying.		
STRIDE	S. T. E		

ID	A12	Name	Encryption Attack
Description	Weak or custom encryption		

STRIDE	S. T. E

ID	A13	Name	Form Manipulation
Description	Query string and form field manipulation		
STRIDE	T. E		

Vulnerabilities identified and rates

Use the methodologies presented in chapter 9 and chapter 10 we have identified and rated the flowing vulnerabilities:

Description	Clear text configuration stored on server				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	5	5	15	5	7

Description	Entity bean do not validate method parameters				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	5	5	15	5	7

Description	Users' activities are not logged				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED	DISCOVERABILITY	OVERALL

			USERS		
5	15	5	15	5	7

Description	Exposing exception details to the client				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
5	5	5	15	15	7

Certification

Based on the rated vulnerabilities, security level of DUKE'S BANK is certified as high, as all its vulnerabilities are rated 5-7.

Appendix C: Sample III - Credit Card Payment Component

Overview

Credit Card Payment Component (CCPC) is an add-on component of osCommerce, a premiere open source e-commerce system, by osCommerce, currently being installed and utilized by 11,400 online stores. osCommerce is developed using PHP and Mysql. The main functions of CCPC are providing Forms to accept credit card information, doing a validation, and storing the information to the Order.

Develop a common security risks and vulnerabilities taxonomy

We use the taxonomy described in chapter 4 for this sample as it covers the most common application level attacks and vulnerabilities that plague web-based enterprise applications.

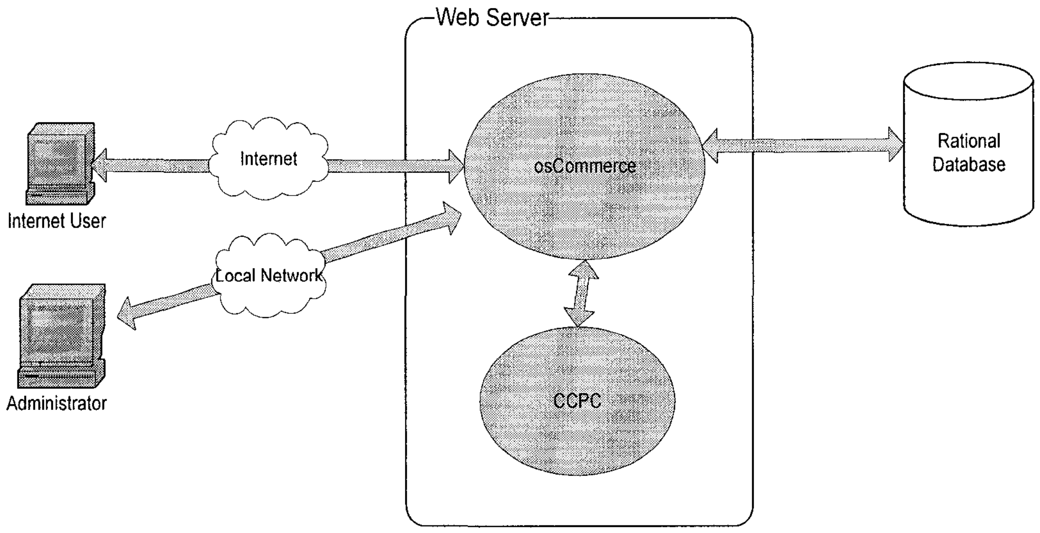
Gain an architecture overview from the security perspective

Here we try to understand the CCPC from the perspective of security. Our goal is to identify the deployment Scenarios, the key functionality, characteristics, and roles. This will help us to identify relevant attacks later.

End-to-End Deployment Scenario

The deployment scenario is relatively simple. It provides APIs for osCommerce to invoke its functions. osCommerce is a web-based e-commerce system, developed using PHP and Mysql Figure 15.1 is a basic deployment scenario of .CCPC.

Figure C-1-1 Deployment Scenario of CCPC



Roles

Identify the component's roles: that is, identify who can do what and cannot do what within the application.

Invoker:

Legal activities:

- Invoke APIs

Illegal activities:

- N/A

Key Usage Scenarios

Important application scenarios are:

- Invoker calls API update_status() to determine if the credit card payment is enabled.
- Invoker calls API javascript_validation() to get a snippet of input validation javascript program.
- Invoker calls API selection() to get Forms to accept user's input
- Invoker calls API pre_confirmation_check() to validate the credit card information.
- Invoker calls API confirmation() to get masked credit card information.
- Invoker calls API process_button() to get Forms with hidden fields of credit card information.
- Invoker calls API before_process() to store card number to order
- Invoker calls API after_process() to email order information to customer.
- Invoker calls API install() to store configuration information to the configuration table.
- Invoker calls API remove() to remove configuration information from the configuration table.

Technologies

Identifying technologies helps us to focus on technology-specific attacks. CCPC uses the following technologies:

- PHP

- SQL query

Application Security Mechanisms

- N/A

Model the component from an adversary's View

VMCP looks at a component from an adversary's perspective to anticipate attack goals. VMCP is based on the following two premises:

1. An adversary will not attack the system without assets of interest.
2. An adversary cannot attack a system without entry points, interfaces the system has with the outside world

By identifying all assets of interest and entry points, adversaries gather the basic information for undertaking attacks, what are the attack targets, and where they can enter the system to reach the assets. Then the next step is to figure out where are the most possible places they can break the system. These usually are the trust boundaries, where trust levels change. We have modeled Credit Card Payment Component from these three aspects: assets of interest, entry points and trust boundaries, to get enough information for later attack analysis.

Identified Assets

ID	Name	Description	Trust Roles	Catalogue
A1	Credit Card Information	The cardholder, card number and expire date, stored in memory.	Invoker(R)	Application data
A2	Order	Order information stored in memory.	Invoker(R)	Application data
A3	Configuration Data	Configuration information stored in back-end database.	Invoker(CRUD)	Application data
A4	Geographic Zones	Geographic Zones and countries.	Invoker(R)	Application data
A5	Process	Processes running within the same machine where the component is running	N/A	System resource
A6	Physical Machine asset	Assets of the environment where the component is running, e.g. files stored in the file system.	N/A	System resource

A7	Ability to trace and audit actions occurred	Ability to trace hacker's exploit action and audit what users have done.	N/A	Non-Functionality
A8	Availability of service	Ability to keep the service available to users during a certain period.	N/A	Non-Functionality

C: Create, **R**: Retrieve, **U**: Update, **D**: Delete.

Identified entry point

ID	Name	Description	Trust Roles	Catalogue
EP1	update_status()	Determine if the credit card payment is enabled.	Invoker	Component interface
EP2	javascript_validation()	Return a snippet of input validation javascript program.	Invoker	Component interface
EP3	selection()	Return Forms which accept user's input	Invoker	Component interface
EP4	pre_confirmation_check()	Validate the credit card accepted.	Invoker	Component interface
EP5	confirmation()	Return masked credit card information	Invoker	Component interface
EP6	process_button()	Return Forms with hidden fields of credit card information.	Invoker	Component interface
EP7	before_process()	Store card number to order	Invoker	Component interface
EP8	after_process()	Email order information to customer.	Invoker	Component interface
EP9	install()	Store configuration information to the configuration table.	Invoker	Component interface System interface
EP10	remove()	Remove configuration information from the configuration table.	Invoker	Component interface

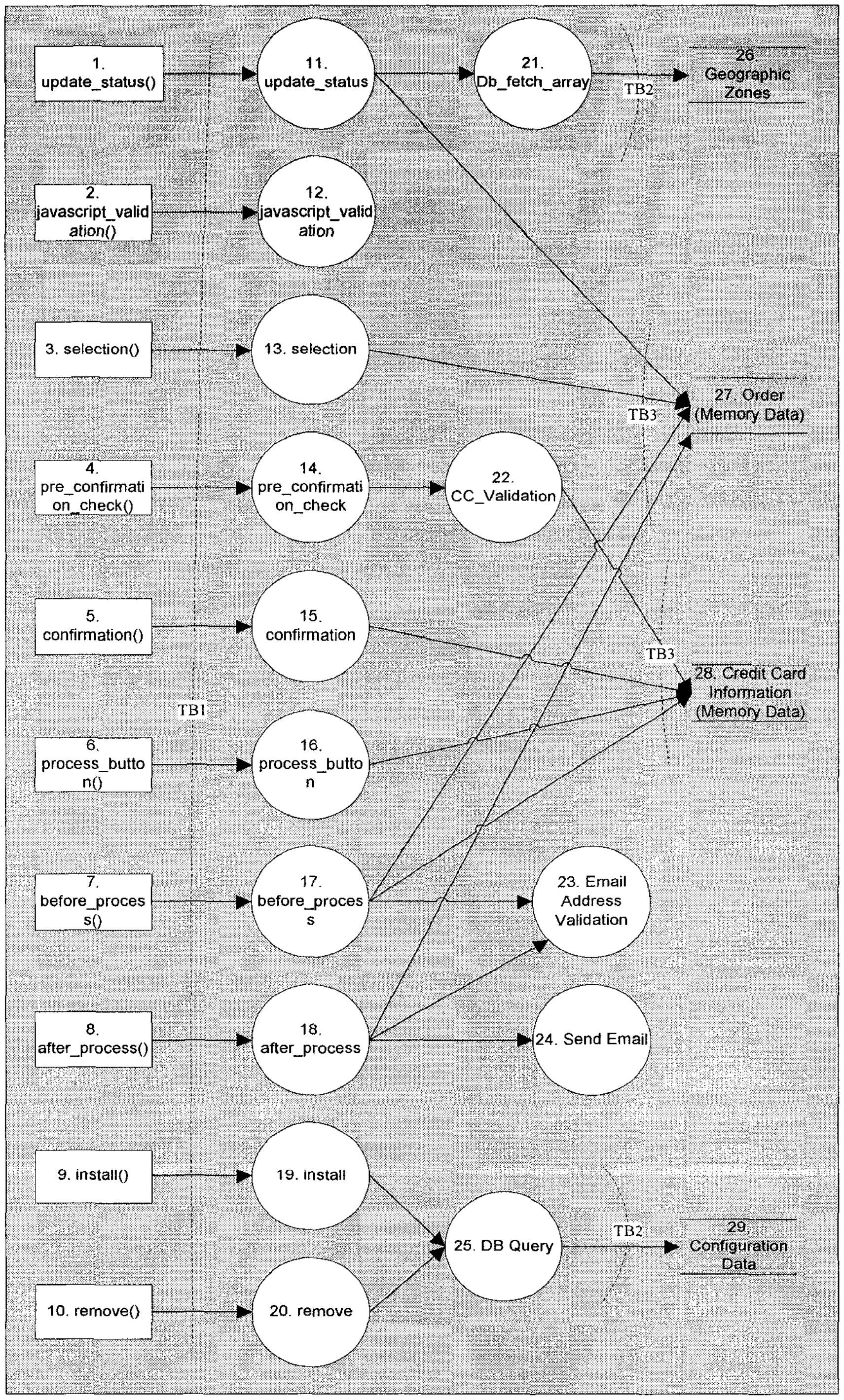
Identified Trust Boundary

ID	Name	Description	Category
TB1	API Boundary	The boundary between outside invokers and the component.	Service Boundary
TB2	Database Boundary	The boundary between the component and database.	System Boundary
TB3	Memory Data Boundary	The boundary between processes and Memory Data	System Boundary

Model attack paths and security criteria

Using the theory and method presented in chapter 7, we have modeled the DFD of Credit Card Payment Component, attack paths and security criteria.

Figure C-5-1 DFD of Credit Card Payment Component Integrated with Trust Boundaries



Attack paths and security criteria
(Specail case: 2→12 is not a attach paths as no asset is connected to them)

Path ID	P1
Description	1→11→21→26 Invoker calls update_status() to check the status of credit card payment. update_status() read geographic zones information from database.
Passenger	Invoker
Action on Asset	Geographic Zones (R)
Rules	N/A

Path ID	P2
Description	1→11→27 Invoker calls update_status() to check the status of credit card payment. update_status() read Order information from memory.
Passenger	Invoker
Action on Asset	Order (R)
Rules	No other memory data can be read out

Path ID	P3
Description	3→13→27 Invoker calls selection() to get forms to accept user's input. selection() read Order information from memory to build forms. .

Passenger	Invoker
Action on Asset	Order (R)
Rules	No other memory data can be read out

Path ID	P4
Description	Invoker calls pre_confirmation_check() to validate the credit card accepted. Credit card information is read from memory.
Passenger	Invoker
Action on Asset	Credit card information (R)
Rules	No other memory data can be read out

Path ID	P5
Description	Invoker calls confirmation() to get masked credit card information.
Passenger	Invoker
Action on Asset	Credit card information (R)
Rules	No other memory data can be read out

Path ID	P6
Description	6→16→28 Invoker calls process_button() to get forms with hidden fields of credit card information. process_button() read credit card information from memory.

Passenger	Invoker
Action on Asset	Credit card information (R)
Rules	No other memory data can be read out

Path ID	P7
Description	7→17→27 Invokers calls before_process() to store card number to Order. before_process() writes card number to Order.
Passenger	Invoker
Action	Order (U)
Rules	No other memory data can be read out

Path ID	P8
Description	7→17→28 Invokers calls before_process() to store card number to Order. before_process() reads card number from memory.
Passenger	Invoker
Action	Credit Card Information (R)
Rules	No other memory data can be read out

Path ID	P9
Description	8→18→27 Invoker calls after_process() to email order information to

	customer.
Passenger	Invoker
Action	Order (R)
Rules	No other memory data can be read out

Path ID	P10
Description	9→19→25→29 Invokers calls install() to store configuration information to the configuration table.
Passenger	Invoker
Action	Configuration Data (U)
Rules	Can not overwrite the configuration information of other components.

Path ID	P11
Description	10→20→25→29 Invokers calls remove() to remove configuration information from the configuration table.
Passenger	Invoker
Action	Configuration Data (D)
Rules	Can not remove the configuration information of other components.

Attack analysis

In accordance with the attack analysis methodology described in chapter 8, we choose path P1 and P4 as candidates for detailed attack analysis because:

- Their passenger is Invoker who have the highest privilege.
- Both paths have only 'READ' access to the asset.
- They cover all three trust boundaries.
- They cover all two catalogues asset.

Possible attacks along path P1 and P4

ID	A1	Name	Buffer overflow
Description	Buffer overflow and Integer overflow.		
STRIDE	T. D. E		

ID	A2	Name	SQL injection
Description	SQL injection occurs, enabling an attacker to exploit an input validation vulnerability to execute commands in the database and thereby access and/or modify data.		
STRIDE	T. I. E		

ID	A3	Name	Cross-Site Scripting
Description	Cross-site scripting occurs when an attacker succeeds in injecting script code.		
STRIDE	T. I. D. E		

ID	A4	Name	Information Disclosure
Description	Information is disclosed and sensitive exception details are revealed to the client.		
STRIDE	I		

ID	A5	Name	Server Attack
Description	An attacker manages to take control of the servers, gain unauthorized access to the database, and run commands against the database.		

STRIDE	T. D. E

ID	A6	Name	Configuration Attack
Description	Retrieval of clear text configuration secrets		
STRIDE	T. D		

ID	A7	Name	Individual Accountability
Description	Lack of individual accountability		
STRIDE	R		

ID	A8	Name	Session Attack
Description	Session hijacking and replaying.		
STRIDE	S. T. E		

ID	A9	Name	Form Manipulation
Description	Query string and form field manipulation		
STRIDE	T. E		

Vulnerabilities identified and rates

Use the methodologies presented in chapter 9 and chapter 10 we have identified and rated the flowing vulnerabilities:

Description	No maximum length check for the credit card number and cardholder
Rate	

DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	5	5	15	15	11

Description	Clear text configuration stored on database				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
15	5	5	15	5	7

Description	No validation on parameters passed from Order and these parameters are used to generate SQL query.				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
10	15	15	10	5	11

Description	Exposing exception details to the client				
Rate					
DAMAGE	REPRODUCABILITY	EXPLOITABILITY	AFFECTED USERS	DISCOVERABILITY	OVERALL
5	5	5	15	15	7

Certification

Based on the rated vulnerabilities, security level of Credit Card Payment Component is certified as moderate, as two of its vulnerabilities are rated 8-11.

Appendix D: Security Attacks & Vulnerabilities Taxonomy

This taxonomy is developed only for the purpose of demonstrating VMCP and the three sample applications, since developing a common taxonomy of security attacks and vulnerabilities is one of the steps of VMCP. At this stage I classified security attacks and vulnerabilities into nine groups.

10. **Input and data validation**
11. **Authentication**
12. **Authorization**
13. **Session Management**
14. **Insecure Data Storage**
15. **Insecure Configuration Management**
16. **Cryptography**
17. **Parameter Manipulation**
18. **Exception handling, Auditing and Logging**

Input and data validation

Overview

Input and data validation requires applications not to blindly trust any input or data before they pass the validation of the type, length, format, range or even the content. It is a must to validate the input or data before processing them. An attacker can compromise your application if any such vulnerability is identified. Applications that do not perform input and data validation are susceptible for following attacks.

- Buffer Overflow
- Cross-site scripting
- SQL injection
- Canonicalization
- Format string attacks

Buffer overflow

Buffer overflow, or buffer overrun, is a programming error that data is stored beyond the boundaries of a fixed-length buffer. It results in the extra data overwriting adjacent memory locations, which may contain other buffers, variables and program flow data. Buffer overflows may cause a process to crash or produce unexpected results. Moreover, if it is triggered by inputs specifically designed to execute malicious code (code injection) a breach of system security is possible. The following piece of C code demonstrates an example:

```
char small_buffer[10];
// declare buffer that is bigger than expected
char large_buffer[] = "This string is longer than 10 characters!!!";
strcpy(small_buffer, large_buffer); // overrun buffer !!!
```

Causing Vulnerabilities:

- Array bounds are not checked whenever an array is accessed
- Application's use of unmanaged code
- When unmanaged APIs are called in the application, the values passed for the parameters of unmanaged API are not checked.

Cross-site scripting

Cross-site scripting commonly referred as XSS allows code injection by malicious web users into the web pages viewed by other users. Attackers usually inject HTML code or client-side scripts into a vulnerable application to fool a user in order to gather data from them. The browser is not able to tell whether the code is legitimate or malicious since the script code is downloaded by itself from a trusted site. Hackers normally conduct Cross-site scripting by identifying the vulnerable page that outputs the invalidated input back to the browser. The following snippet of code shows a XSS attack.

```
Excerpt from script.php:
        echo $HTTP_GET_VARS["input"];

HTTP request:
http://www.xxx.com/script.php?input=%3cscript%20src=%22http%3a%2f%2fwww.
myserver.com%2fbadscript.js%22%3e%3c%2fscript%3e

Generated HTML:
        <script src="http://www.myserver.com/badscript.js"></script>
```

Causing Vulnerabilities:

- Thorough input validation is not performed on form fields, query strings, or cookies, especially for scripting tags and filters.
- User inputs are not encoded using HTMLEncode and URLEncode functions.

SQL injection

SQL injection is an attack that exploits a security vulnerability occurring in the database layer of an application which constructs dynamic SQL statements based on the user input or executes a stored procedure with arguments based on the user's input. The vulnerability is caused by the string literal escape characters embedded in SQL statements, which should be correctly filtered by the validation process. The damage caused by SQL injection is based on the privilege of the account under which the SQL command is being executed. Following snippet of code shows how this attack can be exploited.

Original database query in lookupuser.asp:

```
sql = "SELECT lname, fname, phone FROM usertable WHERE lname='" &
Request.QueryString("lname") & "';"
```

HTTP request:

```
http://www.xxx.com/lookupuser.asp?lname=buffett%27%3bupdate%20usertable%20s
et%20passwd%3d%27null%27%3b--%00
```

Executed database query:

```
SELECT lname, fname, phone FROM usertable WHERE lname='buffett';update
usertable set passwd='null';
```

- Execute stored procedures using arguments based on the user input
- Privileges to execute the SQL commands is not set appropriately

Canonicalization

Canonicalization deals with the way in which applications convert data that has more than one possible representation into a "standard", simplest canonical representation. Web applications have to deal with lots of canonicalization issues from URL encoding to IP address translation. If an application makes security decisions according to canonical forms of data, it is fragile to canonicalization attacks.

Directory traversal is a typical example of a canonicalization issue. For example, a web server may have a security rule of "only execute files under the bin directory (C:\myproject\root\bin)". The rule is implemented by checking that the path starts with "C:\myproject\root\bin\", and if it does, the file is executed. Under this rule "C:\myproject\root\bin\..\..\..\Windows\System32\cmd.exe" will be treated as a legal file and be executed. Can you image what will happen?

Causing Vulnerabilities:

- Accept file name as input. When there is a need for accepting input to grant access, do not convert the name to canonical form prior providing security decisions
- Do not assume well formed filenames are received and check whether they are within your application' directory hierarchy
- Use input file names, URLs, or user names for security decisions
- Do not ensure that the character encoding is set correctly to limit how input can be represented.

Format string attacks
Format string attacks, discovered around 1999, can be used to crash a program or to execute malicious code. The problem originates from certain C functions that do formatting, such as printf(). The format string parameter of these routines is not validated. Data from the stack or possibly other locations in memory can be easily printed out by using the %s and %x format tokens. The %n format token can command printf() and other similar functions to write arbitrary data to arbitrary locations.

Format string bugs are usually caused by the laziness of programmers. When a programmer wishes to print a string containing user inputs, they mean to write something like:

```
printf("%s", str);
```

but instead they decide to save time by writing:

```
printf(str);
```

The first version simply prints a string to the screen, as the programmer intended but the second version interprets buffer as a format string, and parses any formatting instructions it may contain. If a hacker feeds the program with a string containing special format tokens, damage will be caused.

Causing Vulnerabilities:

- Do not filter out format tokens in user's input
- Use incorrect format of functions

Authentication

Overview

Authentication addresses the question: who are you? It is the process of uniquely identifying the clients of your applications and services by validating the user with whom they claims to be. This is typically achieved through credentials, such as a user name and password. These clients may be end users, other services, processes, or servers. Following are the possible attacks that an attacker can conduct to exploit failures in an application.

- Brute force attacks
- Dictionary attacks
- Cookie replay attacks
- Credential theft

Brute force attacks

Brute force in computer science refers to a method of finding a solution by trying all permutations of a problem, in contrast to the implementation of a more intelligent algorithm. When performing brute force attacks hackers rely on the computational power such as an automated process of trial and error to figure out the hash string and encryption technique used for securing the sensitive information data like passwords.

Causing Vulnerabilities:

- A weak hash key strings is used
- Using weak passwords mechanism like unlimited password retry times

Dictionary attacks

Dictionary attacks refer to trying "every word in the dictionary" as a possible password for an encrypted message. As people tend to choose poor passwords, a dictionary attack is more efficient than a brute force attack in general. If an application implements a poor authentication mechanism that does not force users to choose complex passwords, a dictionary attack may be possible.

Causing Vulnerabilities:

- Poor password mechanisms to allow the use of weak passwords that are not complex. Do not use mixture of uppercase, lowercase, numerals, and special characters in the password that makes difficult to crack.
- Store only reversible password hashes in the user store.

Cookie replay attacks

When authentication information is stored in a cookie, an attacker can read authentication information that is submitted for the application to gain access if they gain access to that cookie. Then by replaying the same information to the application the attacker can authenticate back to the web application.

Causing Vulnerabilities:

- Cookie information passed through the channel is in plain text form or weakly encrypted
- Timeout property for the cookie information is set incorrectly. This will increase the probability of attack.

Credential theft

Credential theft attacks in web application refer to using the credential information stored by the browser to gain access to a web application. If setup incorrectly browser history and cache may store user login information for future use. In this case if someone gets access to a terminal that is logged on by others and hits the same page, the saved login will be available.

Causing Vulnerabilities:

- Use weak passwords
- Do not store password verifiers with one way hash with added salt
- Do not enforce account lockout for end-users after a set number of retry attempts
- Do not set the expiry property for the content rendered in the browser or allow the browser to cache the information

Authorization

Overview:
Authorization addresses the question: what can you do? It is all about granting or denying access to the resources and operations for which the authenticated client requests access. Authorization is usually accomplished based upon user identity and role membership. Resources include files, databases, tables, rows, and so on, together with system-level resources such as registry keys and memory data. Operations include performing transactions such as purchasing a product, depositing and withdrawing money from one account, or updating user profile. Top attacks that exploit authorization are

- Elevation of privilege
- Disclosure of confidential data
- Luring attacks

Elevation of privilege
Elevation of privilege refers to a process or an attack by which a malicious user try to become a member of the group with higher privilege than that for which he has been authorized. This kind of attack could enable hackers to compromise or destroy a system, or to access unauthorized information.

Causing Vulnerabilities:
- Bad design in the application, not ensuring that application gains access only to least privileged process, services and user accounts.

Disclosure of confidential data
Disclosure of confidential data refers to unauthorized users gaining access to sensitive data. Applications should always store confidential data securely in the persistent store like databases, XML files and other configuration files. Particular attention is required when confidential data is transmitted through the network and displayed to users. Proper access control is needed to ensure the right person gets the right information.

Causing Vulnerabilities:
- A poor role check before providing access to sensitive data
- A poor access control mechanism is used.
- Persistent stores like database and configuration files do not stored the sensitive information in the encrypted form

Luring attacks
A luring attack occurs when an attacker lures a component with more privileges to perform something on his behalf. Luring attacks are a particular case of the elevation of privilege attack. It is normally performed by convincing the target to

run the attacker's code in a more privileged security context.

Causing Vulnerabilities:

- Do not restrict access to trusted code with appropriate authorization.

Session Management

Overview:
In order to provide a friendly environment to the users, web-based applications often use sessions to maintain states through user's subsequent requests. Sessions are stored on servers and linked to users by session IDs. Session IDs are an attractive target for hackers as they can act as the associated users once they get their session ID. Moreover, sometimes applications store sensitive information in the session objects managed by the application layer. The attractive session ID and sensitive information stored in the session objects lead to potential attacks. They include:

- Session hijacking
- Session fixating
- Session forging
- Session replay

Session Hijacking

Session Hijacking refers to the exploitation of a valid session ID to gain unauthorized access to remote server. Many web-based application use session cookies to confirm users identify so that users do not have to re-enter their username and password on every page. If an attacker is able to steal this "magic cookie", an attacker can spoof the user session and gain access to the system. The he can perform all the operations as that of the legitimate user.

Causing Vulnerabilities:

- Log out method is not provided or logging out does not clear all session state and remove or invalidate any residual cookies.
- Improper expiry times on persistent cookies.
- Store session tokens in the URL or other trivially modified data entry point.
- Store sensitive data in the session objects.
- The authentication token stored in the cookie is transmitted in plain text.
- Allow not only one session per user at a time. If a new session is started for the same user, do not implement logout functionality.

Session fixating

Session fixating refers to exploitation of the vulnerability that allows one person to fixate another person's session ID. Normally the attacker issues a session ID to the user's browser before the user even logs into the target server, thereby forcing the browser into using a fixated ID and eliminating the need to obtain the session ID afterwards.

Causing Vulnerabilities:

- Accept session identifiers from GET / POST variables.
- Do not regenerate SID on each request.

- Session destruction, either due to logging out or timeout, takes place just on the browser, not on the server too.
- Do not bind the session ID to the user's SSL client certificate.
- Do not verify that additional information is consistent throughout session.

Session forging
Session forging acts the same way as brute force attacks. Although many websites have implemented strategies to prohibit brute force attacks against passwords, there is not much done against session forging, which allows an attacker try hundreds or thousands of session tokens embedded in a legitimate URL or cookie. Session forging may result in elevated privileges and DoS.

Causing Vulnerabilities:

- Improper session management strategy
- Accept session identifiers from GET / POST variables.
- No detection mechanism is implemented to keep users from trying to manipulate their token to gain elevated privileges

Session replay
Session replay is masquerading as an authorized user on an interactive Web site. If an attacker steals the authentication token stored in a cookie, they gain access and the ability to do anything the authorized user can do on the Web site. Session replay attack is difficult to detect because it does not occur in real time. It may only be discovered when the real user learns he has been the victim of identity theft or some other form of fraud.

Causing Vulnerabilities:

- Use persistent cookies to store the session token
- Store authentication information on the client
- When a critical function is being called or an operation is performed, do not re-authenticate the user
- No proper session token timeouts and token regeneration is setup to reduce the window of opportunity to replay tokens.

Insecure Data Storage

Overview:
There is a misunderstanding that if the encryption is strong enough no sensitive data will be stolen. However encryption may be totally compromised by a single vulnerability. This answers the question why devastating thefts of sensitive data continue to occur even though enterprises worldwide spent approximately $20 billion per year on IT security. Sensitive data is always at great risk as it is always the target of malicious attacks. Most of the security cost and effort are usually spent on protecting sensitive data. Common attacks regarding data storage are:

- Unauthorized access to data in storage
- Unauthorized access to data in memory
- Network eavesdropping
- Data tampering

Unauthorized access to data in storage
Most web applications have a need to store sensitive information in persistent data stores. Access control mechanisms should be utilized to prevent external attackers, or even internal employees, from obtaining illegal access to the data store. When dictionaries containing data files are not isolated from web access or the database does not have its own authentication and authorization protection, the possibility for an attack to be successful can be very high.

Causing Vulnerabilities:
- Do not minimize the use of encryption and only keep information that is absolutely necessary
- Poor access control management to the information stored in data storage
- Store the sensitive data in a plain text format
- Incorrect implementation of cryptography
- Poor choice of algorithm
- Do not differentiate view and modify operations separately and provide access accordingly.

Unauthorized access to data in memory
One of the best practices regarding application security is only keeping information that is absolutely necessary. For example, rather than encrypting and storing credit card numbers onto persistent storage, simply ask users to re-enter the numbers each time. This has brought out another security issue: sensitive memory data. Although stealing memory data is much harder than stealing data stored on the persistent storage, it is still not wise to ignore protecting memory data.

Causing Vulnerabilities:
- Improper storage of secrets in memory

- Poor sources of randomness
- Keep sensitive date in memory for an unnecessary long period
- Keep secret in codes.

Network Eavesdropping
Network eavesdropping refers to the interception of network data sent from browser to the server or vice versa. By using network monitoring tools, attackers can capture the information transferred on the network and can even modify the information and send it back onto the network.

Causing Vulnerabilities:

- Passing sensitive data over the network in plain text format.
- Do not encrypt the communication channel by implementing SSL when required

Data tampering

Data tampering refers to unauthorized modification of data. This is another major concern regarding sensitive data. When talking about security we first think of external hackers but it is reported that 70 percent of data risks may come from internal users. Anti-tampering measures are critically important to safeguarding data against inside threats.

Causing Vulnerabilities:

- Use no-tamper-resistant protocols such as hashed message authentication codes.
- Use poor role-based security mechanism to differentiate between users who can view data and users who can modify data.

Insecure Configuration Management

Overview:
Today web applications frequently use services provided by the application server and/or web server such as data storage, directory services, mail and so on. However the component development group (provider) is separate from the group using the component (consumer). Very often a wide gap between those who write the component and those responsible for the operations environment (consumers) is created by the improper assumptions made by the writers that how consumers will configure their server. Web application security concerns often span this gap. In addition, Most of the web applications are configurable and store the configuration parameters in files or databases. To facility management of configuration, applications normally provide configuration management interfaces to allow users with high privileges, say administrators, to change configuration parameters and perform maintenance. This makes the situation even worse. The following are common attacks due to insecure configuration management.

- Unauthorized access to configuration management interfaces
- Unauthorized access to configuration stores
- Retrieval of plain text configuration secrets

Unauthorized access to configuration management interfaces

Actually this is a specific case of authorization. The reason we discuss it here separately is that most of the web applications have configuration management interfaces and thus have some vulnerabilities in common. Generally configuration management interfaces should be available only to the restricted group. An attacker who obtains access to the configuration management interfaces can easily bring down the system or let it behavior unexpectedly by altering the configuration parameters.

Causing Vulnerabilities:
- Unnecessary administration interfaces are used.
- Unnecessary services enabled, including content management and remote administration.
- Misconfigured SSL certificates and encryption settings.
- Use of self-signed certificates to achieve authentication and man-in-the-middle protection.
- Use of default certificates.
- Improper authentication with external systems.
- Easy-guessed default accounts with their default passwords

Unauthorized access to configuration stores

This is another particular case of authorization. An attacker who obtains access to the configuration stores can easily bring down the system or let it behavior strangely by altering the configuration data. And this will impact all users. Due to the sensitive nature of the data maintained in configuration stores, the stores should be kept secured.

Causing Vulnerabilities:

- Keep custom configuration files inside the directory that has web access.
- Overly informative error messages.
- Unnecessary default, backup, or sample files, including scripts, applications, configuration files, and web pages

Retrieval of clear text configuration secrets
Today most configure files are in plain text format such as a XML file and sensitive data such as database connection strings and passwords are stored also in plain text format in these configuration files. Attackers who gain access can see this sensitive information. Internal users, such as disgruntled employees and administrators, can misuse this sensitive information.

Causing Vulnerabilities:

- Storing the data in plain text format, rather than store the sensitive information in encrypted formats.
- Insecure access control policies on text based configuration files.

Cryptography

Overview:
Today most web-based applications use cryptography to protect sensitive information when transmitted and stored. Basically cryptographic systems can provide four services: authentication, non-repudiation, confidentiality and integrity. Cryptography is one of the most advanced topics of application security and there are many approaches to encryption, each with advantages and disadvantages. Very often expert experience is needed when architects and developers try to choose a cryptography approach and implement it correctly and accurately. A small mistake in configuration or coding may result in a useless cryptography. Typical cryptographic attacks are:

- Cryptographic key attacks

Cryptographic key attacks

Cryptographic keys should be adequately protected as cryptography relies on keys to assure a user's identity, provide confidentiality and integrity as well as non-repudiation. If an attacker gets access to the encryption key, they can easily decrypt the encrypted information. Poor management of keys or bad key generation algorithms usually results in this type of attack.

Causing Vulnerabilities:
- Do not use built-in encryption routines that include secure key management.
- Keys are not in binary format
- Keys are not protected with file system permissions.
- Store keys in a open location
- Do not expire keys regularly
- Distributing keys in an insecure manner

Parameter Manipulation

Overview:
Manipulating the data sent between the browser and the web application is a simple but effective way to change application behaviors. Information captured from the browser is usually sent to the server in one of these four formats: URL query string, form fields, cookies and HTTP headers. In a badly designed and developed web application, malicious users can modify data before it is be transmitted so even cryptographic protection in the transport layer (SSL) is insufficient. Parameter tampering can often be done with:

- URL Query String
- HTML Form field
- Cookie
- HTTP header

URL Query String manipulation

If HTML Forms submit their results using a method GET, all form element names and their values will appear in the query string of the next URL the user sees. Tampering with query strings is very easy;. one need only look at the URL in the browser's address bar and change the values. If the application relies on the query string values to make security decisions, the application is vulnerable to security attacks.

Causing Vulnerabilities:

- Use HTTP GET rather than using HTTP POST.
- Pass security related information through query string
- Information passed via query string is not cryptographically protected.
- Information passed via query string is not accompanied by a valid session token.

HTML Form field manipulation

In most web-based applications, developers use Form Fields as a convenient way to store data in the browser. However, attackers can easily manipulate these form fields no matter whether they are pre-selected (drop down, check boxes etc.), free form or hidden. In most cases what an attack has to do is simply save the page using "view source", "save", edit the HTML and re-load the page in the web browser.

Causing Vulnerabilities:

- Make security decisions according to the value of Form parameters.
- No additional hidden field (e.g. Outgoing Form Digest) is used to protect the value of a critical Form Field parameter.
- Use meaningful parameter names.

Cookie Manipulation

Cookie manipulation refers to altering cookie values on the client's web browser to exploit security issues within a web application. Cookies are usually used to maintain states in the stateless HTTP protocol, as well as to store user preferences and other data including session tokens. Many people think that non-persistent cookies cannot be modified but the fact is that both persistent and non-persistent cookies, secure or insecure can be modified by the client and sent to the server with URL requests. The extent of cookie manipulation depends on what the cookie is used for. Here is an example from a real world example on a travel web site.

```
Cookie: lang=en-us; ADMIN=no; y=1 ; time=10:30GMT ;
```

The attacker can simply modify the cookie to;

```
Cookie: lang=en-us; ADMIN=yes; y=1 ; time=12:30GMT ;
```

Causing Vulnerabilities:

- Use multi-session token to reference properties stored in a server-side cache.
- Do not encrypt the cookie to prevent tampering.
- Depend only on cookie values to make security decisions.
- In case of persistent cookies stored in the client computer, do not use encryption or hashing to protect the information.

HTTP Header manipulation

In web-based applications, the HTTP protocol is used to transfer data between the browser and the server. Most web application developers do not pay any attention to the HTTP headers as they often are used by the browser and the web server software only. However some web developers choose to make security decisions by inspecting incoming headers. When doing this most of the developers do not realize that the requested headers originate at the client side, and they may thus be altered by an attacker.

Causing Vulnerabilities:

- Use HTTP Headers to make security decisions
- Do not cryptographically protected headers originated from the server-side.

Exception handling, Auditing and Logging

Overview:
Exception handling, auditing and logging are three different aspects of the same topic: how to track events within an application. Applications should always fail safe. When an application fails to an unknown state, the exception information shown might not be making sense for the end user but might be a very interesting message for an attacker. Motivated attackers may be able to exploit this indeterminate state to access unauthorized functionality, or worse manipulate data. Well-written applications enable auditing and logging to easily track or identify potential fraud or anomalies end-to-end. This helps to identify which user is trying to exploit and what actions have been done. With this kind of information necessary actions can be taken to prevent the system from such attacks. The following attacks are related to this area.

- Detailed error message attacks
- Repudiation
- Escape from being traced
- Cover tracks

Detailed error message attacks

Detailed error messages provide attackers with lots of useful information as they might leak information that leads to further attacks, or may leak privacy related information. One of the most common situations is that database exceptions reveal SQL information like tables, connection strings, column names, etc. that will open a door for an attacker to enter into the application.

Causing Vulnerabilities:

- Use functional error handling instead of structured exception handler
- Do not handle all types of exception through out the code base.
- Show inappropriate information in the front end to the user who received this exception
- Detailed error messages, such as stack traces or leaking privacy related information, are presented to the user.

Repudiation

Repudiation refers to a user denies that s/he has performed an action or initiated a transaction. To address the issue of repudiation, developers have to define and implement a good defense mechanism to ensure that all the user activity can be tracked and recorded.

Causing Vulnerabilities:

- Do not enable auditing and logging in web server, database server and application server.
- Fail to identify the key events and log them. For example, login, logout events

- Use shared accounts

Escape from being traced
Attackers can escape from being traced if detection mechanism is not implemented correctly to identify the suspicious activities that have occurred in the system. A good detection mechanism should be able to log both the occurrence of exploit and whether someone is trying to exploit.

Causing Vulnerabilities:
- Not all critical application level operations is logged
- Poor detect suspicious activity.
- Do not maintain the back up of log files

Cover tracks
Attackers may be able to exploit and tamper log files if they are not well protected. Log files are as important as other sensitive files regarding application security, so they needed to be well protected from external hackers and internal employees.

Causing Vulnerabilities:
- Keep the log files in the default location folder.
- Do not secure log files