Software Testing Conference - 2005

Security Testing for Web Applications

Vivek Devarajan

Wipro Technologies
No. 72, Electronics City,
Hosur Main Road,
Bangalore – 560100
Karnataka, India
(+91-080) 30292191
vivek.devarajan@wipro.com

Abstract

Security is a very critical aspect of Web Applications, keeping in mind the alarming rise in cyber crimes and intrusion attempts by malicious users. The financial losses and legal consequences that organizations can face due to security incidents are immense.

This paper addresses some of the common maladies faced by QA Personnel and Customers from a Security Perspective. It describes the common Security vulnerabilities in applications with examples and techniques for testing the application for those vulnerabilities. Modelling techniques are discussed that help in identifying vulnerabilities and evaluate the Security Risks for an application. The paper suggests Processes, Framework, and Tools to help in providing highly effective and low cost Security Testing Solutions (specifically Penetration Testing) for Web Applications. Embedded in the paper are Traceability Templates, a Test Plan Template, a Checklist and a Process flow for Security Testing.

The objective is to enable detection of Security loopholes in applications at an early stage, to help in protecting applications and data from malicious users and to ensure information availability and integrity. Current trends in the IT Security Industry and budgeting guidelines are discussed, which will help organizations in making budgeting decisions for Application Security. Security breaches are discussed with impact and examples.

1. Introduction

Businesses today are under increasing pressure to be more accountable: accountable to investors, accountable to government, and accountable to the market (customers). Business accountability now includes security – specifically, software security.

Industry experts estimate that 70% of intrusion attempts are made at the application level. Reactive approaches such as patching are becoming increasingly ineffective with the growth of zero-day attacks (attacks that take advantage of software vulnerabilities for which there are no fixes) and the shrinking half-life of vulnerabilities (the time between when a vulnerability is released and when it is exploited).

The costs of application security are significant – and escalating. The U.S. Department of Commerce's National Institute of Standards and Technology reports that software flaws each year cost the U.S. economy $59.6 billion, including the cost of attacks on flawed code. The report also confirmed that remediation of software security problems after the occurrence of security incidents is expensive and time-consuming – as much as six times the cost of fixing them during the coding/testing stage.

Web Applications – Attacks on Port 80: Web Applications are being increasingly used today by organizations to conduct business online. Many of these applications utilize HTTP through Port 80 and expose the organization to security threats from crafted requests, hostile mobile code, and inappropriate web content.

Port 80 is used by all HTTP traffic and therefore always has to be left open. An attacker can pass a specifically crafted — but legitimate — HTTP message through the firewall to a Web server, exposing its vulnerabilities. Attacks buried in these messages sail past firewalls, filters, platform hardening, SSL, and IDS without notice because they are inside legal HTTP requests. The HTTP message can then exploit one or more of these vulnerabilities and cause a chain of events that ultimately allows the intruder to obtain privileged access to the Web server machine.

The vast majority of attacks, nearly 80 percent, are launched on port 80, the same port that Web traffic flows on. This poses a particular problem because curtailing access to port 80 would also negatively affect productivity and access to Web traffic.

2. The Scenario - Current Trends and Security Breaches

2.1 Current Trends in IT Security Industry

The Computer Security Institute (CSI) conducts the "*Computer Crime and Security Survey* "regularly. The survey covers the top 500 US Corporations, government agencies, financial institutions, medical institutions and universities. The following are some of the highlights of a recent survey by CSI: -

- Budget Allocation - Up to 5% of IT Budget spent for IT Security
- An increased investment in IT Security resulting in decreased Financial Losses
- Losses of $142 Million reported amongst 480 top US Companies due to Cyber Attacks
- Insurance - More Organizations (currently 30%) opting for Cyber Security Insurance
- Audits & Training - More Organizations (currently 80%) conducting Security Audits and Security Awareness Training

There is likely to be an increase in investments on IT Security initiatives, with many legal laws and acts coming into picture. Especially, the Sarbanes-Oxley Act (SOX), currently drawing much limelight, requires stringent Security Controls to be implemented on Applications dealing with financial data. Also, ensuring compliance with Security Standards such as BS7799 require significant investment in IT Security.

2.2 Security Breaches

A security breach of an application is the violation of security controls built in the application in order to gain unauthorized access to application resources.

Examples of Security Breaches: -

- Employee tampering with Payroll Data
- Airlines tricked into selling air travel tickets for a few dollars
- Exposure of Online Customers' credit card details
- Manipulation of Corporate Company's invoice/billing data
- Websites defaced with controversial content

Impact of Security Breaches: -

IT Staff – Wastage of Manpower: This happens due to additional efforts of IT Staff for fixing/patching up the security flaws and for tracking down culprits.

Non-availability of information: This happens when business critical information is tampered with or made inaccessible by attacking and crashing the application servers.

Legal Consequences: Many cyber security laws/acts make it mandatory for companies to protect their customers' confidential data. These laws hold the company accountable for any security breaches.

Increases in insurance premiums: With more IT companies opting for Cyber Security Insurance, repeated security breaches and losses could result in Insurance companies charging more premiums from organizations.

Loss of Public Image: Reported security breaches leads to negative publicity and loss of reputation.

Loss of Customer Trust: A large number of customer relationships are based on assurances of confidentiality of Customer Data. Such relations get affected when there is a breach in data confidentiality.

Competitor's access to information: Any confidential or proprietary information falling into the hands of competitors could place them in an advantageous position and give them the edge.

3. Web Application - Architecture and Vulnerabilities

3.1 Web Application Architecture

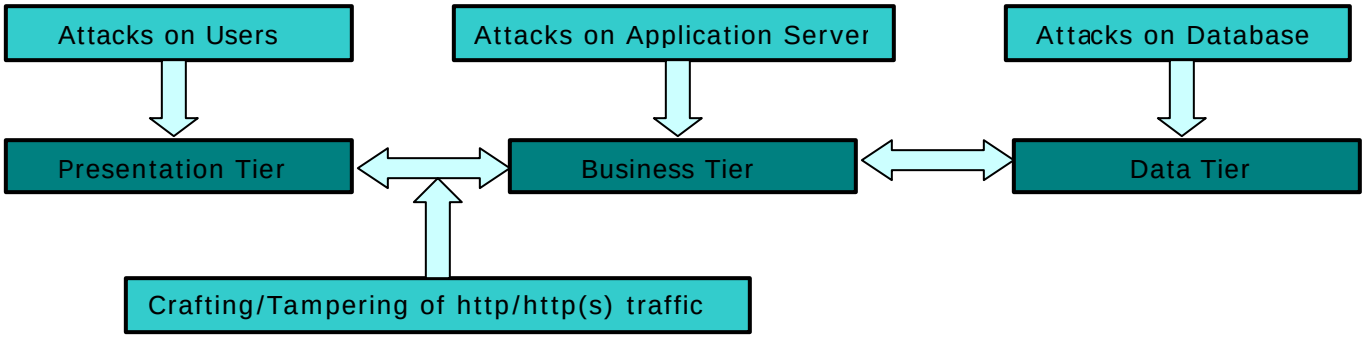


Figure 3.1 - Architecture of a Web Application

A Web Application typically consists of the following three tiers:

Presentation Tier: This tier implements the "look and feel" of an application. It is responsible for the presentation of data, receiving user events and controlling the user interface.
Business Tier: This tier implements the business logic of the applications. It is usually powered by an Application Server.
Data Tier: This tier is responsible for data storage and manages the persistence of application information. It is usually powered by a relational database server (Oracle or MS SQLServer) or User stores (such as LDAP).

An attack can be at launched at any of the three tiers. Some examples are SQL Injection attack (Data tier), Denial of Service Attack (Business Tier) and Cross Site Scripting attack (Presentation Tier).

3.2 Detection of Common Vulnerabilities in Web Applications

An application vulnerability is a weak link in the application security controls that could be exploited by a user to gain unauthorized access to information or disrupt critical processing. This section describes the most common vulnerabilities found in Web Applications.

A). Un-validated Input:

Description: Web applications use input from HTTP requests to determine how to respond. Insufficient server side validation of these input requests results in attackers tampering with parts of the HTTP request to try and bypass the site's security mechanisms.

Practical example: Consider an online shopping website where various products can be purchased online. Here, the price of the item may be stored in a request parameter in order to make use of the value while navigating through the shopping cart and checkout pages. In such a case, the request parameter may be modified using a web proxy tool and hence, the item could be bought for a reduced price.

Testing for the Vulnerability:

- Submitting crafted http requests with unexpected values to the Application Using Penetration Test Tools (such as Web Proxies) and checking for the responses.
- Detailed code review with respect to server side validations of all HTTP request parameters

B). Broken Access Control:

Description: Access control is how a web application grants access to content and functions to some users and not to others. Many flaws in access control schemes are discovered and exploited by crafting malicious requests for functions or content that should not be granted. This enables the attacker to tamper with unauthorized content, performing unauthorized functions, or even take over site administration.

Practical Example: Consider the case of an online timesheet application capturing the efforts of consultants/contractors in a company. The consultant (who bills the company on an hourly basis) enters his daily

efforts online after which the efforts entered are reviewed and approved by the supervisor. After approval, the timesheet is frozen (non-editable). Subsequently only the project management would have the necessary permissions to modify the timesheet.
However if the access control is not implemented properly (for example, if the user access level is a part of the client request), the consultant could send in a crafted request for updating the timesheet by tampering with the access level parameter and subsequently, update the timesheet successfully.

Testing for the Vulnerability:

- Review of the Design Document for Access Control Policies
- Submitting crafted http requests for performing higher privileged operations on the Application Using Penetration Test Tools (such as Web Proxies) and checking for the responses
- Review the Server Configurations and the code that implements the Access Control mechanism

C). Broken Authentication and Session Management:

Description: Authentication and session management includes all aspects of handling user authentication and managing active sessions. User authentication on the web typically involves the use of a user-id and password. Unless all authentication credentials and session identifiers are protected with SSL at all times and protected against disclosure from other flaws, such as cross-site scripting, an attacker can hijack a user's session and assume their identity. Many solid authentication mechanisms can be undermined by flawed credential management functions, including password change, forgot password, account update, and other related functions.

Practical Example: Consider the case of an online financial website where various end users have individual accounts with features such as "forgot password", "reset password", etc that don't typically require a user to login. If the user answers the hint questions in the "forgot password" functionality, and then subsequently cancels the password retrieval operation by clicking on a "Cancel" button provided, that returns him to the home page. If the session has not been properly been cleaned up, the user's hint questions and answers (entered earlier) would still be present in the session and then could be exploited by another user, especially if the first user did not close the browser window, even after browsing various other websites subsequently. If the second user types in the URL for the page that occurs after the Hint Q & A page, he is able to change the password of the first user and subsequently gain full control of the first user's account.

Testing for the Vulnerability:

- Testing the Session keys in order to ensure that session identifiers are protected from hijacking
- Testing the session management mechanisms (session cleanup, data maintained in sessions).
- Verifying if the user's credentials are protected during transit (from the client to the server) as well as during storage (e.g. in database).
- Review of the Authentication and Session Management Code
- Conduct brute force attacks on authentication credentials and session tokens

D). Cross-Site Scripting (XSS) Flaws:

Description: This problem can occur when a website uses the input entered by the user as part of the output displayed elsewhere in the application. An attacker could send malicious code as input to a different end user, whose browser executes the code. The malicious code could steal cookies, session tokens, or other sensitive information retained by the browser, defacing of the website with undesirable content disclosure of end user files.

Practical Example: Consider the case of an online insurance application where the applicant enters his name (say John Smith), fills the other fields and submits the application. A message ("Processing Insurance Application for John Smith") is displayed, containing the applicant name that was keyed in earlier. A malicious user could inject a script in the "Applicant Name" field that can steal the cookie or the session ID of the user. Here, the intruder may need to have his own server to relay the malicious code to the user's browser. Subsequently, the end user's cookie or session ID gets stolen, which can be hijacked by the intruder.

Testing for the Vulnerability:

- Testing of various screens for the display of fields/data that are being fed by the user from other screens. Here, the tester feeds in malicious scripts through the input fields, using a penetration testing tool, if needed.

- Testing the validation (client side as well as server side) mechanisms of input fields for malicious characters.

E). Buffer Overflows:

Description: A buffer overflow typically occurs when huge chunks of data are embedded as part of the requests sent to application servers leading to the code writing more information into the buffer than the space it has allocated in the memory. This way, an attacker can cause the web application to execute arbitrary code – effectively taking over the machine or crashing the server.

Practical Example: Consider an example in which the user is required to enter a 10 digit phone number. Based on that assumption, the programmer might write the code to allocate a 15-character buffer to contain the returned input. But in case the user returns an input that is 5000 characters long, the allocated buffer would be too small to contain all the input. The remaining characters will run over important parts of the application and could cause the system to actually execute parts of the input as if they were legitimate parts of the application code. Carefully crafted inputs can execute arbitrary commands on the server, usually with high permissions.

Testing for the Vulnerability:

- Testing the application with arbitrarily large inputs from users, using penetration testing tools in order to bypass client side validations.
- Verifying if latest patches are applied for products (Ex. Application servers) based on their bug reports.
- Code review to check how the application handles large inputs.

F). Injection Flaws:

Description: When a web application passes information from an HTTP request as part of an external request (such as a database call), it needs to be carefully scrubbed; otherwise it gives an attacker the chance to execute malicious commands. For example, the attacker can embed malicious SQL commands into the content of a parameter that the application passes through to a database and trick the application into forwarding a malicious query to the database. Thus the attacker can tamper with database data or even obtain database admin privileges.

Practical Example: Consider an application with a login screen that accesses the login database to authenticate the user. If the database is searched based on the Login ID and the keyed in password, a non-zero result set indicates a valid password, and the user is authenticated successfully. Now, if a malicious user bypasses client side validations and "injects" his own SQL command into the UserID field (such as OR '1' = '1'), the condition is always true and hence all the records in the Login database will satisfy this condition. Hence, the user gets authenticated, irrespective of the password entered.

Testing for the Vulnerability:

- Testing the application with inputs from users containing SQLs or system commands using penetration testing tools in order to bypass client side validations.
- Checking the validation mechanisms of input fields that form part of external system commands

G). Improper Error Handling:

Background: Web applications frequently generate error conditions during normal operation. When detailed internal error messages such as stack traces, database dumps, and error codes are displayed to the user (attacker), they reveal implementation details that can provide attackers important clues on potential flaws in the site. This could lead to system crashes or consume significant resources, or also help the attacker execute directory traversal attacks depending on the information that is revealed by the error messages.

Practical Example: Consider the case of a user registering at a website and not entering some mandatory information required for registration. If the user bypasses the client side validation and no server side validation is done for the same, a "null pointer exception" could occur, depending on how the coding is done. If the exception is not properly handled, the stack trace could be displayed in the screen which typically throws up information about the internal package/folder structure and also the names of some files. This information provides a malicious user with useful hints about the server's internal structure, thus enabling him to execute directory traversal attacks and possibly download some sensitive files or the source code.

Testing for the Vulnerability:

- Testing the application with input data that can simulate internal error conditions
- Code review to check implementation and consistency of error handling mechanisms

H). Insecure Storage:

Description: Most web applications have a need to store sensitive information (such as passwords, credit card numbers), either in a database or on a file system somewhere. Some common mistakes include failure to encrypt critical data, insecure storage of keys, certificates, and passwords, poor sources of randomness, poor choice of algorithms, failure to include support for encryption key changes and other required maintenance procedures. This vulnerability could lead to exposure of sensitive data, such as passwords, SSN, Credit Card numbers.

Practical Example: Consider the case of a financial website using a weak encoding mechanism for storing credit card numbers or passwords. These data can be sniffed and decoded during transmission or obtained from the database by a person with access to the same. Hence sensitive data may get exposed.

Testing for the Vulnerability:

- To check tokens, session IDs, cookies and other credentials (using penetration testing tools such as web proxies) for randomness and encryption.
- Review the implementation of cryptographic functions
- Conduct brute force attacks on tokens, session identifiers, user credentials

I). Denial of Service (DoS):

Description: Web applications are particularly susceptible to denial of service attacks. Once an attacker can consume all of some required resource, he can prevent legitimate users from using the system. Some resources that are limited include bandwidth, database connections, disk storage, CPU, memory, threads, or application specific resources. All of these resources can be consumed by attacks that target them.

Practical Example: Consider a functionality (such as registration) which typically does not require authentication. An attacker can easily place a heavy load on the server by simulate multiple registration operations and by feeding in arbitrarily huge input data through the registration fields, thus placing further load on the server and also consuming database connections. This could cause the server to crash or slow down to a crawl.

Testing for the Vulnerability:

- Testing the behaviour of the system under heavy load using load testing tools. Particular focus needs to be given to operations that can be performed by unauthenticated users and testing from a single IP address.

J). Insecure Configuration Management:

Description: Some server configuration problems that can plague the security of a site are un-patched security flaws in server software, Server mis-configurations that permit directory listing and directory traversal attacks, unnecessary default, backup or sample files, improper file and directory permissions, Default accounts/passwords, mis-configured SSL certificates and encryption settings, use of default certificates. An attacker could conduct directory traversal attacks, download the source code, get access to confidential files and possibly get admin access to the server.

Practical example: Consider the case of an email server not configured properly for directory level permissions. This could allow an attacker to conduct directory traversal attack, thus gaining access to sensitive files/mail attachments of other users.

Testing for the vulnerability:

- Executing tests such as directory traversal attacks and malicious characters through the URL
- Checking the Server Configuration for all security mechanisms
- Checking the setup of roles, permissions, and accounts, including disabling all default accounts or changing their passwords
- Testing the Logging/Audit trailing processes

4. Web Application Security Testing - Framework, Modelling, Process, Tools

4.1 Framework

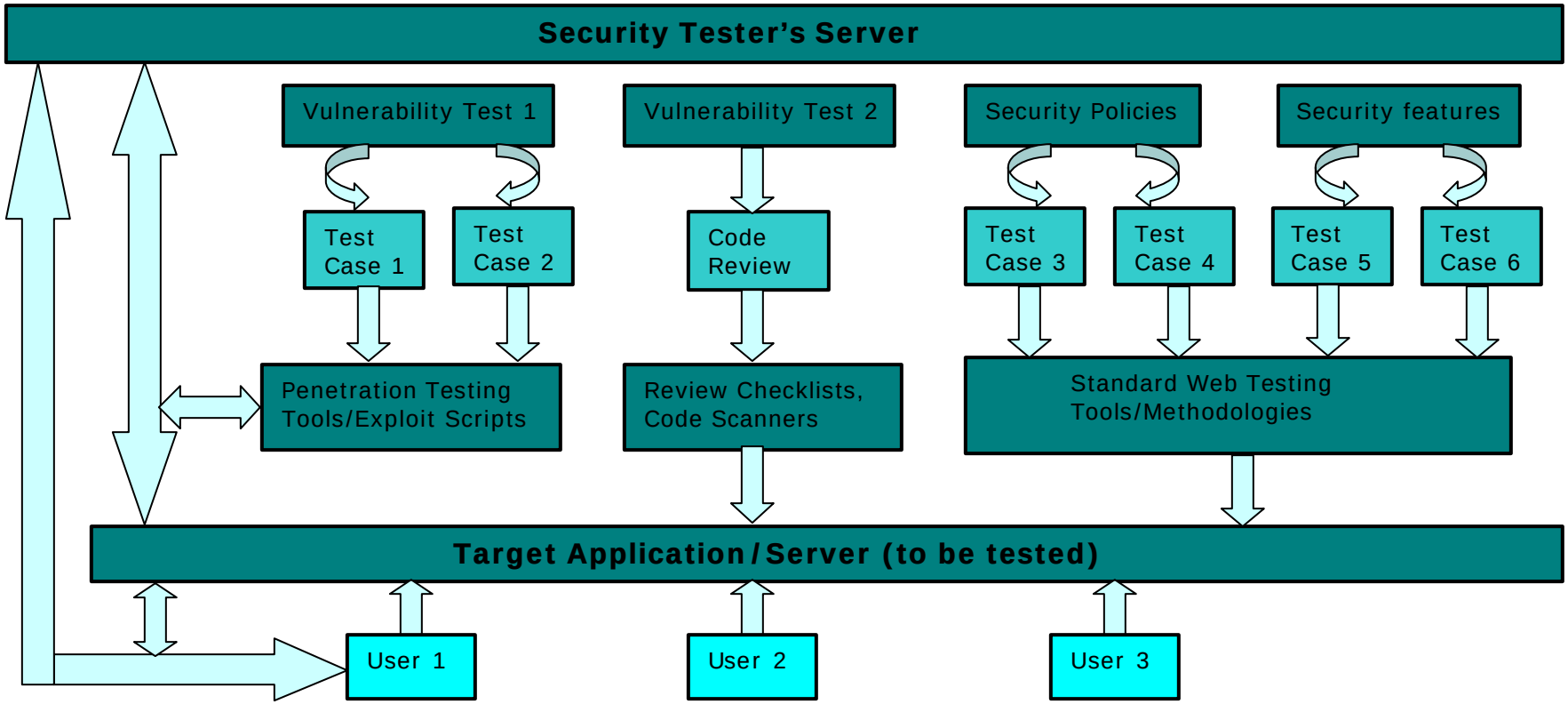


Figure 4.1 – Framework for Web Application Security Testing

Terminologies used in the Frame work:

Target Application/Server: This is the server that hosts the application to be tested.

Penetration Testing Tools: These tools aid in conducting penetration tests and simulating attacks on an application. Typical examples are Proxies, Web Spiders and Attack simulators that simulate attacks such as DoS and Buffer Overflows. These tools could be custom made, free ware or licensed.

Security Tester's Server: This is the server maintained by the Security tester or hacker to help in executing attacks and storing the stolen information from the Target application, server or user.

Exploit Scripts: These are test stubs/scripts written by the Tester and executed in the Tester's server or the Target Server. They could be hosted in the Tester's server or embedded in the Penetration Tools, or could be mobile code which could be executed through the client (browser).

Security Policies: These are the various policies framed to facilitate proper implementation of Security (either application specific or at an organization level). A typical example is "Password Policy". The implementation of these policies is tested using standard tools/methodologies that are employed for Web Testing.

Application Security features: These are the various features pertaining to Security in the target application. Typical examples are Authentication (such as login) and authorization (such as access control) features that are implemented as part of the application functionality and sometimes also configured in the application server. Also, other features such as encryption, Audit trail, etc could be implemented. Most of these features are tested using standard tools/methodologies that are employed for Web Testing. In some cases (such as encryption), appropriate stubs could be written.

Standard Web Testing Tools/Methodologies: These include commonly used tools for Web Testing, such as Browsers, Database Clients, and Automation tools (such as Winrunner).

Review Checklists: The review checklists ensure that the Application Code confirms to the recommended practices for Secure Coding.

Code Scanners: Code scanners are tools that scan the target application code for non-conformities to Secure Coding Practices.

4.2 Modelling Techniques

Various modelling techniques employed in Application Security Testing enable systematic identification and rating of threats based on understanding of the application's architecture and implementation details.

The following is a real world example that illustrates the terminologies used during modelling for Application Security:-
Consider a simple house analogy: An item of jewellery in a house is an **asset** and a burglar is an attacker executing the **attack** (theft). There is a **threat** from the burglar to the asset. A door is a feature of the house and an open door represents **vulnerability**. The burglar can exploit the open door to gain access to the house and steal the jewellery. In other words, the attacker exploits a vulnerability to gain access to an asset. The appropriate **countermeasure** in this case is to close and lock the door.
The terminologies are marked in bold in the above example and are defined in the appendix.

4.2.1Threat Modelling:

Threat modelling is an iterative approach to assessing the vulnerabilities in applications. It consists of the following steps:-

1. Asset Identification: An asset could range from confidential data, such as customer or orders database, SSN, Credit Card numbers to Web Pages or Web site availability.

2. Architecture Study: This involves a study of the Functionality, Architecture and Technologies used.

3. Application Decomposition: This step involves breaking down the application to create a security profile for the application based on traditional areas of vulnerability. Typically, trust boundaries, Data Flow, Entry points to the application, privileged code and mechanisms for Validation, Authorization and Authentication are identified and studied in this phase.

4. Threat Identification: This step involves identification of threats that might affect the system and compromise the assets. Threats are classified into three broad categories: 1). Network threats 2). Host threats 3).Application threats

5. Threat Documentation: This involves documentation of the information pertaining to the threat such as Threat target, Risk, Attack techniques and Countermeasures.

6. Threat Rating: In this step of the process, the threats are rated based on the risks they pose.
Risk = Probability * Damage Potential

4.2.2 DREAD Modelling:

The DREAD model can be used to help calculate risk. By using the DREAD model, we can arrive at the risk rating for a given threat by asking the following questions:
Damage potential: How great is the damage if the vulnerability is exploited?
Reproducibility: How easy is it to reproduce the attack?
Exploitability: How easy is it to launch an attack?
Affected users: As a rough percentage, how many users are affected?
Discoverability: How easy is it to find the vulnerability?

A simple threat rating scheme such as High (1), Medium (2), and Low (3) can be used.
The following table shows a typical example of a rating table that can be used when prioritizing threats.

	Rating	**High (3)**	**Medium (2)**	**Low (1)**
D	Damage potential	The attacker can subvert the security system; get full trust authorization; run as administrator; upload content.	Leaking sensitive information	Leaking trivial information
R	Reproducibility	The attack can be reproduced every time and does not require a timing window.	The attack can be reproduced, but only with a timing window and a particular race situation.	The attack is very difficult to reproduce, even with knowledge of the security hole.
E	Exploitability	A novice programmer could make the attack in a short time.	A skilled programmer could make the attack, and then repeat the steps.	The attack requires an extremely skilled person and in-depth knowledge every time to exploit.
A	Affected users	All users, default configuration, key customers	Some users, non-default configuration	Very small percentage of users, obscure feature; affects anonymous users
D	Discoverability	Published information explains the attack. The vulnerability is found in the most commonly used feature and is very noticeable.	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.	The bug is obscure, and it is unlikely that users will work out damage potential.

4.3 Processes - Verification and Validation (V&V) Process for Application Security Testing

This process is useful when conducting security tests for applications that are in the process of being developed from scratch. Here, the Security Testers are involved right from the initial phases of the project. This approach helps in early detection of vulnerabilities in applications, thereby reducing the cost involved in fixing them. Threat Modelling could be implemented during the "Requirement Study" phase of the process. A diagrammatic depiction of the V & V process is embedded in a word document herewith.

V & V Process.doc

Maintaining Requirement Traceability - a recommended practice:

A "Requirement Traceability Matrix" document serves as a link between the Security Requirements, Vulnerabilities and Test Cases. A "Feature v/s Vulnerability" traceability matrix gives the links between test case IDs, Vulnerabilities and Application Features. Traceability practices minimize the efforts required in modifying test cases when there are requirement changes.

Templates for "Feature v/s Vulnerability" matrix, Traceability Matrix and a Test Plan are attached herewith.
Also attached is a checklist for detection of security issues in the application and a diagrammatic depiction of the V & V process.

Requirements Traceability Matrix Te

Application Feature vs Vulnerability Matrix

Checklist - Detection of Security Issues.do

Test Plan Template.doc

4.4 Tools:

Some of the commonly used categories of tools for Security Testing for Web Applications are as follows:
Penetration testing tools: These tools enable the conduction of Penetration Testing on Web applications. They enable crafting of http(s) requests/responses, analysis of session Ids, cookies, tokens, etc for randomness and crawling features that help in digging out hidden URLs and query strings. They also help in simulating attacks on the application server, such as "Denial of Service", "Buffer Overflow" and "Directory Traversal" attacks.
Examples of such Tools are WebScarab (from OWASP project), Samspade tools (from Samspade), BOU Utility (from IMPerva), Trin00 (from F-Secure).

Web Application scanners: These tools scan a Web Application for common vulnerabilities such as SQL injection, Buffer Overflow, etc. Examples of such tools are SpikeProxy (from Debian Project) and WebInspect (from SPI Dynamics).

5. Application Security – Budgeting Guidelines

Budgeting for any Security initiative involves evaluating the cost of various expected security breaches versus the cost of the Security initiatives. Care needs to be taken that the cost of the latter should not exceed the cost of the former.
Example of a Security Breach: While discussing the various factors that could influence the cost of security breaches and security initiatives, we can consider the example of an online shopping website where products can be bought online. A possible security breach that could occur in this case is a user trying to buy a product online without paying for the same by trying to bypass the billing mechanism of the shopping cart.

Some factors that could influence the cost of Security Breaches are as follows:

A). Revenue potential of vulnerable assets: In the above case, the products available for sale online are among the assets that are being secured. Higher cost of these products and higher frequency of purchase of these products would mean higher losses in case the security breach described above occurs.

B). Number of users likely to breach the security: In the above case, the number of customers registered with the above site (in case registration is needed) or the approximate number of targeted customers would be the number of users who might try to breach the security. This number, when significantly high, results in huge losses due to security breaches.

C). Legal costs: In the above example, when information pertaining to a customer (such as credit card or any other personal information) is leaked, legal liabilities could result from such a security breach. This is especially true when the concerned country's cyber security laws are stringent.

D). Intangible Costs: These costs could result due to loss of public image and loss of new customers in case the security incident gets published.

The Various factors that could influence the cost of Security Initiatives are as follows:

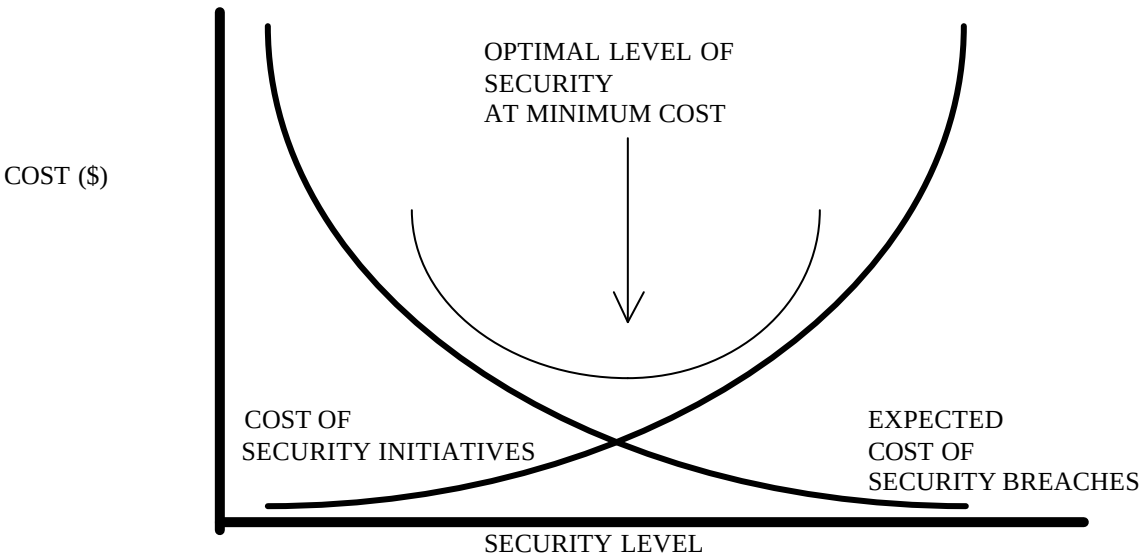
A). Application Complexity and Security Features: More complex applications with more authentication/authorization features tend to increase the effort involved in the Security Requirements study and vulnerability identification.

B). Security Standards Implementation: Cost of enforcing Security standards and incorporating security fixes in application code. This cost will be more for already developed applications that need to be fixed for security issues.

C). Penetration Testing: Effort involved in conducting penetration tests to check for the presence of the relevant vulnerabilities.

D). Tools: Cost of the tools that are used for penetration testing or any additional tools used for development/security fixes.

The graph below indicates that maximum benefits can be derived from a Security Initiative if the cost of the initiative is less than the cost of possible security breaches.



6. Benefits of Security Testing

Thorough Security Testing helps in the delivery of robust and secure Applications.
It ensures proper implementation of Security measures within an application, the advantages of which are manifold.

Reduced Cost of fixing Security issues: Fixing a security issue after it has been found in live applications is 6 times more expensive as compared to fixing the same issue during the Security Testing Cycle. A thorough Security testing process results in early uncovering of Security loopholes, thus saving on the cost needed to fix the issue.

Compliance to Cyber Security Laws: Proper Security testing ensures that applications comply with the various Cyber Security and Cyber Crime Laws that are applicable to the country in question. This avoids subsequent lawsuits and legal costs that result from reported security breaches.

Reduced Cost of IT Support: Timely detection of Security issues prevents wastage of efforts of IT staff in tracking down culprits and support activities (such as password resets, account reactivations).

Preservation of Integrity of Data and IT assets: A Web Application well tested for Security issues prevents malicious users from conducting unauthorized activities (such as admin operations) or viewing data for which they do not have access. This also helps in preventing theft of proprietary data by potential competitors.

Information availability: Attacks on an application server such as “Denial of Service” and “Buffer Overflow” cause the application server to crash or slow down to a crawl, thus preventing legitimate users from accessing the application. Proper Security Testing ensures that a Web Application becomes immune to such attacks, and thereby, ensures information availability.

7. Conclusion

With the increase in the number of Web applications and the volume of business conducted online, the criticality of thorough Security Testing for these applications cannot be over emphasized. Advancement in technology has also led to improvisation in cyber attack techniques, alarming rise in Cyber Crimes and immeasurable damages resulting from them.

From the above article, it is clear that Security Measures employed at the network level (such as Firewalls and Intrusion Detection systems) cannot prevent security breaches at an application level and the data handled by the applications. Hence, it is necessary to implement Security at an application level as part of Application Design and Coding practices. During the Testing phase, it is necessary to conduct penetration tests on the application, which help in timely detection of Security loopholes resulting from vulnerabilities.

The solution for combating this rising menace is to ensure the Delivery of Robust and secure Applications by conducting thorough security tests and audits on these applications against known vulnerabilities using well-defined processes and modelling techniques.

8. References

http://cnscenter.future.co.kr
http://msdn.microsoft.com
http://www.finjan.com
http://www.cgisecurity.com
http://www.gocsi.com
http://www.owasp.org
http://www.penetration-testing.com
http://www.securesoftware.com
http://www.windowsecurity.com

9. Author's Biography

Vivek Devarajan (vivek.devarajan@wipro.com) is currently working as a Technical Specialist in Wipro Technologies, Bangalore. He has around 7 years of experience in Enterprise and Legacy Software Design, Development and Testing. He is currently responsible for Software Testing projects of Enterprise Applications. He has been involved in architecting Test Solutions in technology areas such as Mainframes, Enterprise Applications, XML, Test Automation and Security Testing. He has received his Bachelor of Engineering degree in Electronics and Telecommunication Engineering University of Pune.

10. Appendix

Security breach: A security breach of an application is the violation of security controls built in the application in order to gain unauthorized access to application resources.
Authentication: The verification of the identity of a user.
Authorization: The process of deciding if a user is allowed to have access to a given resource or service.
DoS: Denial of Service
V & V: Verification & Validation

Definitions as taken from Microsoft website (http://msdn.microsoft.com):-

Asset: A resource of value, such as the data in a database or on the file system or a system resource.
Threat: A potential occurrence, malicious or otherwise, that might damage or compromise your assets.
Vulnerability: A weakness in some aspect or feature of a system that makes a threat possible. Vulnerabilities might exist at the network, host, or application levels.
Attack (or exploit): An action taken by someone or something that harms an asset. This could be someone following through on a threat or exploiting vulnerability.
Countermeasure: A safeguard that addresses a threat and mitigates risk.
Damage potential: The consequences to the application if an attack were to occur.

Definitions as taken from cnscenter (http://cnscenter.future.co.kr):-

Zero-day attacks: Attacks that take advantage of software vulnerabilities for which there are no fixes
Half-life of vulnerabilities: The time between when vulnerability is released and when it is exploited.