

Security Testing Handbook

Version: 1.2

Date: 6/23/2006

Prepared by: Jeff Guhin

TS 5150 – Enterprise System Testing

Table of Contents

Introduction.....	4
Testers and Hackers	4
Table 1. Vulnerability Classification. Source: OWASP 2004.....	5
Gathering info on the Target.....	5
Performing Reconnaissance.....	5
Areas to focus on:	5
Table 2. Grep Patterns. Source: Andrews and Whittaker, 2006.	6
Testers Toolbox:	6
Attack: Guessing Files and Directories.....	7
Areas of Focus:	7
How This Attack is Conducted:.....	7
How to Protect:	7
Testers Toolbox:	7
Attack: Risks with Third Party Components	7
Areas of Focus:	7
How This Attack is Conducted:.....	7
How to Protect:	8
Testers Toolbox:	8
Attacking the Client.....	8
Attack: Bypass Restrictions on Input Choices.....	8
Areas of Focus:	8
How This Attack is Conducted:.....	8
How to Protect:	9
Testers Toolbox:	9
Attack: Bypass Client-Side Validation.....	9
Areas of Focus:	9
How to test:	9
How to Protect:	9
Testers Toolbox:	9
State-Based Attacks	10
Attack: Hidden Fields	10
Areas of Focus:	10
How This Attack is Conducted:.....	10
How to Protect:	10
Testers Toolbox:	10
Attack: CGI Parameters	11
Areas of Focus:	11
How This Attack is Conducted:.....	11
How to Protect:	11
Attack: Cookie Poisoning	11
Areas of Focus:	11
How the Attack is Conducted:.....	12
How to Protect:	12
Attack: URL Jumping.....	12

Areas of Focus:	12
How This Attack is Conducted:	12
How to Protect:	12
Testers Toolbox:	12
Attack: Session Hijacking.....	13
Areas of Focus:	13
How This Attack is Conducted:	13
How to Protect:	13
Testers Toolbox:	13
Attacking User-Supplied Input Data.....	14
Attack: Cross-Site Scripting:	14
Areas of Focus:	14
How This Attack is Conducted:	14
How to Protect:	14
Testers Toolbox:	14
Attack: SQL Injections	15
Areas of Focus:	15
How This Attack is Conducted:	15
How to Protect:	15
Testers Toolbox:	15
Attack: Directory Traversal	15
Areas of Focus:	15
How This Attack is Conducted:	15
How to Protect:	16
Testers Toolbox:	16
Language-Based Attacks	16
Server-Side Attacks	17
Authentication.....	17
Conclusion	18
References.....	19

Introduction

Designing secure computer systems becomes more difficult as attackers find new ways to exploit application and system vulnerabilities. These attackers keep software vendors in a reactive mode to provide security as a necessary feature for their software products and networks. Years of research have aided security analysts with many powerful techniques to solve a wide variety of security issues. Claiming that a particular system utilizes a public-key infrastructure may sound impressive, but the statement has little value when taken out of context. The question then becomes, is this security feature necessary to protect the system and does it meet the system's security needs? Security measures should not be implemented frivolously. Bruce Schneier (1999) states that "Security is a chain; it's only as secure as the weakest link. Security is a process, not a product." A systematic engineering approach should be taken to identify security risks, requirements, and recovery strategies. Exposing the threats of a system or application before it is implemented helps architects develop testable security requirements.

Testers and Hackers

Hackers and security testers have some common characteristics but have different reasons for doing what they do. Hackers want to exploit bugs often with criminal intent. The testers want find them to prove they exist so they can be fixed. Both the hacker and the tester must observe subtle clues within the systems and application and insist in believing that there are vulnerabilities to be found.

The Hacker's mindset believes there is always security hole and they remain persistent in finding a way in. Security testers must also be persistent in finding these bugs, and know when to go above and beyond the conventional boundaries when testing security.

Hackers have formed a sub-culture that has a history of sharing information by publishing the exploits they find on news groups. They learn from each other enhancing samples of exploited code. Security testers also share techniques and tricks they have learned by testing them out in an isolated environment or sandbox that is safe to experiment. Testers have established a trusted network with their peers and industry experts to keep up with threats. Small improvements can help protect many applications from attack. The insider knowledge is critical to finding flaws before the hacker does.

Knowing the goals of a hacker helps the tester to defend attacks against the organizations assets. For example the attacker typically wants to obtain login and password information to authenticate into the system and copy data or corrupt it so the users don't trust it, intercepting or redirecting information such as confidential emails. Testers need to analyze the application or system under test and identify the operations the hacker would want to perform and the objects they would want to control. The testers must determine who could benefit from an attack, why they would want to do it, and how it can be done. The answers to these questions allow testers to draft test cases to find the protect or prevent a threat.

Security tests can be designed, implemented, and executed within any testing phase but are typically conducted at System Test and rerun during User Acceptance Testing. Identifying the classes of security vulnerabilities are essential in developing an adequate test plan. According to OWASP (2004) research, 92% of all security issues are application related. Table 1 below describes the major vulnerabilities of software and types of attacks which happen:

Vulnerability Classification	% of Occurrence
Encryption Issues	0%
Network and Protocol Stack Issues	1%
Communication Protocol Issues	2%
Hardware Issues	4%
Operating System Issues	15%
Non-Server Applications	36%
Server Applications	41%
Others	2%

Table 1. Vulnerability Classification. Source: OWASP 2004

Gathering info on the Target

It is critical to understand how the application is implemented and determine its assets in order to understand how an attacker would compromise security. A strategic approach to security will help to determine what tests should be executed first. This section features the following topics:

- Reconnaissance
- Guessing Files and Directories
- Risks with Third Party Components

Performing Reconnaissance

Areas to focus on:

- Comments embedded in HTML source code
- Sensitive information left in HTML source code
- Server side error messages and a HTTP responses

- Application error messages

The objective is to verify sensitive information is not revealed in source code or error messages such as passwords, usernames, database names that could be useful to an attacker.

Old code snippets found in comments may provide hints for an attacker. You can manually scan the pages or use an automated tool such as Grep for larger sites. Below in Table 2 are some patterns you can include in your test cases.

Item	Description	Grep Pattern
HTML comments	Usually not a big issue, but can occasionally reveal something useful	<!--[^\s/][w/W]*?>-->
Application comments	There should not be any application comments visible to the user.	<!--[w/W] *?--> ColdFusion //.* Single-line comments ^\s*[w/W] *?\/ C-Style block comments ^'.* Rem\s.* VB comments
IP addresses	Verify any IP addresses listed are not referencing servers other than the primary app server. Such as database or clustered servers.	[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}
E-mail addresses	Make sure private email addresses are not displayed such as a developers.	[w]*(\.[w]*)*@[.\w]*
SQL queries	Any queries found in the source of a pages can show the database structure and how queries are constructed	SELECT\s[\w]*\s/(\s,)+\sFROM\s[\w]+ UPDATE\s[\w]+\sSET\s[\w,']+=+ INSERT\sINTO\s[\d\w]+([\s\w\d\)]\s,)*\sVALUES\s([\d\w\']\s,)+ + DELETE\sFROM\s[\d\w\']+=+
Database connection strings	Search for common key words in connection strings.	Provider/Data/sSource/Driver
Hidden input fields	Identify their location to save time.	<input\s[\w\W]*?type="(\\)?hidden(\\)?[\w\W]*?>

Table 2. Grep Patterns. Source: Andrews and Whittaker, 2006.

Testers Toolbox:

CGYWIN: This is a free Linux-type environment tool for windows platforms that includes the Grep utility.

Regulator 2.0: <http://regex.osherove.com/>

This tool helps you create your own search expressions.

Attack: Guessing Files and Directories

Many files are stored in standard locations in a sites files structure. If an attacker can guess the folder and files names they could potentially access the page or directory with the absolute URL even if there is not a hyperlink to it (Andrews, Whittaker, 2006).

Areas of Focus:

- Sensitive documents that should be protected.
- Administrative and configuration pages

How This Attack is Conducted:

- Look for patterns such as sequential numbers incrementing or decrementing for file name and account numbers.
- Look for sub-sites in the root site. Folders titles admin, control, cp are good directories to try.
- Admin pages could be running on a different port on the web server. There are a limited number of port available, and you may be able to recognize those that are already reserved. Application specific ports are typically greater than 1024.

How to Protect:

Make sure to configure the web server to disable directory browsing so it can't serve pages outside the application.

Testers Toolbox:

- A port scanner could be used to map open ports identify a control panel and connect to it.
- HTTrack is a website copier and a great tool for reverse engineering a sites file structure. <http://www.httrack.com> It allows the user to download the site locally in the same directory structure on the server including all files and images maintaining the relative links.

Attack: Risks with Third Party Components

Developers often reuse existing components whenever possible. There is not a central organization that validates these components so anytime a developer reuses code written by others there are risks involved.

Areas of Focus:

Third party components installed on the web server.

How This Attack is Conducted:

Hackers subscribe to mailing lists that publish vulnerabilities. Many of these provide details of what the issue is and how to exploit it.

How to Protect:

Audit the web server for third party components. If any are found research how frequently it gets updated, if it has been subjected to previous attacks or if there are security concerns. Any component that is installed should be validated for proper input and output validation.

Testers Toolbox:

Nikto: <http://www.cirt.net/code/nikto.shtml>

Nikto is an open source tool used for scanning a web server for vulnerabilities. Configuration issues and custom error pages may produce false positives.

Attacking the Client

The client system is the hardest part of the system to secure, which makes the end-user's computer a likely candidate for an attacker to focus on.

Attack: Bypass Restrictions on Input Choices

Areas of Focus:

Identify input validation mechanisms that are coded on the client side that can be bypassed. The following are ideal targets in web forms:

- Checkboxes
- radio buttons
- drop-down lists
- text input boxes.

Look in the source HTML for text boxes that have restrictions set for the MAXLENGTH attribute on input elements.

Web applications communicate with servers via Common Gateway Interface (CGI). Data sent via links are displayed in the URL after a question mark. Attackers can modify the url requests because browsers don't validate this. The client side validation is bypassed completely.

If an attacker inserts data into your application that is not validated a number of unexpected results could occur including unauthorized credits/debits, revealing error messages, or data corruption (Andrews, Whittaker, 2006).

How This Attack is Conducted:

In this example, lets refer to a web form that takes orders online. An attacker has two options to perform this attack.

1. Save the web page locally. View the source and modify default restrictions. The attacker can use the modified page locally instead of the page served by the web server to send the modified values to the remote server.
2. Use a program to automate sending requests with pre-selected or dynamically created values to bypass the GUI imposed selection criteria. This technique generates more possibilities in a short amount of time compared to the first option.

How to Protect:

Prevention is the best approach in this case. All restrictions enforced on the client side should be verified on the server side to avoid this attack.

Testers Toolbox:

Paros: <http://www.parosproxy.org/index.shtml>

This free tool is available for testers to assess the security in web applications

Attack: Bypass Client-Side Validation

Client-side scripts run when the user performs an action on the page, like moving the cursor or clicking a button. These scripts typically are used to validate the users input. But any validation done and the client can be modified by an attacker.

Areas of Focus:

JavaScript is the most commonly seen on the client. It is easy to see if it is being used on the web page by viewing the HTML source. Developers may include a reference to a file with a .js extension but it can also appear with tags like: `<script>...</script>`. You may also see the other HTML tags prefixed with 'JavaScript'. JavaScript is triggered by an event, so look for keywords that are used in validation: OnLoad, OnBlur, OnSubmit.

How to test:

As a tester, you want to identify what is being validated, remove the validation, and determine if the data is being validated on the server-side. If its not, write a defect. Any mismatch in data could cause maintenance problems for developers.

You can turn off the ability to run scripts in the browser by using the browser settings, but it may break other unrelated features like navigation. An alternative would be to save the HTML source locally and edit the validation script and change the relative URLs to absolute.

How to Protect:

Any real validation should also be handled on the server side, but there may be performance hit in result.

Testers Toolbox:

PageSpy: <http://www.sembel.net/>

HTML document analysis tool that can use the Document Object Model (DOM) to identify associated scripts to a page and allows you to turn script elements off.

State-Based Attacks

The web by default does not remember which pages a user has accessed or the order they have viewed them. It is up to the developers to code rules to grant page access and track session management (Andrews, Whittaker, 2006). The following areas are discussed in relation to how web testers would want to validate their application under test is not vulnerable to state-based attacks.

Attack: Hidden Fields

Hidden fields are one way to preserve the state in web pages. When a form is processed to the web server hidden fields are included in the GET or POST parameters. Users can't see them but the web application does. Hackers look at hidden fields to determine if there is any useful information they can use to exploit the system.

Areas of Focus:

Hidden fields do have data types associated with them. An attacker could change the value to overly long strings, input special characters that could result in negatively affecting the web server.

How This Attack is Conducted:

Open the HTML source and search for the word: "hidden".
Save the page locally, change relative links to absolute links, remove the word "hidden" in the HTML source, Save and reload the page. Removing the "hidden" value displays a textbox on the page that can now be modified.

How to Protect:

If the application does use hidden fields, determine if the information stored there would be useful to an attacker. Does it contain session information or pass product details such as prices or quantities from one page to the next?

Avoid use of hidden fields at all. If they are necessary consider using obscure field names that are less obvious and encrypting or hashing the values.

Testers Toolbox:

PageSpy: <http://www.sembel.net/>

HTML document analysis tool that can use the Document Object Model (DOM) to identify hidden fields and allow you to change them from a GUI.

Attack: CGI Parameters

CGI parameters function similar to hidden fields, but instead of the user pressing a submit button located in a form, the user can pass data by clicking a hyperlink or image. CGI parameters are passed via the page requests URL after the question mark “?” and name-value pairs are separated by the ampersand “&”(Andrews, Whittaker, 2006).

Areas of Focus:

You can tell if a current page is passing CGI parameters if you analyze the address bar of the browser. If the functionality is enabled you would see the parameters in the URL. For example: <http://www.google.com/search?hl=en&q=CGI+parameters>.

CGI parameters can be used to pass information on user preferences, pages the user has accessed, whether the user has registered or been authenticated. Some parameters have short names that consist of single characters and may represent Boolean values. It is important to know what parameters are being used to determine the security risks.

How This Attack is Conducted:

- First browse the site and note the address bar value.
- Hover the mouse over hyperlinks to note the URL displayed in the information bar at the bottom of the browser.
- Note any CGI parameter (the values after the ? in a URL) and determine what that data represents and if it could benefit an attacker.
- You can modify the HTML source to change the POST method values.
- You can simply modify the URL for GET parameters in the address bar.

How to Protect:

Validate all input on the server side.

Testers Toolbox:

Paros: <http://www.parosproxy.org/index.shtml>

This free tool allows you to view and modify all HTTP traffic traveling between the web server.

Attack: Cookie Poisoning

Cookies are text files that are stored on a client machine. They can be used to help the web application track a variety of things such as demographic information, how often a user visits a site or how long they stay there. They can be used to personalize the user experience by customizing the content the user sees

Areas of Focus:

Cookie poisoning involves the modification of the contents of a cookie in order to bypass security mechanisms. Using cookie poisoning attacks, attackers can gain unauthorized information about another user and steal his identity (Imperva 2006).

How the Attack is Conducted:

Look for expiration dates userid and password information in your cookies folder located on the client machine. For example Internet Explorer stores cookies in: C:\Documents and Settings\username\Cookies. You can use Google to find the specific cookie formats for the browser you are using. Determine if there is any sensitive information being stored, see if you can edit the values and revisit the website and watch the behavior.

How to Protect:

Consider carefully what data is getting stored on client machines. Use certificates or server authentication processes for granting access instead of cookies.

<http://www.lodoga.co.uk/attackinfo/thethreat/examples/cook.htm>

Consider encrypting any sensitive information located in cookies.

Attack: URL Jumping

The web was designed to be stateless, so users can jump from page to next if they type in the URL if they know it or can guess it. Developer may want the user to follow a specific sequence of pages before accessing a checkout page for example.

Areas of Focus:

The focus of this test is to determine if the user can bypass pages that the developer intended to be sequenced

How This Attack is Conducted:

It is helpful for the tester to have a page map and diagram the intended sequence of pages.

- First navigate through the site as how the developer intends a user to experience the site, and note the addresses of each page and their sequence.
- Then use the list to attempt to access the URL's at random, and look for any error messages the attacker might find useful.
- You can modify hidden fields, cookies values, or the HTTP referrer to try and force access. If you can access a page this way then write of a defect.

How to Protect:

Developers may compare the last visited page to the page requested to validate they are following the proper sequence. It is slightly more secure to use cookies to store this information than hidden fields or CGI parameters because temporary cookies are passed in the HTTP header which users can't control through the browser and is harder to modify (Andrews, Whittaker, 2006).

Testers Toolbox:

Paros: <http://www.parosproxy.org/index.shtml>

This free tool is available for testers to change cookie's value.

Attack: Session Hijacking

Session management grants each user a unique identifier that is assigned to them while they are using a web site. This identifier can authenticate users and store their state information on the server. Session identifiers are typically stored in cookies and passed to the server when each page is loaded, however they can be initiated via hidden fields or CGI parameters.

Session hijacking refers to the ability to breach session management by swapping the session identifier with another user. According to (Andrews, Whittaker, 2006), there are several ways to do this:

- Modify data at random hoping to find the identifier of another user
- Determine the sequence of unique identifiers that are being issued
- “Fixing” the session identifier of another user

Areas of Focus:

- Hidden fields
- CGI Parameters
- Cookies

How This Attack is Conducted:

Gather multiple session identifiers and attempt to decipher a pattern. IF you can find a pattern, replace the session identifier value with another valid and request the page again. Attempt this scenario several time and if you are able to view personalized information of another user, write the defect up.

How to Protect:

Andrews and Whittaker (2006), recommend a number of precautions:

- Make sure the sequence of identification numbers are unpredictable
- Cookies are generally more difficult to modify than hidden fields or CGI parameters. You can protect them by using mechanisms like setting the secure flag (so they cannot be "sniffed" unencrypted on the network).
- Time-out session identifiers so someone cannot reuse them after a predetermined period of time.
- Allow users to log out and clear their session.
- Utilize the HTTP referrer field to identify multiple clients browsing with the same identifier.
- Make sure session cookies are sent over secure channels so they can't be captured in transit.

Testers Toolbox:

Paros: <http://www.parosproxy.org/index.shtml>

This free tool is available for testers to change cookie's value.

Attacking User-Supplied Input Data

Attack: Cross-Site Scripting:

Cross-Site Scripting is also referred to as XSS. According to <http://www.Webopedia.com> This attack conducted on dynamic web pages in which an application is sent with a script that executes when read by an unsuspecting user's browser or by an application that has not protected itself against XSS.

Areas of Focus:

Dynamic web sites deliver content based on user input, an attacker can input a malicious script into the page by hiding it within legitimate requests, steal a users cookies to impersonate them, or redirect the user to an another site that appears to be the trusted site so the user will input personal information that the attacker can use (Andrews, Whitaker, 2006).

Common targets for exploitation include:

- search engine boxes
- online forums
- blogs
- guestbooks
- CGI parameters in a URL
- Script sent via email embedded in a link

How This Attack is Conducted:

This attack is commonly attempted through form input fields that write to a database or file or a placed in a URL replacing the CGI parameters (Andrews, Whittaker, 2006). Once XSS has been launched, the attacker can change user settings, hijack accounts, poison cookies with malicious code, expose SSL connections, access restricted sites and even launch false advertisements.

How to Protect:

The simplest way to avoid XSS is to add code to a Web application that causes the dynamic input to ignore certain command tags that are interpreted as code. It is common for developers to have white-list that include a list of acceptable tags.

Testers Toolbox:

XSS cheat sheet: <http://hackers.org/xss.html>

<http://plynt.com/blog/2005/06/testing-xss-special-cases/>

<http://www.microsoft.com/technet/community/columns/secmvp/sv0505.msp?pf=true>

Attack: SQL Injections

This attack targets dynamic web sites where the attacker executes unauthorized SQL commands on the server bypassing the firewall.

Areas of Focus:

SQL injection attacks attempt to steal information from a database which requires authentication and authorization of the user.

How This Attack is Conducted:

An attacker focus on submitting queries into web forms or parameter calls to an API. This typically starts with inserting a select command into the form to determine if any mitigations are in place to protect against this attack. Once an attacker determines this is a vulnerability there is potential to steal data, insert values into the database, or perhaps drop tables from the database. See the examples of actual sql statements in the Testers Toolbox section listed below.

How to Protect:

SQL injection attacks can be avoided with server side input validation. Any validation on the client side can be tampered with. Andrews, Whittaker, (2006), recommend:

- Filter special characters from URL's, forms, and user controls.
- Use stored procedures
- Create database credentials for each user and only allow access to what they need.

Testers Toolbox:

SQL Injection examples: <http://www.unixwiz.net/techtips/sql-injection.html>

Attack: Directory Traversal

In this attack the Hacker is attempting to access or execute restricted directories and files.

Areas of Focus:

The attacker is focused on:

- Accessing or executing restricted files
- Obtaining connection string information, local passwords, application code
- Useful error messages

Any of this information may prove useful for additional attacks.

How This Attack is Conducted:

Use the web application in the intended manner and note the URL addresses that are accessing files to be rendered in the browser. You can attempt to access other files if you can determine other file names. Enter commands to such as “..” in the URL to attempt to navigate up directory levels. If the attacker is able to navigate all the way up to the servers cmd.exe file they could run the windows command shell and view the directory

listing on the C drive. This is a huge danger. Refer below to the Testers Toolbox link for syntax examples.

How to Protect:

Andrews and Whittaker (2006), provide the following recommendations with the disclaimer that security can also slow down the delivery of the pages and potentially affect performance:

- Restrict web application to server pages from the web root directory and sub directories.
- Use Access Control Lists to grant page access
- Server side filtering

Testers Toolbox:

Directory Traversal syntax examples:

http://www.imperva.com/application_defense_center/glossary/directory_traversal.html

Language-Based Attacks

Language based attacks are focused on known issues with the programming language used on components implemented on the web server. It is important to distinguish that they are not attacking via the web application but through the web server.

Buffer Overflows occur when a program fails to check the size of input data that is being processed. When the input data is larger than the space that has been allocated for it, it overflows to other memory locations on the execution stack corrupting the other memory locations. This causes the machines execution sequence to change and allows the attacker run their code on the web server. The easiest way to protect against this attack is to truncate the extra data locally and let the user know. But also add server side validation because an attacker can bypass any client side validation

Canonicalization: This is also a language based attack.

This refers to verifying that all data is in a standard format. For example, character being passed from HTTP/HTML need to be encoded so the request is not broken when passed via CGI parameters.

Null-String Attacks: This is also a language based attack.

Web applications are written in high-level languages like ASP, JSP, PHP but use libraries of pre-written code often written in C/C++ which is a low-level language. When there are different ways of handling null characters between programming languages there can be risks in the data validation mechanisms.

This document was prepared as a beginners guide to test security mainly on web applications. The common theme throughout the paper emphasized server side validation because any client side validation can be bypassed. Most of the attacks described use the web application to gain access into the server. There are additional security topics not covered in this documents that should also be reviewed when determining what security test coverage is needed. Additional investigation should include the following topics:

Server-Side Attacks

Server-side attacks are focusing on the web and database servers instead of the web application front end. Databases and Operating systems are the primary target. The attacker attempts to send commands to the server to compromise. These server-side attacks and prevention suggestions include:

- SQL Injections using Stored Procedures – Don't use admin accounts to trigger stored procedures. Use accounts specifically set up for this purpose that do not have permissions to drop or create tables. Here is a checklist example for Sql Server security: <http://www.securitymap.net/sdm/docs/windows/mssql-checklist.html>.
 - Command Injection – The primary prevention method here again is input validation on the server. Never rely on client side validation because it can be altered.
 - Fingerprinting the Server – Keep track on any vulnerabilities that are discovered on the systems you own. Attackers know when new exploits are published. You must also be aware. BugTraq is a good source for monitoring new exploits: <http://www.securityfocus.com/>
 - Denial of Service – This is difficult to protect against. Clustering and load balancing can help to reduce the risk. Intrusion detection systems and bandwidth management systems can also be used (Andrews, Whittaker, 2006).
-

Authentication

Web applications also present the need for implementing strong mechanisms for authentication and encryption. Web security should provide the following basic needs: Confidentiality for the intended recipient, integrity in that the intended message or transaction is received unaltered, and availability of the system (Andrews, Whittaker, 2006). Attackers will attempt to compromise the system in four areas. Preventive measures are suggested:

- Fake Cryptography – Developers should use algorithms that are acknowledge as secure such as RSA, Triple DES, and AES, and avoid using any home grown or bleeding edge algorithms because they are easier for attackers to exploit.

- Breaking Authentication – Make sure all authentication occurs on secure connections. Use complex passwords with mixed case, special characters and numbers. Enforce locking of accounts after a number of failed attempts.
 - Cross-Site Tracing – Turn off TRACE HTTP method on all web servers so attackers cannot obtain authentication tokens.
 - Forcing Weak Cryptography – Make sure the server use SSL, remove weak ciphers
-

Conclusion

Security testing requires a wealth of systems knowledge which combined with policy needs to be captured early on in the design phase. System Testers need to validate authentication, integrity, privacy, non-repudiation, and availability. Security testing requires a different mindset to find security bugs. A willingness to think like an attacker and persistence to keep going until you find a bug are essential. Techniques such as threat modeling don't require programming skills however code analysis does. There are a variety of courses and certifications schemes exist to help build these skills.

Anyone involved with security testing must convince the project sponsors how important it is to have valid security requirements in place before implementing the system. Threat Modeling is useful technique in uncovering the threats of the system and documenting assets and the vulnerabilities. Performing risk analysis and mitigation plans are essential. Testing security early in project lifecycle will help discover potential threats early such as in code reviews. It is also important to include security testing in systems testing and User Acceptance Testing. There are a variety of tools for testers to use while testing for code and security flaws. Many companies offer automated s/w tools that test for vulnerabilities e.g. Codenomicon provides test tools that ensure s/q complies with relevant protocols such as BGP for routers, HTTP for web servers, etc.

References

- Enerjy Software. (2005). WHITEPAPER: The Case for Code Quality Management. Empowering Development Managers to Continuously Improve Application Quality. Retrieved May 23, 2006, from <http://stickyminds.com>
- Imperva (2006). Cookie Poisoning Attack. retrieved May 25, 2006, from: http://www.imperva.com/application_defense_center/glossary/cookie_poisoning.html
- J.D. Meier, A. M., Blaine Wastell. (2005). "Threat Modeling Web Applications." Retrieved May 12, 2006, from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/tmwa.asp>
- Lodoga. (2006). Web Attacks: Cookie Poisoning. retrieved May 25, 2006, from: <http://www.lodoga.co.uk/attackinfo/thethreat/examples/cook.htm>
- Meier, J. D., Mackman, A., Dunner, M., Vasireddy, S., Escamilla, R., & Murukan, A. (2003). Improving Web Application Security: Threats and Countermeasures. Retrieved April 28, 2006, from <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/ThreatCounter.asp>
- Myagmar, S., A. J. Lee, et al. (2005). "Threat Modeling as a Basis for Security Requirements." Retrieved May 17, 2006, from <http://www.ncassr.org/projects/sift/papers/sreis05.pdf>.
- OWASP. (2004). The Ten Most Critical Web Application Security Vulnerabilities. Retrieved May 28, 2006, from <http://www.owasp.org/documentation/topten.html>
- Pudipeddi, H., (2005). Improving Software Security. Retrieved April 23, 2006, from <http://stickyminds.com/sitewide.asp?function=search&kind=simple&tt=SRCHBOX&freetext=improving+software+security>
- Scheier, B., (1999). Attack Trees – Modeling Security Threats. Retrieved May 23, 2006, from <http://www.schneier.com/paper-attacktrees-ddj-ft.html> 2-23-06
- Seddon, D., (2002). Penetration Testing: thinking outside the (black) box. Retrieved April 1, 2006, from <http://www.corsaire.com/articles/020807-penetration-testing-thinking-outside-the-black-box.html>
- Singh, A., Iyer, A., Seshadri, V., (2001). A Parametric Approach for Security Testing of Internet Applications. Retrieved April 12, 2006 from http://www.softwareoxide.com/Channels/events/testing2001/Presentations/Arun_infosys.pdf

SPI Dynamics. (2002). SQL Injection: Are Your Web Applications Vulnerable.
Retrieved May 12, 2006 from:

<http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf> 2-23-06

Swiderski, F. and W. Snyder (2004). Threat modeling. Redmond, Wash., Microsoft Press.

Whittaker, J. A. and H. H. Thompson (2006). How to break web software security : functional and security testing of web applications and web services. Boston, Pearson/Addison Wesley.