

Subject: Security Testing

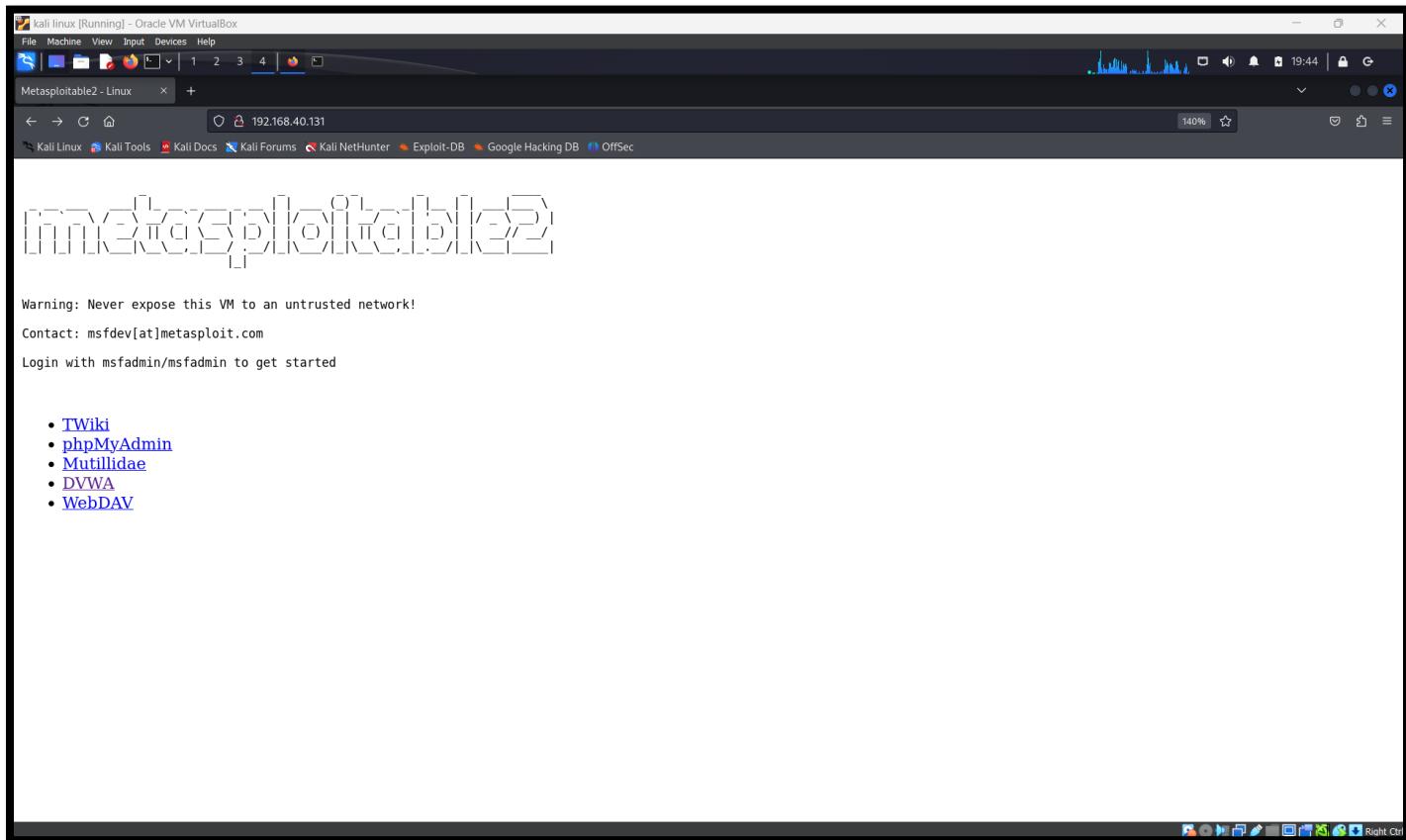
SENG 8061

ASSIGNMENT – 2

**Vulnerability Assessment
on DVWA application**

| Name | Student ID |
|------------------|------------|
| Shivani Varu | 8941914 |
| Mohammed Rafique | 8954785 |

We installed Metasploitable2 on our virtual machine and started it up. Next, we logged into the terminal of Metasploitable2. To identify the IP address, we entered the command 'ifconfig' in the terminal and pressed enter. After obtaining the IP address, we copied it for later use. With the IP address in hand, we opened our web browser and entered the copied IP address into the address bar. This directed us to the website hosting DVWA (Damn Vulnerable Web Application).



Vulnerability #1: No password authentication was found.

Discovered Vulnerabilities

We found a serious vulnerability when we evaluated the DVWA application. We discovered that there was no password authentication mechanism when we tried to log in as the admin user. This means that anyone can use the application without being required to provide authentic credentials.

The screenshot shows a Firefox browser window running on a Kali Linux VM. The address bar displays the URL <http://192.168.40.131/dvwa/index.php>. The DVWA logo is at the top right. The main content area features a green header bar with the text "Welcome to Damn Vulnerable Web App!". Below it is a "WARNING!" section, a "Disclaimer" section, and a "General Instructions" section. A sidebar on the left contains links for Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. A message box at the bottom left says "You have logged in as 'admin'". At the bottom of the page, there is a status bar with "Username: admin", "Security Level: high", and "PHPIDS: disabled".

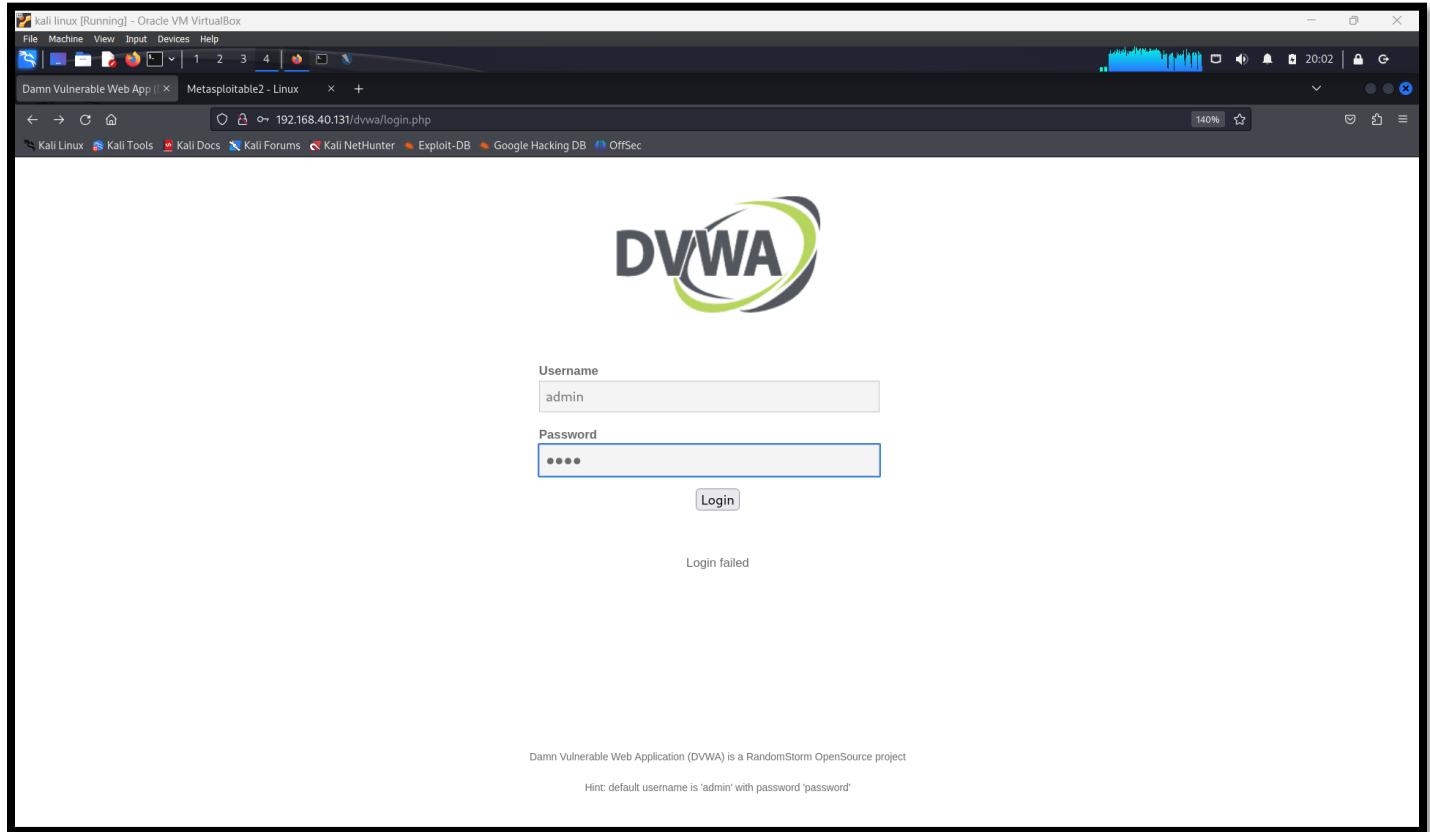
To exploit this vulnerability, we utilized OWASP Zap to execute a brute force attack. With the absence of authentication protections, attackers would have been able to access sensitive areas of the application without authorization.

Our next step was to set up Zap to act like an attack and attempt to bypass the user credentials.

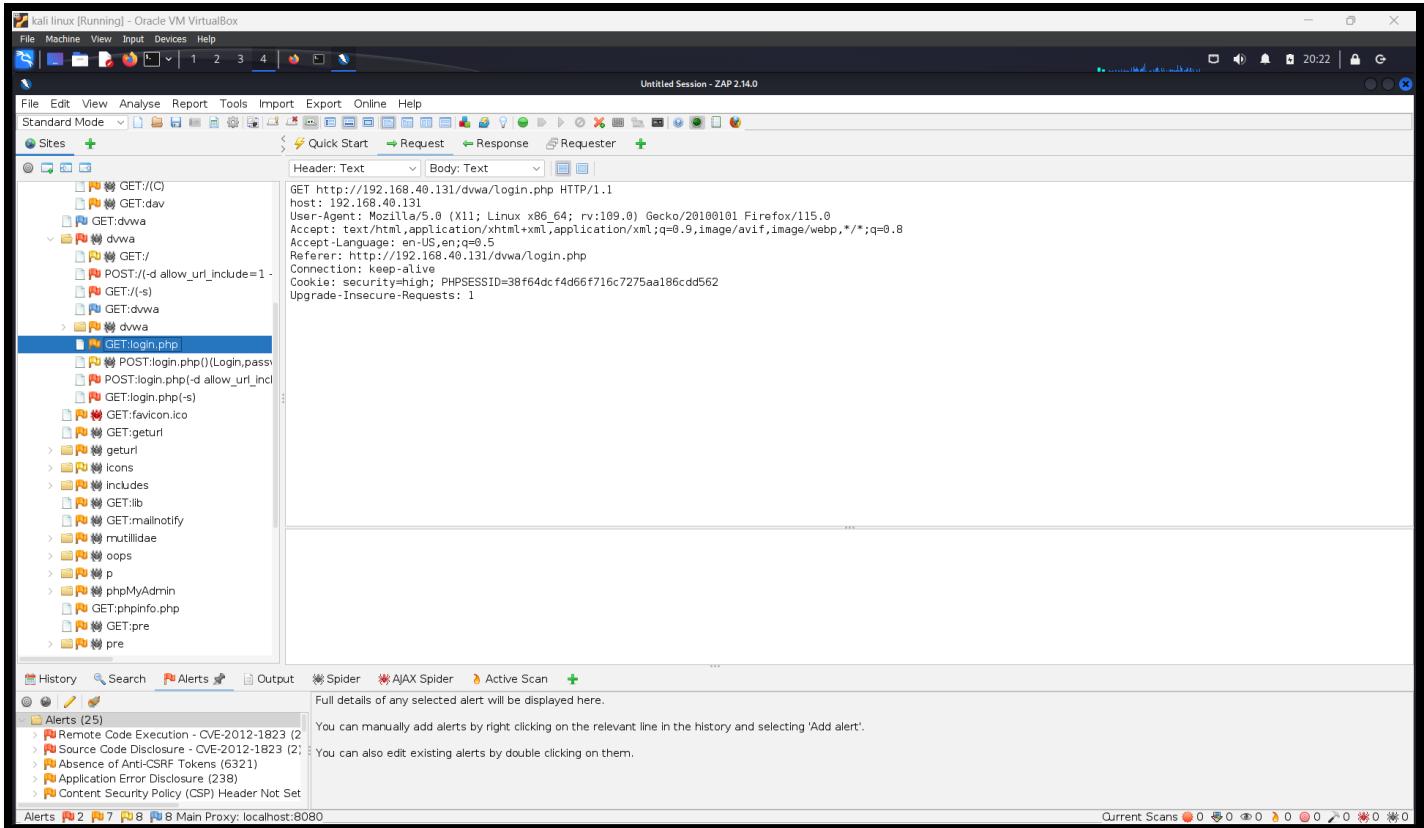
The screenshot shows the OWASP ZAP 2.14.0 interface. The title bar reads "Untitled Session - ZAP 2.14.0". The main window has a "Welcome to ZAP" message. It includes fields for "URL to attack" (set to <http://192.168.40.131/>), "Attack" (with "Firefox Headless" selected), and "Progress" (showing "Not started"). On the left, there's a tree view under "Sites" with "Default Context" selected. The bottom navigation bar includes tabs for History, Search, Alerts, Output, and a table for managing proxy requests.

We opened the DVWA application in the Zap browser and entered the IP address for Metasploitable. Then, we attempted to log in with the username "admin" and the password as SQETA.

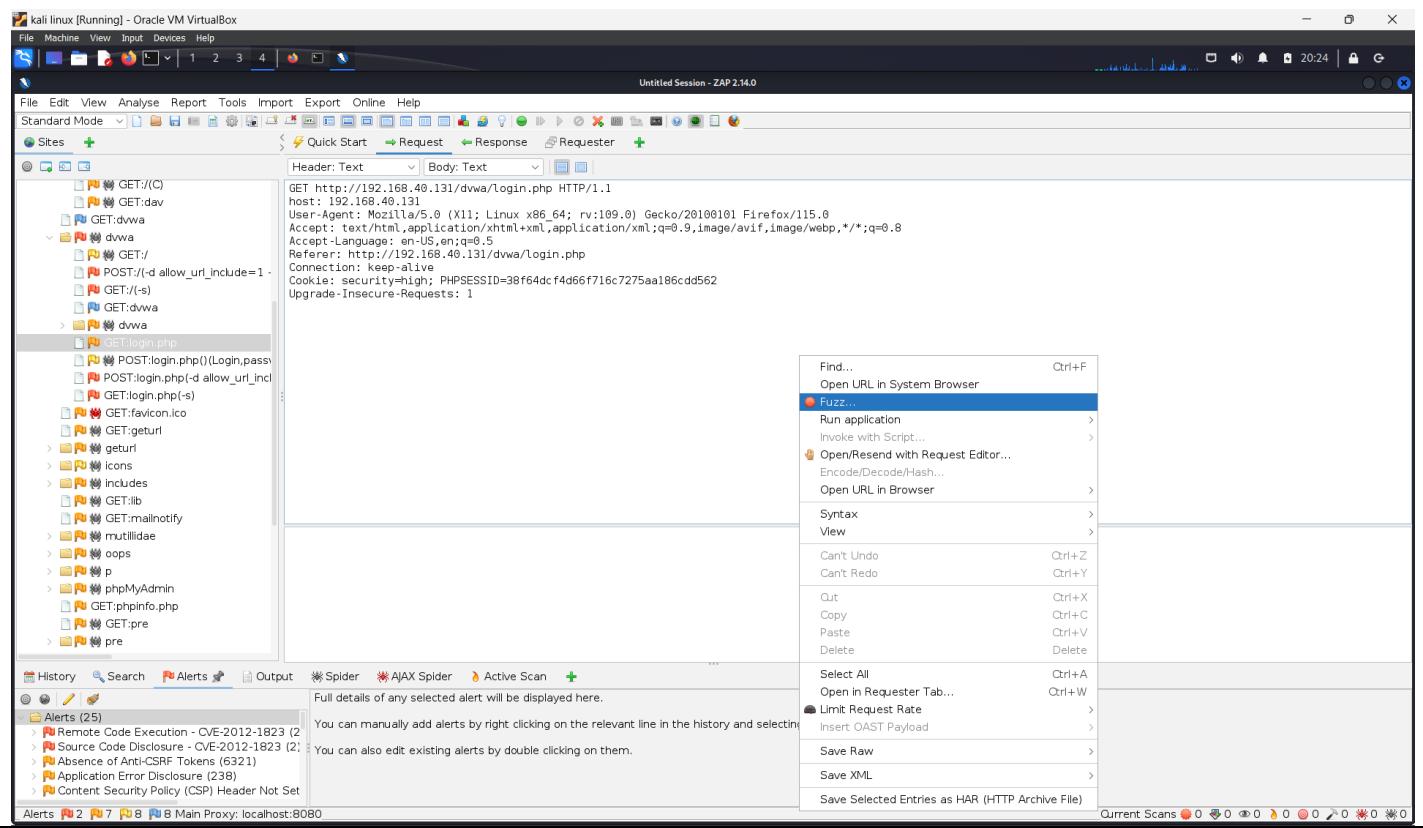
This shows an invalid password now let's manipulate the password payload and try to enter some random strings which are commonly used passwords in the world.

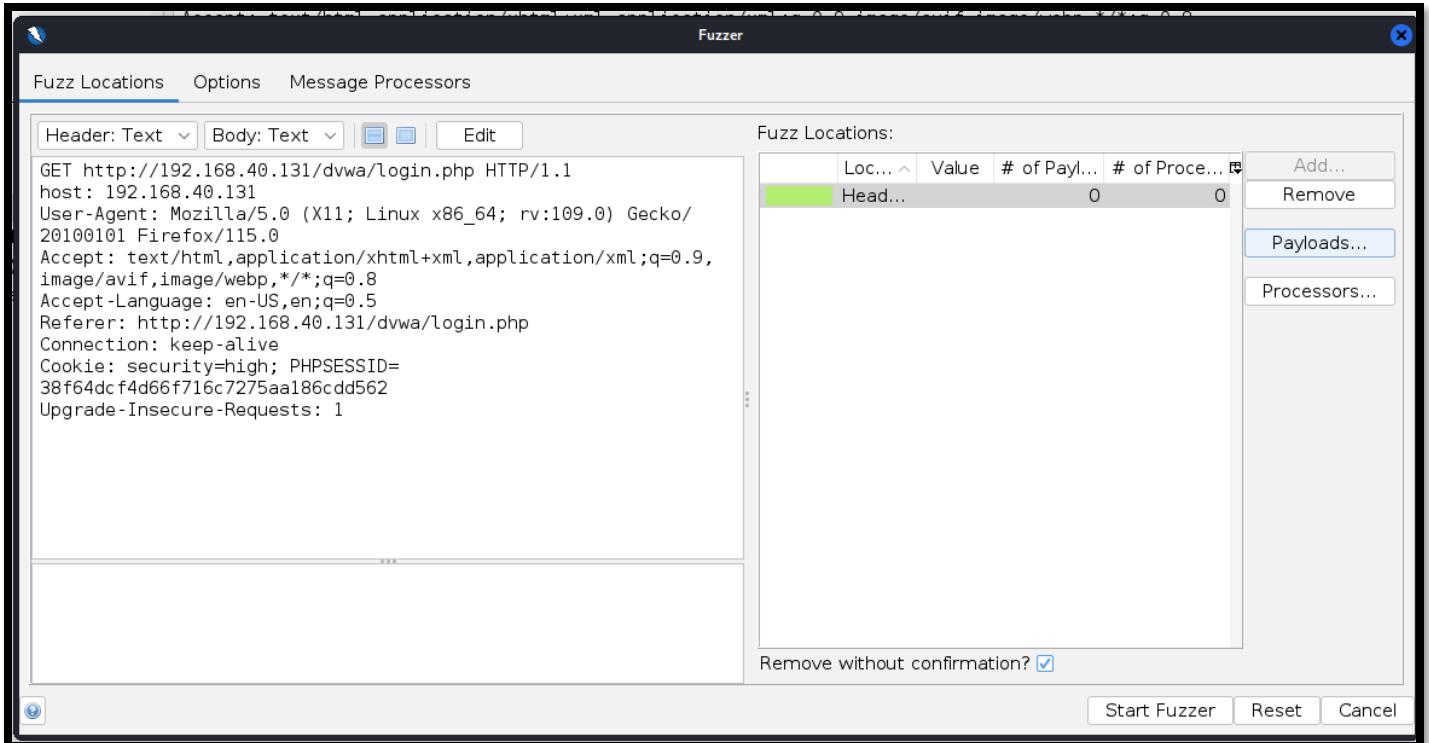


We paste DVWA url path to zap with automated scan option, We clicked on start attack , after attacking completed, we went to sites on left side under the sites we found get login.php, here we clicked on request tab .

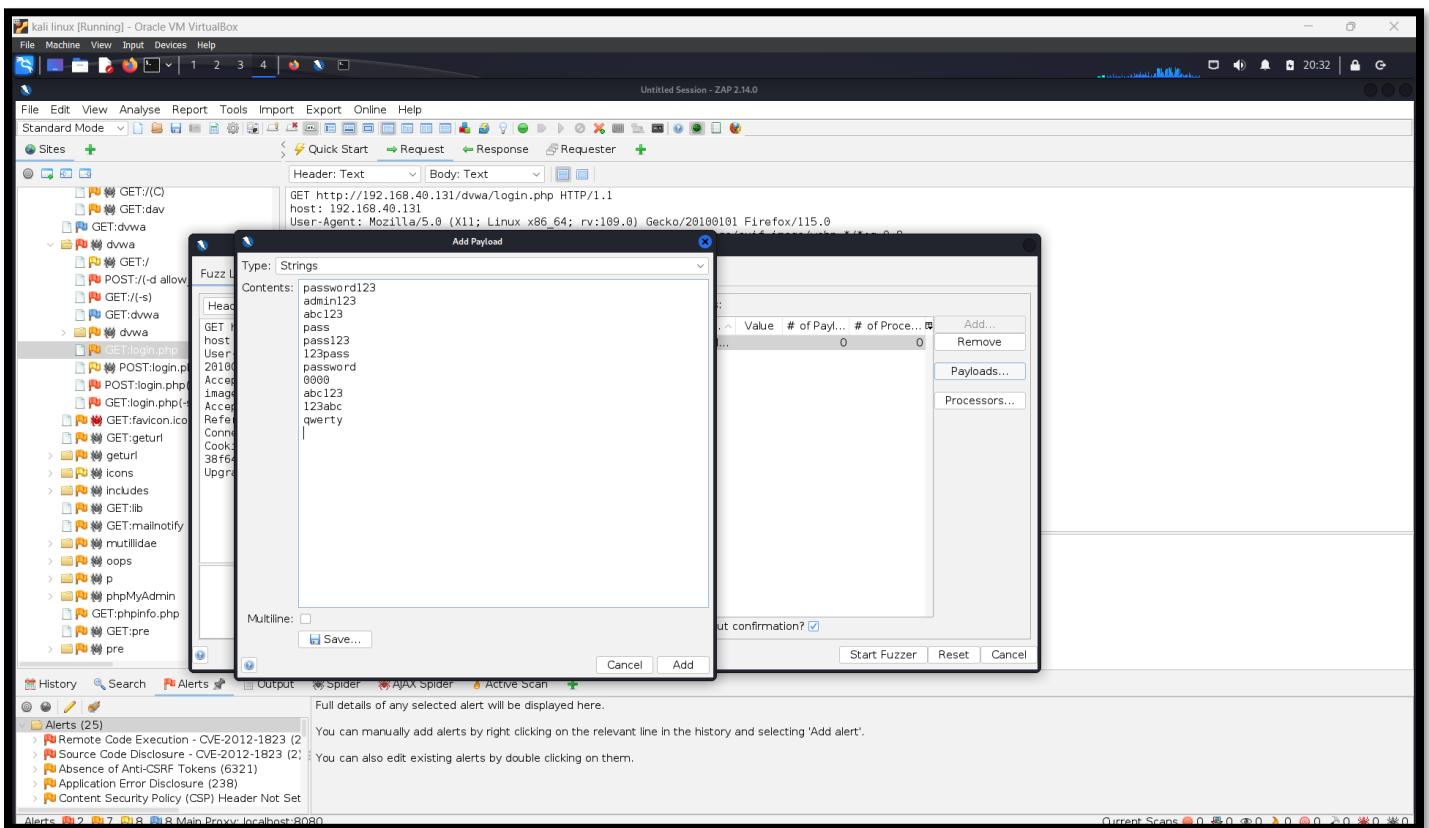


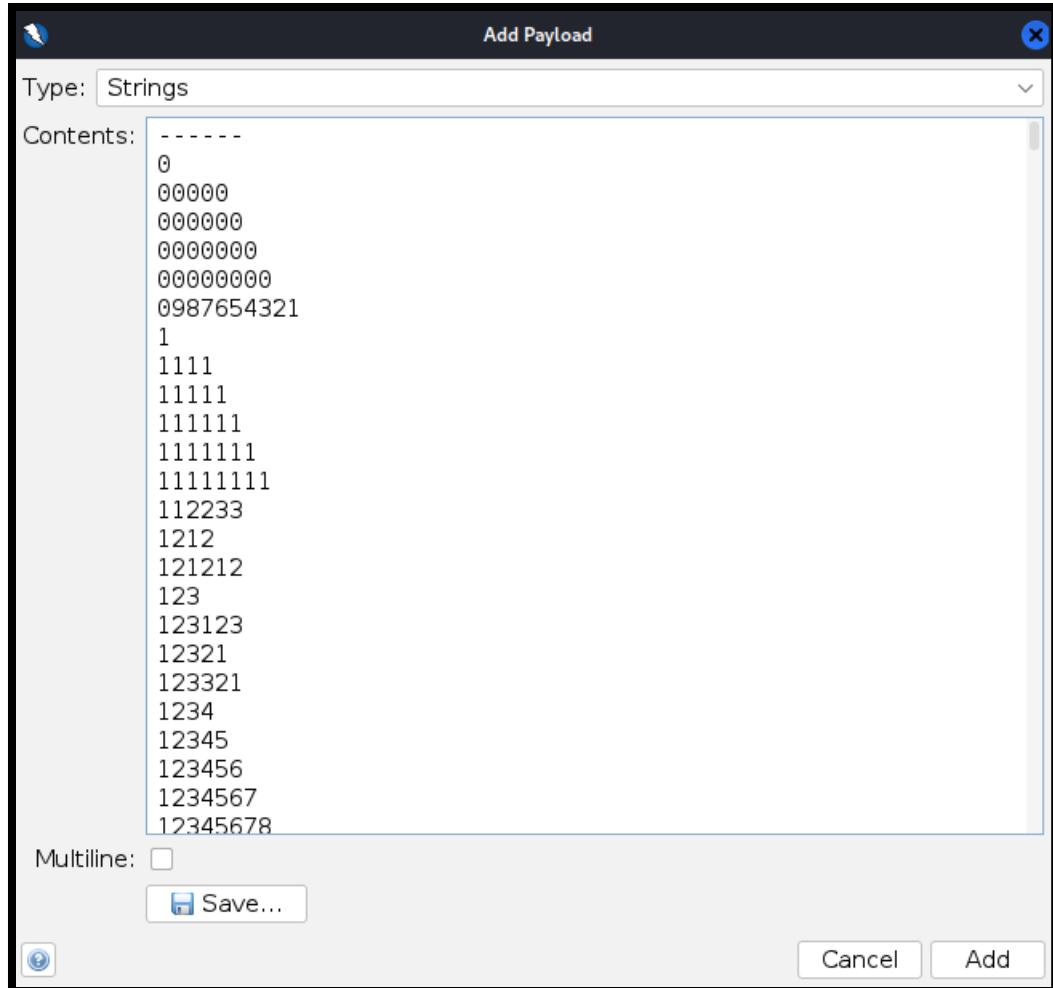
We tried logging in with the given credentials and then right-clicked the page to get to the FUZZ option. We chose fuzzing because it is a security testing technique used to find vulnerabilities in software and code errors. In this case, we used FUZZ to find vulnerabilities in the system. This is done by fuzzing, which involves providing huge amounts of data—often in the form of incorrect or unexpected inputs to test the system for vulnerabilities.





Then we click on payload button where we add many commonly used password copied from econestoga best1050.txt file in our course content.





Then we click ok button, then we click on start fuzzer button. Using OWASP Zap's Fuzzer, we discovered that the page responded to a variety of password strings. We went further and logged into the application to confirm the authenticity of these passwords.

ZAP 2.14.0 - Untitled Session

Header: Text | Body: Text

POST http://192.168.40.131/dwww/login.php HTTP/1.1
host: 192.168.40.131
user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/116.0.0.0 Safari/537.36 OPR/102.0.0.0
pragma: no-cache
cache-control: no-cache
content-type: application/x-www-form-urlencoded
referer: http://192.168.40.131/dwww/login.php
content-length: 37
Cookie: security=high; PHPSESSID=71952e8b0b7cab48206bf16d3a087870

username=ZAP&password=ZAP&Login=Login

History | Search | Alerts | Output | Spider | AJAX Spider | Active Scan | Fuzzer | +

New Fuzzer Progress: 0: HTTP - http://192.168.40.131/dwww/login.php | 100% | Current fuzzers: 0

Messages Sent: 1049 Errors: 0 Show Errors

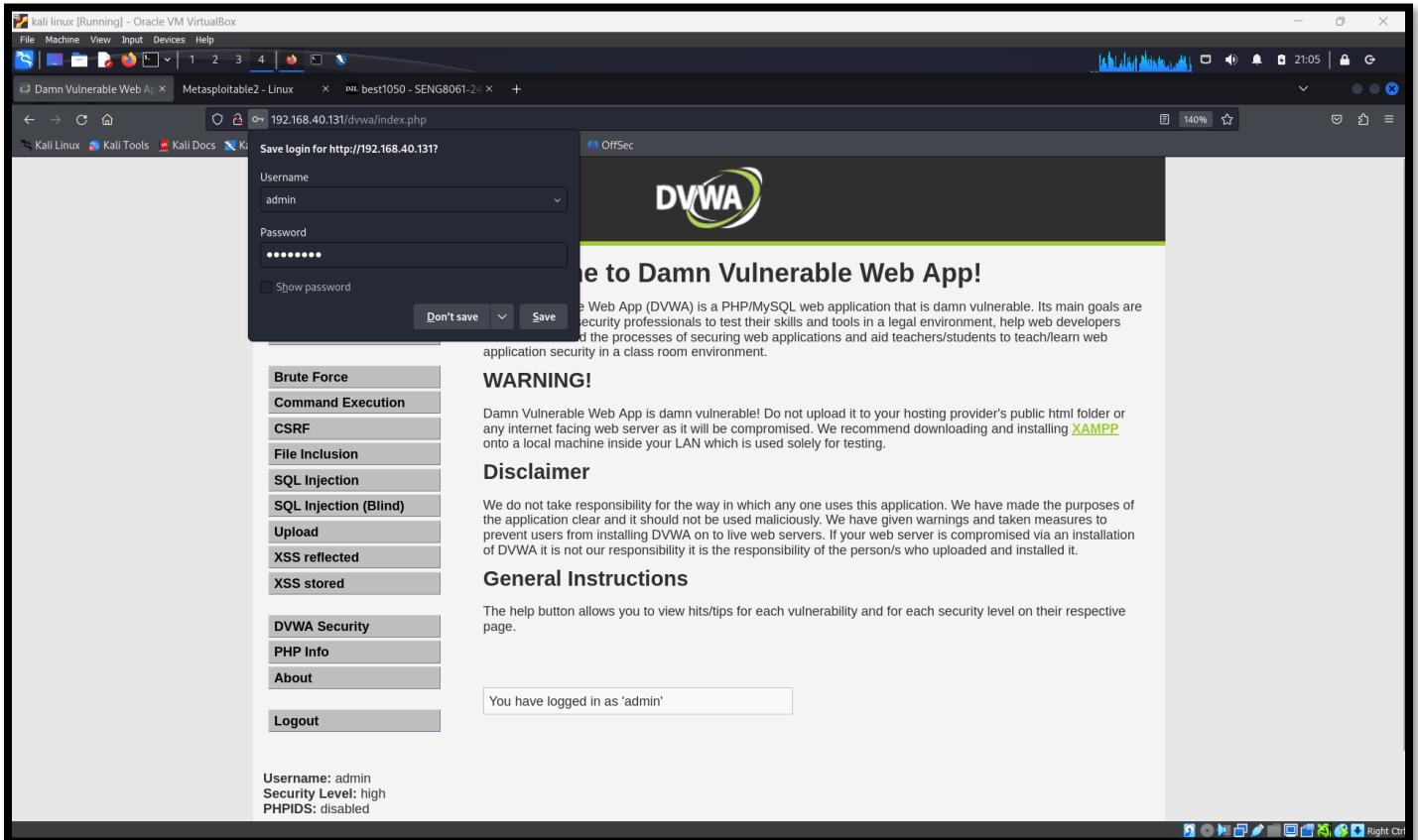
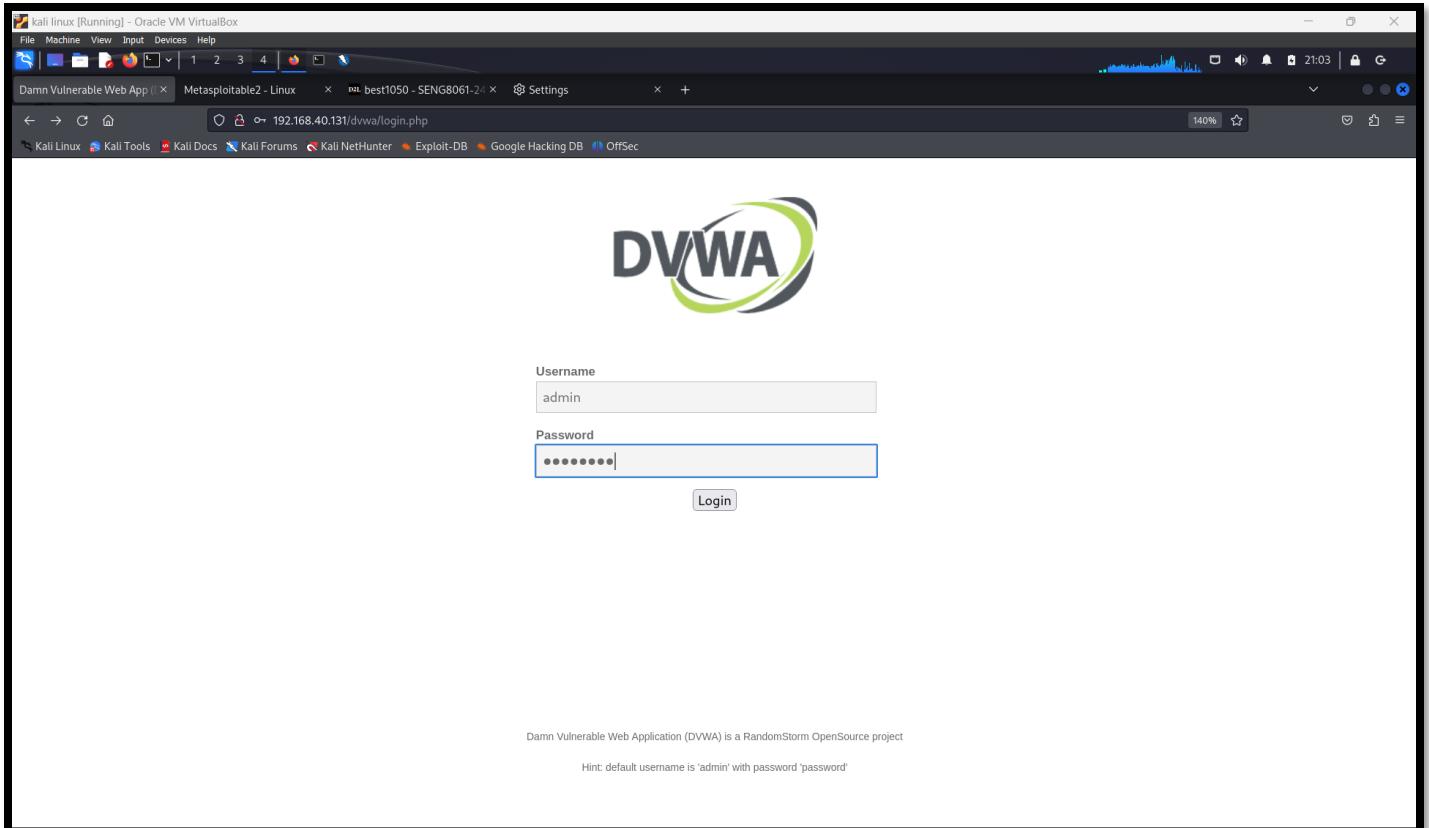
| Task ID | Message Type | Code | Reason | RTT | Size Resp. Header | Size Resp. Body | Highest Alert | State | Payloads |
|---------|--------------|------|--------|-------|-------------------|-----------------|----------------|------------|----------|
| 1,021 | Fuzzed | 200 | OK | 74 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | www | |
| 698 | Fuzzed | 200 | OK | 55 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | password | |
| 690 | Fuzzed | 200 | OK | 56 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | pass | |
| 550 | Fuzzed | 200 | OK | 69 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | login | |
| 362 | Fuzzed | 200 | OK | 79 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | enter | |
| 328 | Fuzzed | 200 | OK | 68 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | default | |
| 114 | Fuzzed | 200 | OK | 33 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | admin | |
| 113 | Fuzzed | 200 | OK | 72 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | action | |
| 8 | Fuzzed | 200 | OK | 52 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | 1 | |
| 2 | Fuzzed | 200 | OK | 69 ms | 291 bytes | 1,289 bytes | 反映 (Reflected) | 0 | |
| 0 | Original | 302 | Found | 18 ms | 335 bytes | 0 bytes | | | |
| 1 | Fuzzed | 302 | Found | 54 ms | 335 bytes | 0 bytes | | | |
| 3 | Fuzzed | 200 | OK | 66 ms | 291 bytes | 1,289 bytes | | 00000 | |
| 4 | Fuzzed | 200 | OK | 64 ms | 291 bytes | 1,289 bytes | | 000000 | |
| 5 | Fuzzed | 200 | OK | 20 ms | 291 bytes | 1,328 bytes | | 00000000 | |
| 6 | Fuzzed | 200 | OK | 47 ms | 291 bytes | 1,289 bytes | | | |
| 7 | Fuzzed | 200 | OK | 48 ms | 291 bytes | 1,289 bytes | | 0987654321 | |
| 9 | Fuzzed | 200 | OK | 48 ms | 291 bytes | 1,289 bytes | | 1111 | |
| 10 | Fuzzed | 200 | OK | 52 ms | 291 bytes | 1,289 bytes | | 11111 | |

Alerts | Main Proxy: localhost:8080 | Current Scans

After completing fuzzer, we filter out state by clicking on it, we got some response.

| Fuzzer Progress: 0: HTTP - http://192.168.1.100/dvwa/login.php | | | | | | | | | | 100% | Current fuzzers: 0 |
|--|--------------|-----------|--------|-------|-------------------|-----------------|---------------|----------|--|-------------|--------------------|
| Messages Sent: 1049 Errors: 0 | | | | | | | | | | Show Errors | Export |
| Task ID | Message Type | Code | Reason | RTT | Size Resp. Header | Size Resp. Body | Highest Alert | State | | Payloads | |
| 1,021 Fuzzed | | 200 OK | | 74 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | www | | | |
| 698 Fuzzed | | 200 OK | | 55 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | password | | | |
| 690 Fuzzed | | 200 OK | | 56 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | pass | | | |
| 550 Fuzzed | | 200 OK | | 69 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | login | | | |
| 362 Fuzzed | | 200 OK | | 79 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | enter | | | |
| 328 Fuzzed | | 200 OK | | 68 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | default | | | |
| 114 Fuzzed | | 200 OK | | 33 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | admin | | | |
| 113 Fuzzed | | 200 OK | | 72 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | action | | | |
| 8 Fuzzed | | 200 OK | | 52 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | 1 | | | |
| 2 Fuzzed | | 200 OK | | 69 ms | 291 bytes | 1,289 bytes | ⚠️ Reflected | 0 | | | |
| 0 Original | | 302 Found | | 18 ms | 335 bytes | 0 bytes | | | | | |
| 1 Fuzzed | | 302 Found | | 54 ms | 335 bytes | 0 bytes | | | | ----- | |

we use those response as our input to admin password.



Finally, we have successfully bypassed the authentication and logged in as admin with password = password., without any secure authorization.

This vulnerability shows that there are weak authentication processes in the DVWA application, which gives attackers the chance to obtain unauthorized access through brute force attacks.

Recommendations/ Mitigations:

- **Put CAPTCHA into the application:** This will allow the application to authenticate that a human user is attempting to log in, preventing automated brute-force attacks.
- **Limit The Number of Logins:** To prevent attackers from continually guessing passwords, set a limit on the number of failed login attempts within a specific time range.
- **Implement Two-Factor Authentication:** The security of the login process can be greatly improved by requiring an additional method of authentication, such as a one-time password given by SMS or produced by an authentication app such as Microsoft Authenticator, Google Authenticator

Risk Analysis

For the DREAD analysis we have used rank criteria as below:

DREAD Ranking:

High (40-50),

Medium (30-39),

Low (20-29)

| DREAD | Detail | Rank |
|-----------------|--|------------------------|
| Damage | This vulnerability can cause damage as it allows unauthorized access to critical areas of the application, ultimately leading to data breaches, modification of sensitive information, and unauthorized actions. | 9/10 |
| Reproducibility | The vulnerability appears to be highly reproducible because it is built within the application, allowing anyone to access it without authorization. | 10/10 |
| Exploitability | Exploiting this vulnerability is relatively easy, as demonstrated by using OWASP Zap to perform a brute force attack, which successfully bypassed the authentication mechanism. | 10/10 |
| Affected Users | This vulnerability potentially affects all users of the DVWA application since it compromises system security and allows unauthorized access. | 10/10 |
| Discoverability | This vulnerability is easy to find because it was found using basic testing and exploitation methods like brute force attacks using frequently used passwords. | 10/10 |
| DREAD SCORE | (Risk Level: HIGH) | Overall score 49/50 |

Vulnerability #2: File Inclusion

Discovered Vulnerabilities

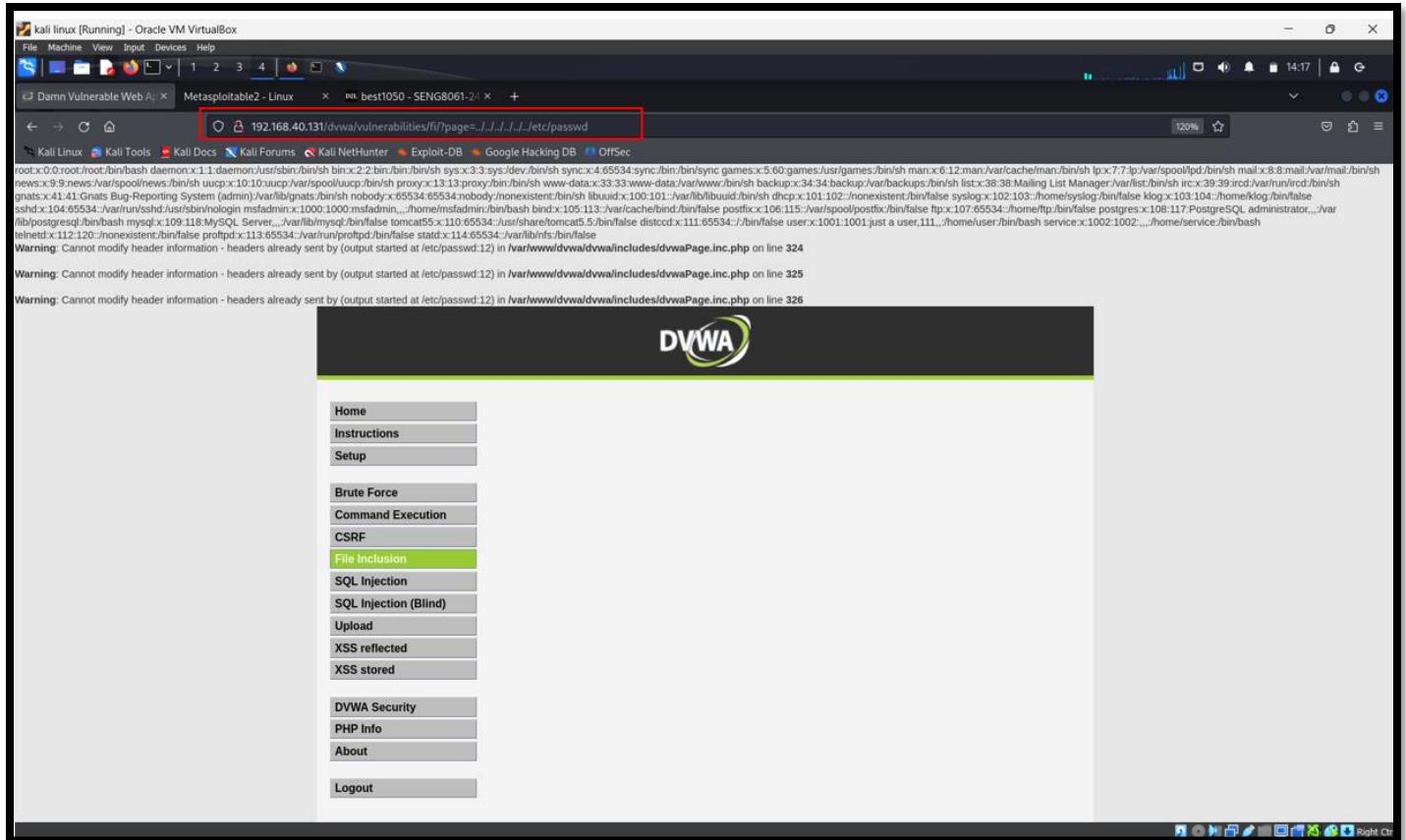
We logged in as admin to DVWA page, then we went to DVWA security, where we select 'LOW' as DVWA vulnerability level and submit the button. Then we went to file inclusion tab.

The screenshot shows a Kali Linux desktop environment with a browser window open to the DVWA security page at <http://192.168.40.131/dvwa/security.php>. The DVWA logo is at the top. On the left is a sidebar menu with the following items:

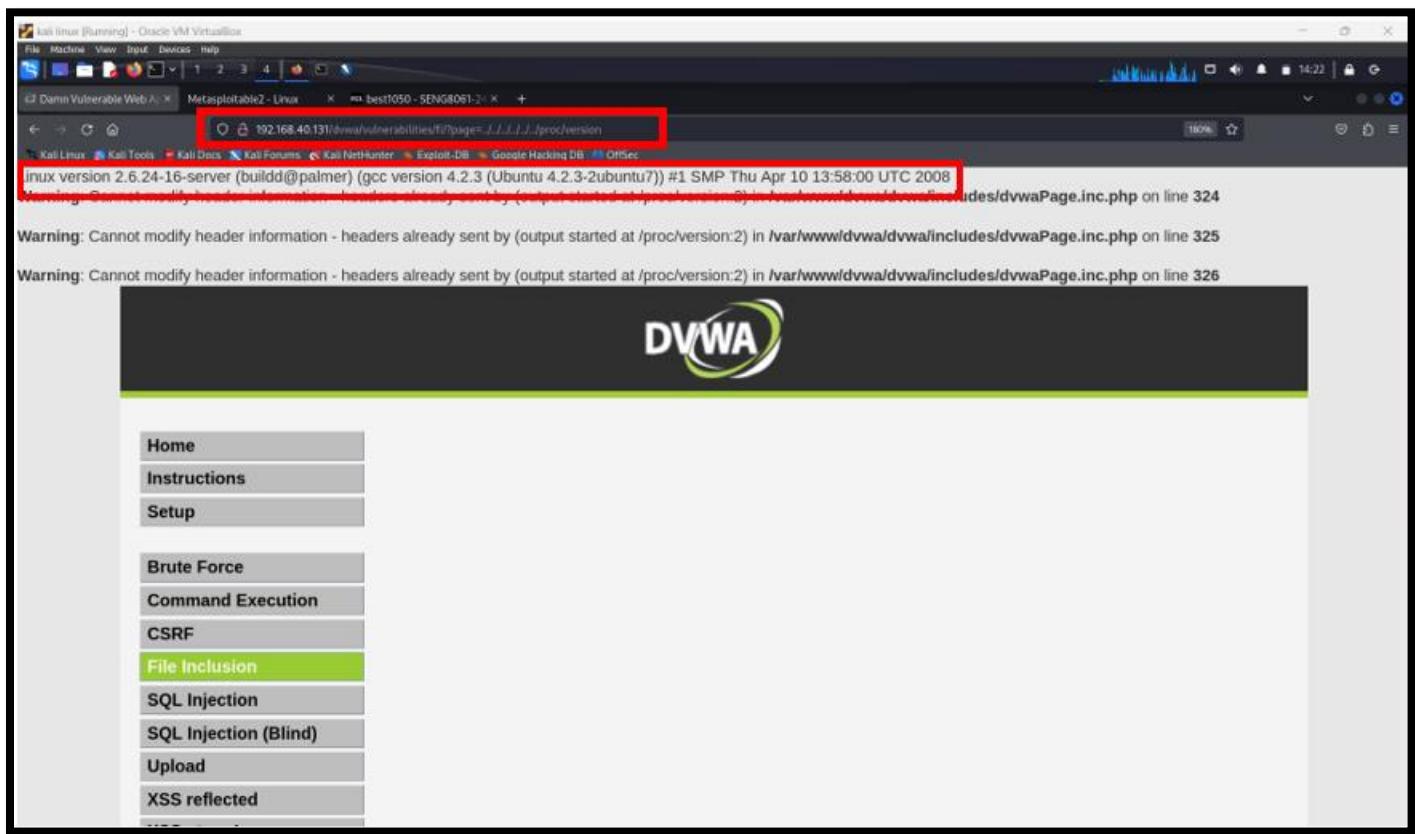
- Home
- Instructions
- Setup
- Brute Force
- Command Execution
- CSRF
- File Inclusion** (highlighted with a red box)
- SQL Injection
- SQL Injection (Blind)
- Upload
- XSS reflected
- XSS stored
- DVWA Security (highlighted with a green box)
- PHP info
- About
- Logout

The main content area displays the DVWA Security page. It shows the current security level is 'low'. A dropdown menu is set to 'low' and has a 'Submit' button next to it. Below this is a section for PHPIDS, which is currently disabled. It includes links for enabling PHPIDS and viewing the IDS log. At the bottom of the main content area, a message box contains the text 'Security level set to low'.

Then we typed on URL `../../../../etc/passwd` which is useful for finding user's information in Linux, then we run the browser. Here we saw that the file included as a php script.



Then we typed change that url path to these path `../../../../proc/version`, we saw find version of the web browser on the page.



Recommendations/ Mitigations:

- Validate and sanitize user input before using it to include files.
- Maintain a whitelist of acceptable inputs.
- When including files, use absolute paths rather than user-controlled relative paths.
- Ensure that the web server process has minimal permissions and can only access necessary files.

Risk Analysis

For the DREAD analysis we have use rank criteria as below:

DREAD Ranking:

High (40-50),

Medium (30-39),

Low (20-29)

| DREAD | Detail | Rank |
|-----------------|---|------------------------|
| Damage | The potential damage of this vulnerability is significant as it allows an attacker to access sensitive system files like /etc/passwd and /proc/version, which can reveal system and user information. | 7/10 |
| Reproducibility | The vulnerability appears to be highly reproducible as it can be exploited by manipulating the URL to include arbitrary files. | 9/10 |
| Exploitability | Exploiting this vulnerability seems relatively straightforward as demonstrated by accessing sensitive system files through URL manipulation. | 7/10 |
| Affected Users | All users accessing the vulnerable file inclusion feature are potentially affected, as the vulnerability allows attackers to access system files regardless of user authentication. | 7/10 |
| Discoverability | Discovering this vulnerability might require some knowledge about file system structure and URL manipulation techniques, but once understood, it's relatively easy to exploit. | 9/10 |
| DREAD SCORE | (Risk Level: Medium) | Overall score 39/50 |

Vulnerability #3: Cross-Site Scripting (XSS) Reflection

Discovered Vulnerabilities

We opened the DVWA site and set the difficulty level to low from the DVWA security tab.

A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL "192.168.40.131/dvwa/security.php". The page title is "DVWA Security" with a lock icon. On the left, a sidebar menu lists various security modules: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The "XSS reflected" option is highlighted in green. The main content area displays the "Script Security" section, stating "Security Level is currently **low**". It includes a dropdown menu set to "low" and a "Submit" button. Below this is the "PHPIDS" section, which is currently disabled. A link to "enable PHPIDS" is provided, along with links to "Simulate attack" and "View IDS log". A message box at the bottom states "Security level set to low".

A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL "192.168.40.131/dvwa/vulnerabilities/xss_r/". The page title is "DVWA". The main content area displays the "Vulnerability: Reflected Cross Site Scripting (XSS)" section. A form asks "What's your name?" with an input field containing "Shivani Varu" and a "Submit" button. The left sidebar menu is identical to the previous screenshot, with the "XSS reflected" option highlighted in green.

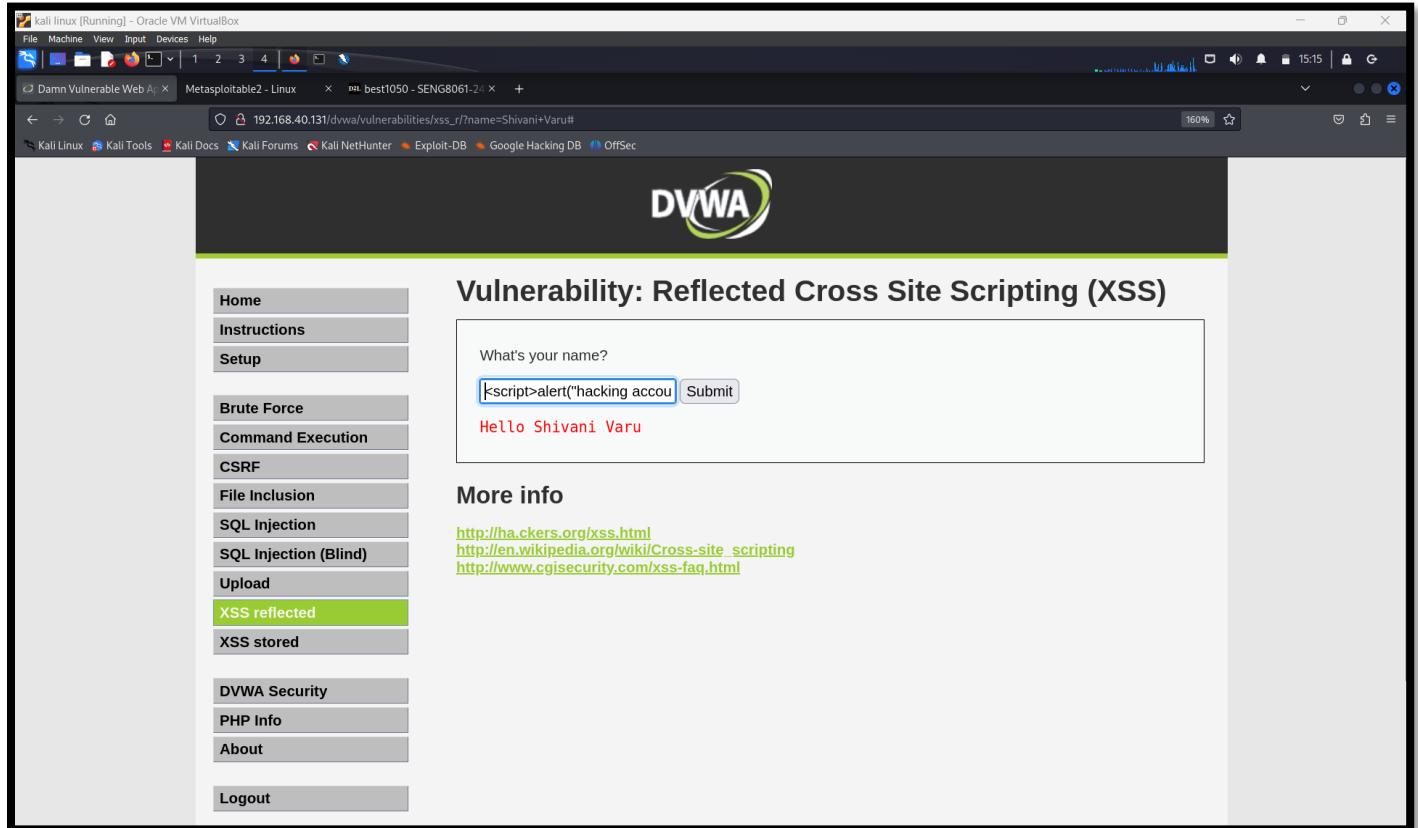
Then, we entered some value in the search field and clicked on submit.

The screenshot shows a browser window with the URL `192.168.40.131/dvwa/vulnerabilities/xss_r/?name=Shivani+Varu#`. The DVWA logo is at the top. On the left, a sidebar menu includes 'XSS reflected' which is highlighted in green. The main content area has a form asking 'What's your name?' with a red 'Submit' button. Below the form, the text 'Hello Shivani Varu' is displayed in red, indicating the user input was reflected back to the user.

We saw the entered value in the source page.

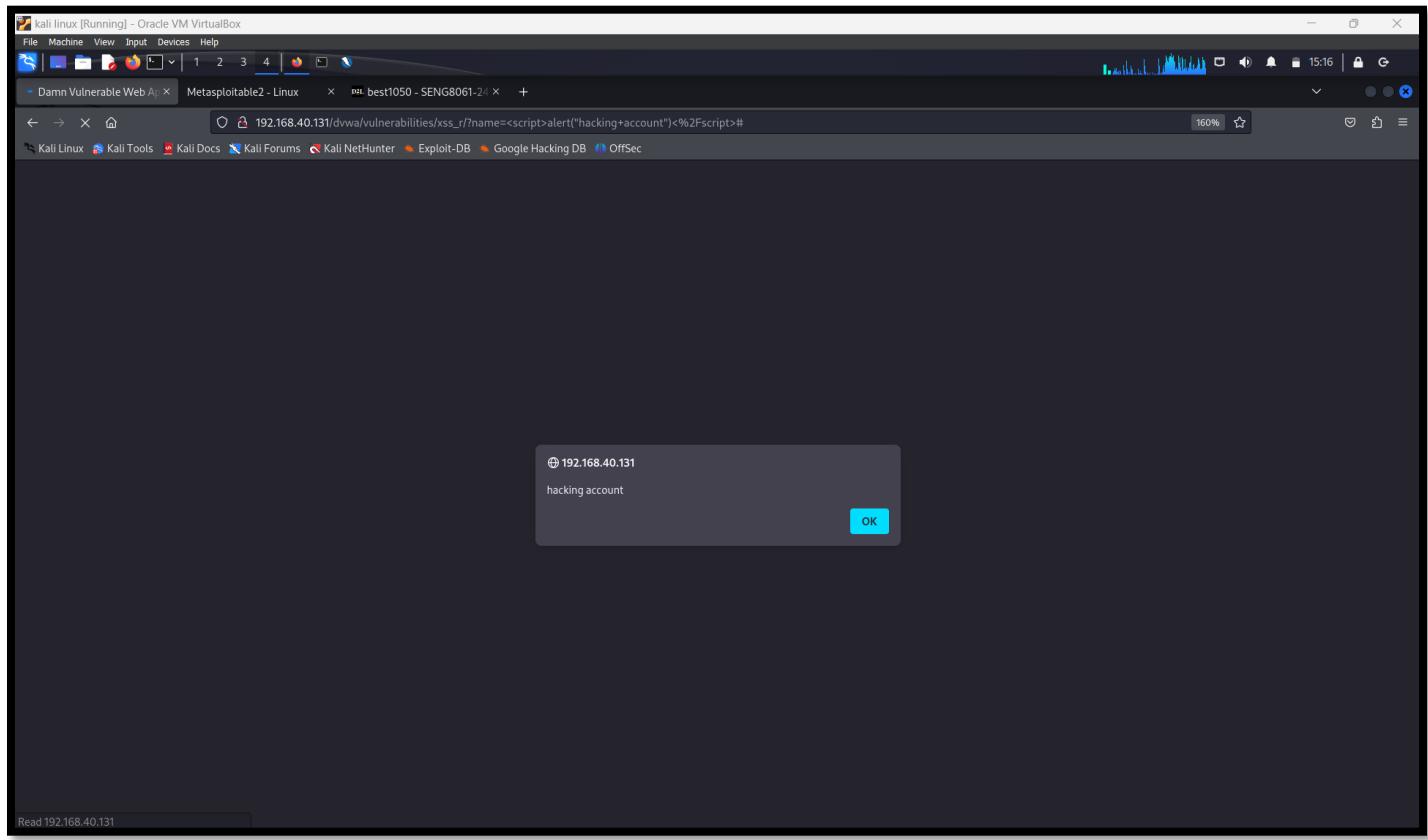
The screenshot shows the Chrome DevTools Inspector with the 'Elements' tab selected. The 'Search HTML' field contains the string 'Hello Shivani Varu'. The DOM tree shows the injected payload in the `<pre>` tag of the reflected XSS form. The right-hand panel displays the CSS styles applied to the element, including the color red, and the Box Model properties, showing dimensions of 611.733x15 pixels.

We provided the script <script> alert("hacking account") </script> in the field and clicked on submit.



A screenshot of a web browser window showing the DVWA (Damn Vulnerable Web Application) interface. The main page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". On the left, there's a sidebar menu with various exploit categories: Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected (which is highlighted in green), XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area has a form asking "What's your name?". Inside the input field, the user has typed "<script>alert('hacking account')</script>". Below the input field is a "Submit" button. The output of the script is displayed as "Hello Shivani Varu" in red text. At the bottom left of the main content area, there's a section titled "More info" with three links: <http://ha.ckers.org/xss.html>, http://en.wikipedia.org/wiki/Cross-site_scripting, and <http://www.cgisecurity.com/xss-faq.html>.

As a result, a black pop-up with the given information appeared, representing exploitation.



A screenshot of a web browser window showing a JavaScript alert dialog box. The dialog box has a dark gray background and contains the text "192.168.40.131" and "hacking account" in white. At the bottom right of the dialog is a blue "OK" button. The browser's address bar shows the URL "192.168.40.131/dvwa/vulnerabilities/xss_r/?name=<script>alert('hacking+account')<%2Fscript>#". The status bar at the bottom left of the browser window shows the text "Read 192.168.40.131".

Then, we also provided the script <script> alert(document.cookie) </script> to find the session ID.

A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL: "192.168.40.131/dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)<%2Fscript>#". The page title is "Vulnerability: Reflected Cross Site Scripting (XSS)". On the left, there is a sidebar menu with various options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected (which is highlighted in green), XSS stored, DVWA Security, PHP Info, About, and Logout. The main content area has a form field asking "What's your name?" with the value "<script>alert(document.cookie)" and a "Submit" button. Below the form, the text "Hello" is displayed. At the bottom, there is a "More info" section with links to external resources: <http://ha.ckers.org/xss.html>, http://en.wikipedia.org/wiki/Cross-site_scripting, and <http://www.cgisecurity.com/xss-faq.html>.

We observed that the session ID was retrieved. We can see the session ID is retrieved.

A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL: "192.168.40.131/dvwa/vulnerabilities/xss_r/?name=<script>alert(document.cookie)<%2Fscript>#". A modal dialog box is displayed in the center of the screen with the following content:
① 192.168.40.131
security=low; PHPSESSID=4ae85fc15a8d0de936fc8264728a14c3
OK

Recommendations/ Mitigations:

- Implement strict input validation and sanitization to ensure that user-supplied data is properly filtered, removing or encoding any potentially dangerous characters or script tags.
- Use output encoding when returning user input in HTML to ensure that any potentially malicious characters are properly encoded.
- Implement a Content Security Policy (CSP) to restrict the sources from which scripts can be loaded.
- Use HTTPOnly cookies to prevent scripts from accessing sensitive cookie data.

Risk Analysis

For the DREAD analysis we have use rank criteria as below:

DREAD Ranking:

High (40-50),

Medium (30-39),

Low (20-29)

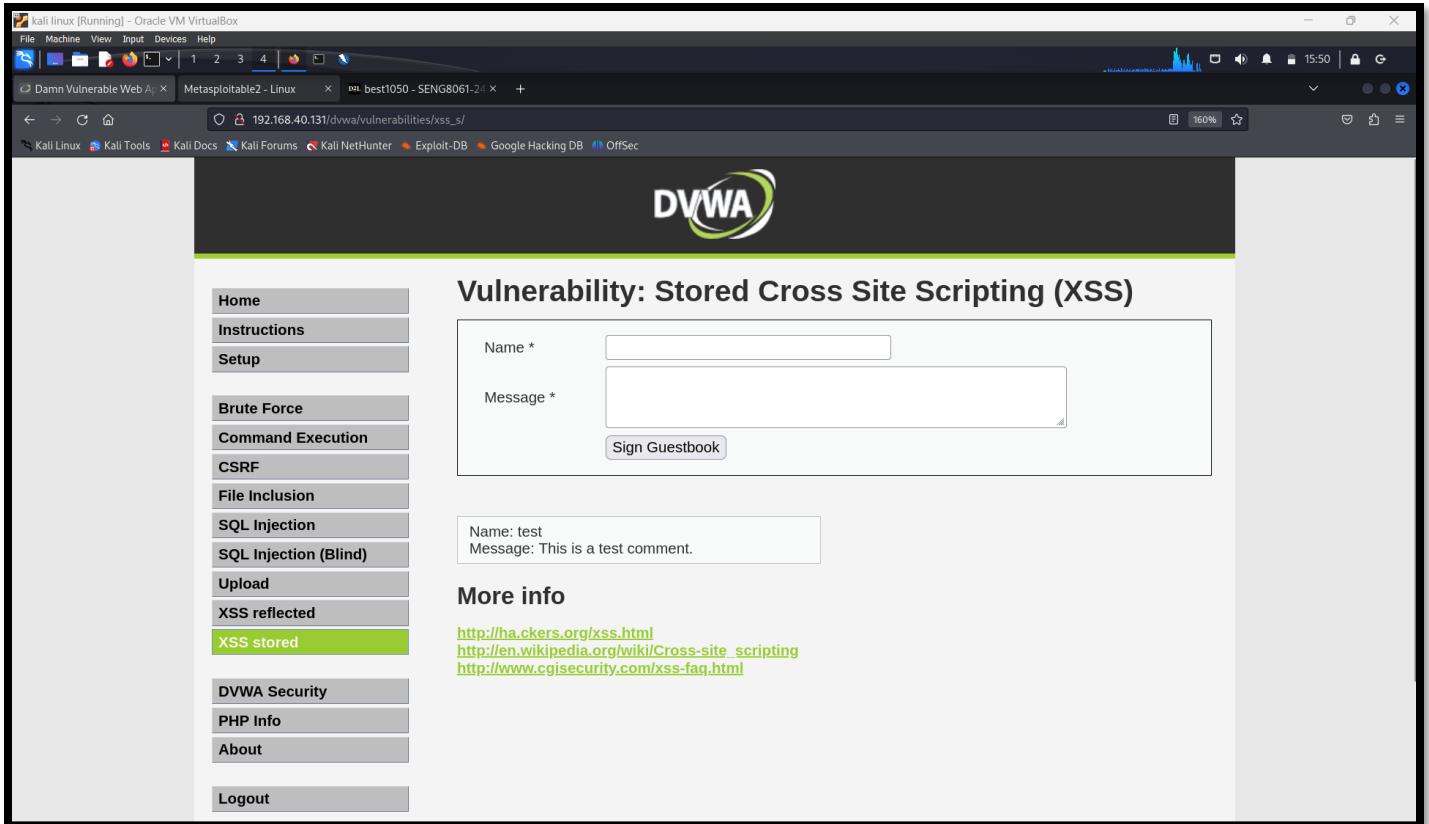
| DREAD | Detail | Rank |
|-----------------|---|---------------------|
| Damage | An attacker could be able to run arbitrary scripts within the sessions of other users, which makes the XSS vulnerability extremely dangerous. This could result in various consequences, such as stealing sensitive information (such as session IDs or cookies), performing on the user's behalf, or damaging the website. | 9/10 |
| Reproducibility | The vulnerability is highly reproducible, as it appears to occur regularly whenever user-supplied data is sent to the web page without enough validation and sanitization. | 10/10 |
| Exploitability | Simple scripts such as alert boxes can be injected to exploit this vulnerability, which appears to be quite straightforward. It can be used for more malicious activities in the future. | 9/10 |
| Affected Users | All users who interact with the vulnerable function can be affected, especially if they provide input that is reflected, they return to the page. | 8/10 |
| Discoverability | This vulnerability appears to be very straightforward to discover, having been discovered through manual testing by entering scripts into the input field and observing the results. | 8/10 |
| DREAD SCORE | (Risk Level: High) | Overall score 44/50 |

Vulnerability #4: Cross-Site Scripting (XSS) Stored

We opened the DVWA site and set the difficulty level to low.

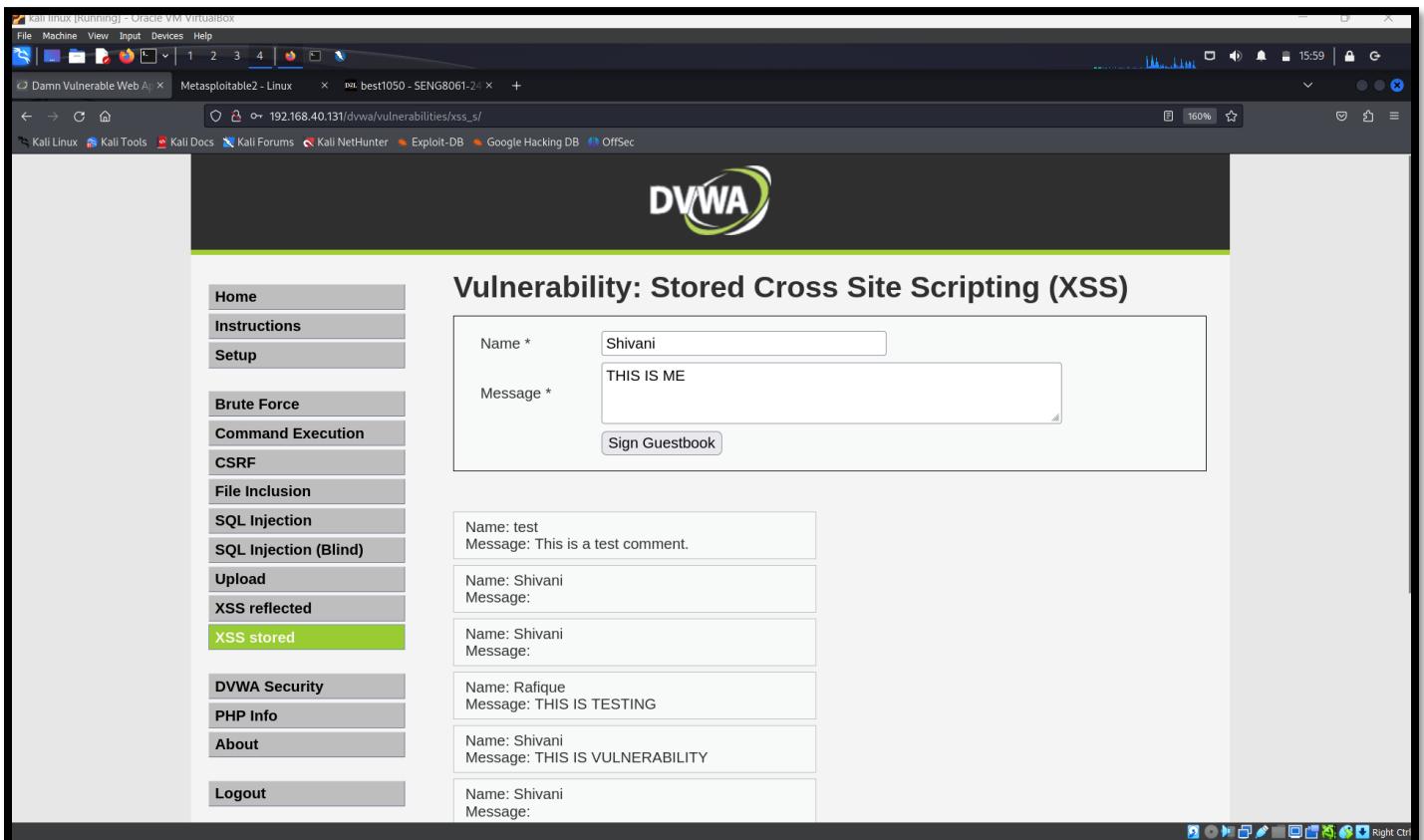
The screenshot shows a Firefox browser window running on a Kali Linux host. The address bar indicates the URL is `192.168.40.131/dvwa/security.php`. The DVWA logo is at the top. The main content area displays the "DVWA Security" page with a "Script Security" section. It shows the security level is currently "low". A dropdown menu allows changing the security level to "medium" or "high", with "Submit" and "Cancel" buttons. Below this is a "PHPIDS" section, which is currently disabled. A note says "You can enable PHPIDS across this site for the duration of your session." There is a link to "enable PHPIDS". At the bottom, a message box says "Security level set to low". On the left sidebar, under the "DVWA Security" heading, the "XSS stored" option is highlighted. Other options include Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and Logout.

After that, we went to XSS stored from navigation bar,



A screenshot of a web browser window showing the DVWA (Damn Vulnerable Web Application) interface. The title bar reads "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL "192.168.40.131/dvwa/vulnerabilities/xss_s/". The main content area displays the "Vulnerability: Stored Cross Site Scripting (XSS)" page. On the left, there is a sidebar menu with various options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The "XSS stored" option is highlighted. The main form has fields for "Name *" and "Message *", both of which contain user input. Below the form, a message box shows the submitted data: "Name: test" and "Message: This is a test comment.". A "Sign Guestbook" button is also present. The DVWA logo is at the top right.

where we entered some values in the name and message fields and clicked on submit.



A screenshot of a web browser window showing the DVWA (Damn Vulnerable Web Application) interface. The title bar reads "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL "192.168.40.131/dvwa/vulnerabilities/xss_s/". The main content area displays the "Vulnerability: Stored Cross Site Scripting (XSS)" page. On the left, there is a sidebar menu with various options like Home, Instructions, Setup, Brute Force, Command Execution, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, and XSS stored. The "XSS stored" option is highlighted. The main form has fields for "Name *" and "Message *", both of which contain user input. Below the form, a message box shows the submitted data: "Name: Shivani" and "Message: THIS IS ME". A "Sign Guestbook" button is also present. To the right of the form, there is a list of previous entries from other users. The DVWA logo is at the top right.

Upon inspecting the source code, we confirmed that the entered data was indeed present.

The screenshot shows a browser window with the URL `192.168.40.131/dvwa/vulnerabilities/xss_s/`. The page displays a guestbook form with several entries. The 'Message' field of the last entry contains the script `<script> alert("ethical hacking") </script>`. The browser's developer tools are open, specifically the 'Elements' tab under 'Inspector'. The 'Script' section of the element tree is selected, highlighting the injected script. The right-hand panel shows the CSS styles for the main body and the container div, including font sizes and colors.

```

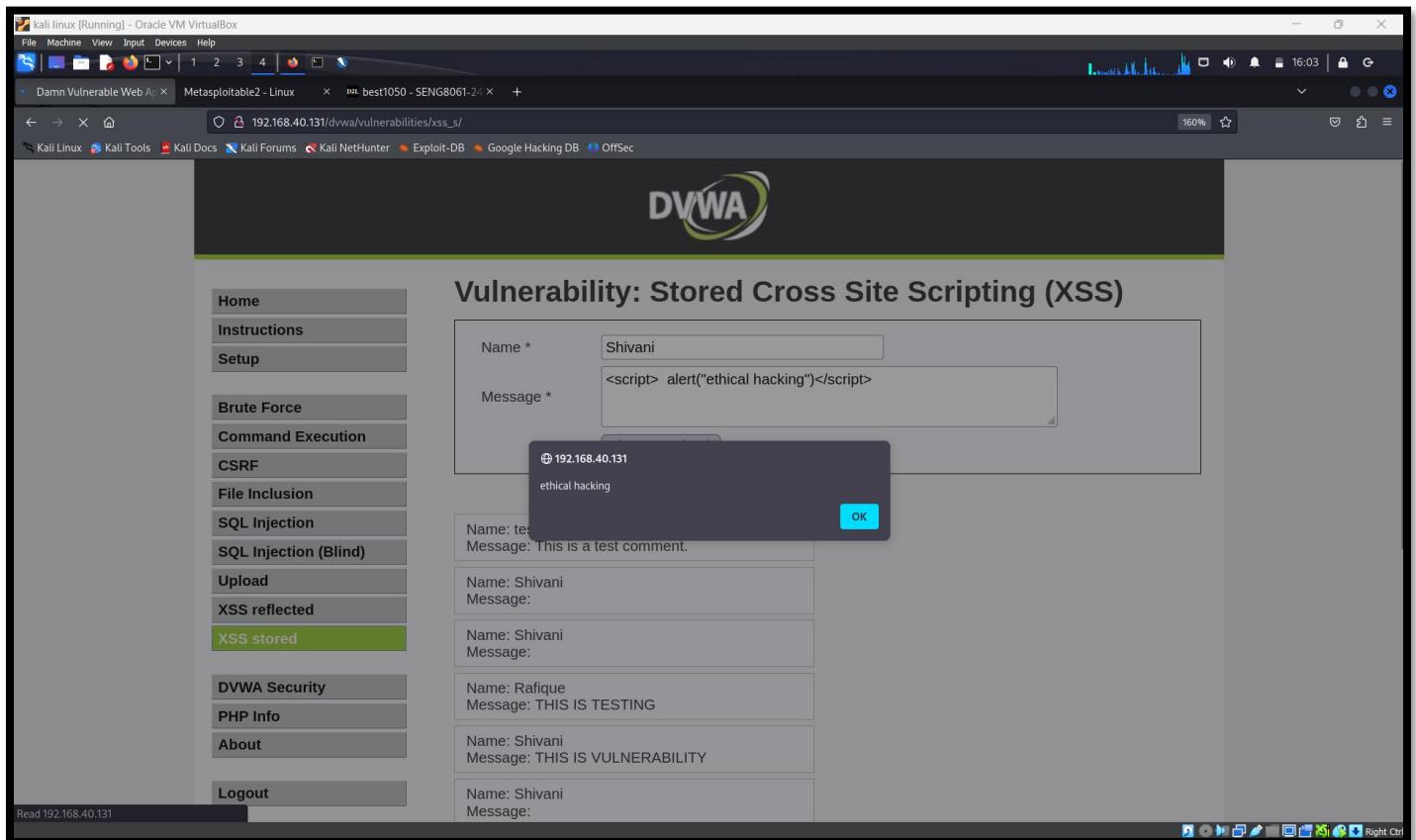
<!--VULNERABILITY: Stored Cross Site Scripting (XSS)-->
<div class="vulnerable_code_area"></div>
<br>
<div id="guestbook_comments">
  Name: test
  <br>
  Message: This is a test comment.
<br>
</div>
<div id="guestbook_comments">
  Name: Shivani
  <br>
  Message:
  <script>("ethical hacking")</script>
  <br>
</div>
<div id="guestbook_comments">
  Name: Shivani
  <br>
  Message:
  <script>("ethical hacking")</script>
  <br>
</div>
<div id="guestbook_comments">
  Name: Shivani
  <br>
  Message:
  <script>("ethical hacking")</script>
  <br>
</div>
<div id="guestbook_comments">
  Name: Rafique
  <br>
  Message: THIS IS TESTING
  <br>
</div>
<div id="guestbook_comments">
  Name: Shivani
  <br>
  Message: THIS IS VULNERABILITY
  <br>
</div>
<div id="guestbook_comments">
  Name: Shivani
  <br>
  Message:
  <script>("ethical hacking")</script>
  <br>
</div>

```

Subsequently, we entered the script `<script> alert("ethical hacking") </script>` in the message field and clicked on submit.

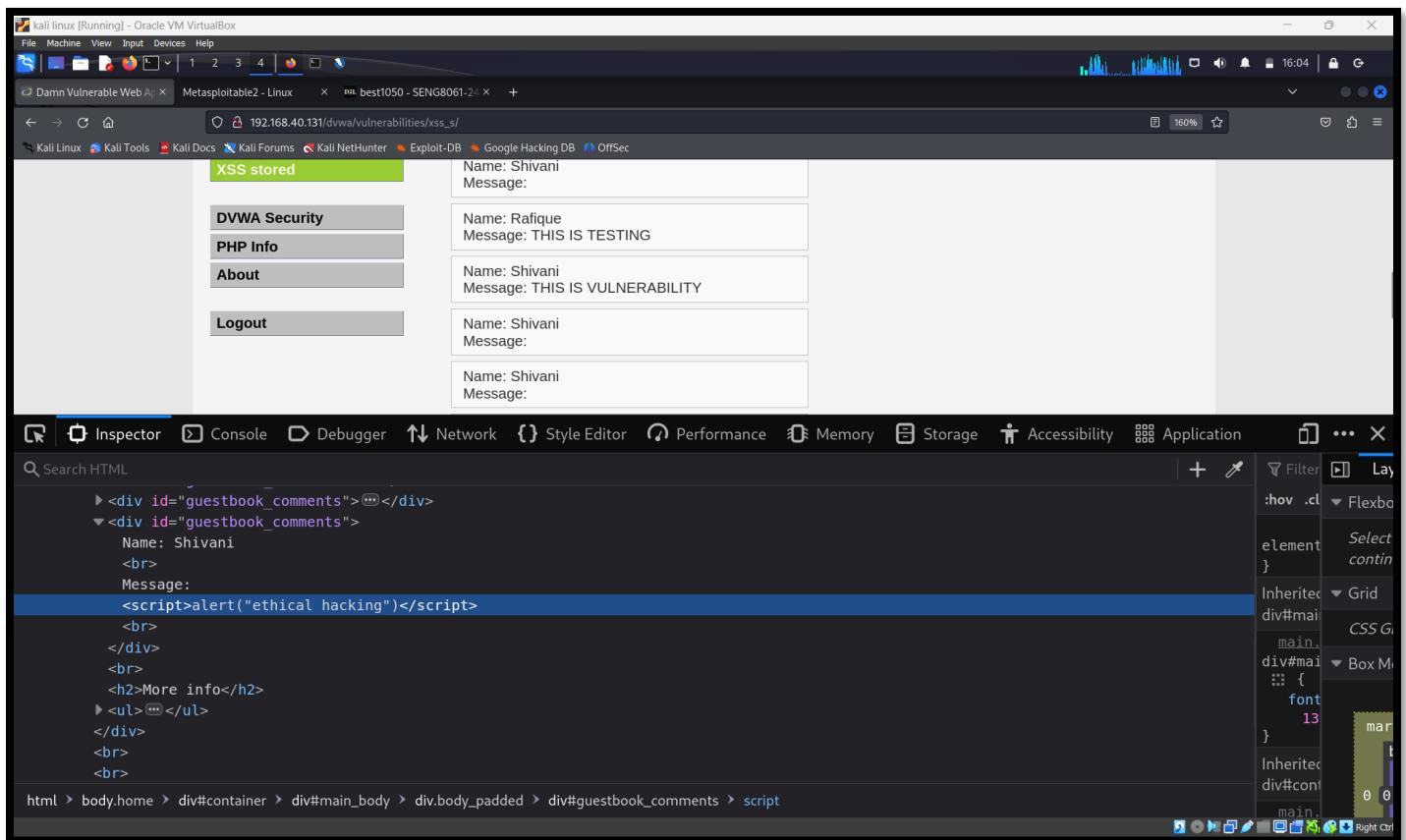
The screenshot shows the DVWA 'XSS stored' page. On the left, a sidebar menu lists various vulnerabilities, with 'XSS stored' currently selected. The main content area displays a form for signing a guestbook. In the 'Message' field, the user has entered the script `<script> alert("ethical hacking") </script>`. Below the form, a list of previous guestbook entries shows the injected script being executed, resulting in an alert box for each entry. The DVWA logo is visible at the top of the page.

Instantly, a pop-up emerged, indicating that exploitation had been successful.



A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL "192.168.40.131/dvwa/vulnerabilities/xss_s/". The main content area displays the DVWA logo and the title "Vulnerability: Stored Cross Site Scripting (XSS)". On the left, a sidebar menu lists various attack types, with "XSS stored" highlighted in green. The main form has fields for "Name" (containing "Shivani") and "Message" (containing "<script> alert('ethical hacking')</script>"). Below the form, a success message box is open, showing the message "ethical hacking" and an "OK" button. The status bar at the bottom left reads "Read 192.168.40.131".

Furthermore, upon examining the source code, we could see that the script we entered was indeed present in the file.



A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows the URL "192.168.40.131/dvwa/vulnerabilities/xss_s/". The main content area displays the DVWA logo and the title "Vulnerability: Stored Cross Site Scripting (XSS)". On the left, a sidebar menu lists various attack types, with "XSS stored" highlighted in green. The main form has fields for "Name" (containing "Shivani") and "Message" (containing "Message:"). Below the form, a success message box is open, showing the message "ethical hacking" and an "OK" button. The status bar at the bottom left reads "Read 192.168.40.131". At the bottom, a developer tools interface (likely Chrome DevTools) is visible, showing the inspected HTML code. The "script" tag containing the XSS payload ("<script>alert('ethical hacking')</script>") is highlighted in blue, indicating it is selected in the DOM tree. The right side of the developer tools shows the CSS and styles applied to the element.

Recommendation/Mitigation:

- Organization should always encrypt the output our application sends to a browser when it uses user input so that the browser interprets it as data rather than as HTML or JavaScript code. This stops the execution of any script.
- When a user types <script> alert("ethical hacking") </script>, for example, the browser should display the text exactly as it was typed, without interpreting it as a script.
- Encoding converts HTML characters with meanings into character entities, which the browser displays as characters rather than code.

Risk Analysis

For the DREAD analysis we have use rank criteria as below:

DREAD Ranking:

High (40-50),

Medium (30-39),

Low (20-29)

| DREAD | Detail | Rank |
|-----------------|---|---------------------|
| Damage | XSS vulnerabilities can potentially lead to various types of damage, including stealing user credentials, session hijacking, defacement of the website, or spreading malware. In this case, the attacker was able to execute arbitrary scripts on the webpage, demonstrating a high potential for damage. | 8/10 |
| Reproducibility | The vulnerability appears to be easily reproducible, as the attacker was able to inject a script and trigger it successfully. | 8/10 |
| Exploitability | The exploitability of XSS vulnerabilities is typically high, as attackers can easily inject malicious scripts through input fields and trick other users into executing them. In this case, the attacker was able to exploit the vulnerability by simply inputting a script into the message field. | 9/10 |
| Affected Users | This depends on the number of users of the vulnerable application. If it's a widely used application, the number of affected users could be high. | 7/10 |
| Discoverability | XSS vulnerabilities are commonly discovered through manual testing or automated scanning tools. In this case, the vulnerability was discovered during testing, indicating moderate discoverability. | 6/10 |
| DREAD SCORE | (Risk Level: Medium) | Overall score 38/50 |

Vulnerability #5: Command Execution

We opened the DVWA site and set the difficulty level to low. Then we went to Command Execution in the left navigation bar. We put 127.0.0.1 as input in the text field.

The screenshot shows a browser window on a Kali Linux host within Oracle VM VirtualBox. The URL in the address bar is `192.168.40.131/dvwa/vulnerabilities/exec/#`. The DVWA logo is at the top. On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, **Command Execution**, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The 'Command Execution' option is highlighted. The main content area is titled 'Vulnerability: Command Execution' and contains a 'Ping for FREE' section. It asks for an IP address and shows the results of a ping to 127.0.0.1. The output is:
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.822 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.090 ms
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.037/0.316/0.822/0.358 ms

Here, we got some response. After that we put pwd at the end of localhost URL and hit the submit button.

The screenshot shows the same DVWA setup as the previous one. The URL is again `192.168.40.131/dvwa/vulnerabilities/exec/#`. The 'Command Execution' section now shows the result of running the 'pwd' command. The output is:
127.0.0.1:pwd
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.822 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.037 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.090 ms
--- 127.0.0.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1998ms
rtt min/avg/max/mdev = 0.037/0.316/0.822/0.358 ms

We got the directory of this machine /var/www/dvwa/vulnerabilities/exec.

A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows "192.168.40.131/dvwa/vulnerabilities/exec/#". The main content is the DVWA logo and the title "Vulnerability: Command Execution". On the left, a sidebar menu lists various attack types: Home, Instructions, Setup, Brute Force, **Command Execution**, CSRF, File Inclusion, SQL Injection, SQL Injection (Blind), Upload, XSS reflected, XSS stored, DVWA Security, PHP Info, About, and Logout. The "Command Execution" item is highlighted. Below the menu, a section titled "Ping for FREE" contains a form with a text input field and a "submit" button. The input field contains "127.0.0.1". The output area shows the results of a ping command:

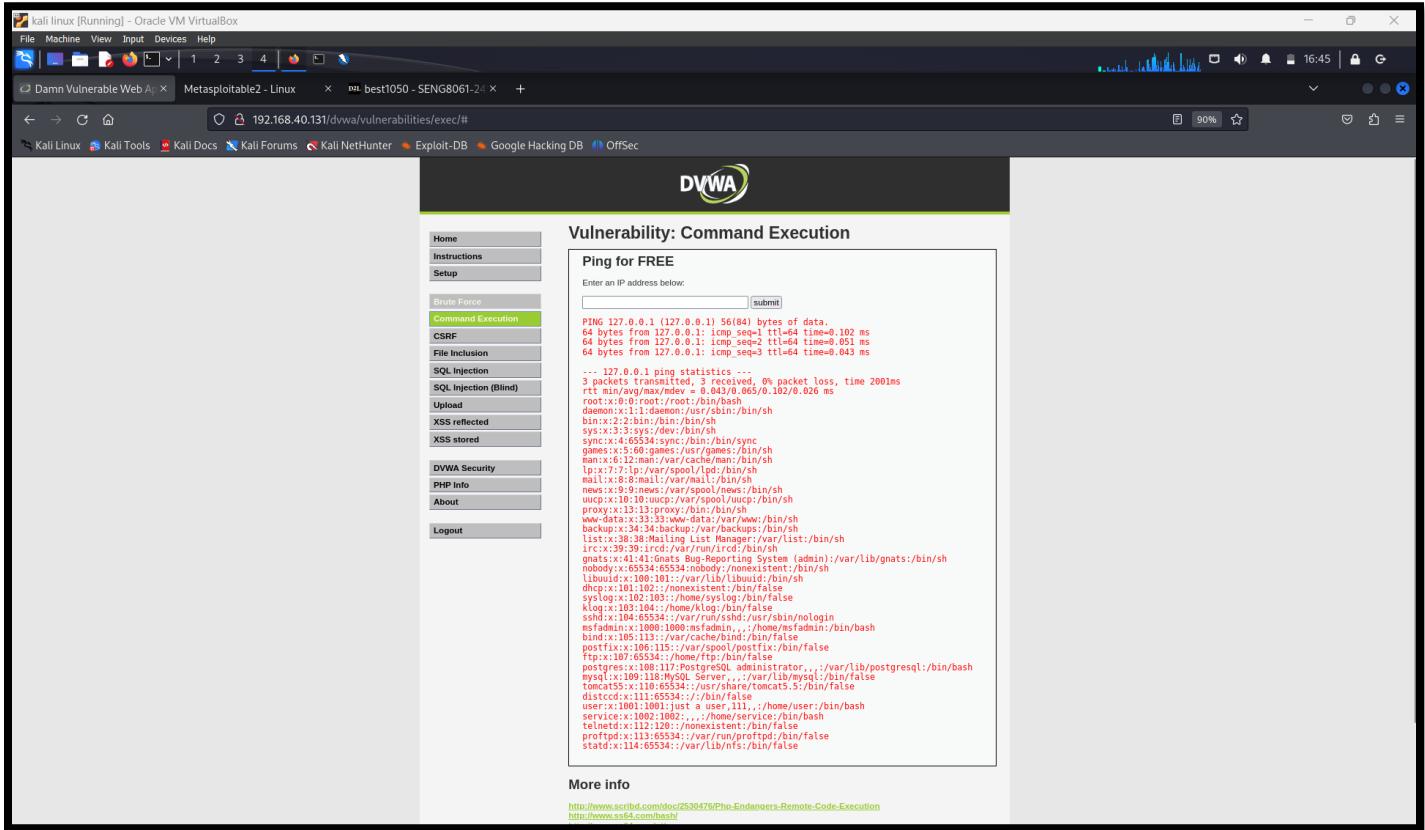
```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.127 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.138 ms  
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.076 ms  
  
--- 127.0.0.1 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 1999ms  
rtt min/avg/max/mdev = 0.076/0.113/0.138/0.029 ms  
/var/www/dvwa/vulnerabilities/exec
```

To make this attack a little more forward, we put ;cat /etc/passwd to see what happened.

A screenshot of a web browser window titled "kali linux [Running] - Oracle VM VirtualBox". The address bar shows "192.168.40.131/dvwa/vulnerabilities/exec/#". The main content is the DVWA logo and the title "Vulnerability: Command Execution". The sidebar menu is identical to the previous screenshot. The "Command Execution" item is highlighted. Below the menu, a section titled "Ping for FREE" contains a form with a text input field and a "submit" button. The input field contains "127.0.0.1;cat /etc/passwd". The output area shows the results of a ping command:

```
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.  
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.096 ms  
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.000 ms  
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.093 ms  
  
--- 127.0.0.1 ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 1999ms  
rtt min/avg/max/mdev = 0.000/0.063/0.096/0.044 ms
```

And we got the directory, which was ATC password files, so this was the vulnerability in low security.



Recommendation/Mitigation:

To fix such types of vulnerabilities, developer needs to follow below:

- Check all user inputs to ensure they follow the expected formats and ranges. Reject any input that appears to have malicious intent or special characters or commands.
- Sanitize user input by eliminating or encrypting special characters that could be interpreted as commands. To prevent injection attacks, use input filtering libraries or custom sanitization techniques.
- To minimize the potential damage of successful attacks, restrict the privileges of web application components. Run server processes with restricted permissions to avoid unwanted access to private information.
- Use secure APIs and libraries when performing commands or interacting with system resources. Keep the application safe from command injection vulnerabilities by avoiding using system calls or shell execution functions.

Risk Analysis

For the DREAD analysis we have use rank criteria as below:

DREAD Ranking:

High (40-50),

Medium (30-39),

Low (20-29)

| DREAD | Detail | Rank |
|-----------------|--|---------------------|
| Damage | This vulnerability allows attackers to execute commands on the server, which leads to data breaches, system compromise, and unauthorized access. Disclosing system files like "/etc/passwd" is a serious threat to security and privacy. | 9/10 |
| Reproducibility | An attacker can easily reproduce the vulnerability. They can put malicious commands into the input areas of the application and observe how the server responds. Because the vulnerability is built into the application's architecture and behavior, it is likely to be exploited consistently. | 10/10 |
| Exploitability | Exploiting this issue needs minimal effort and knowledge. Attackers simply require basic command injection knowledge and access to the vulnerable application. There are further tools and scripts available that can automate the exploitation process. | 9/10 |
| Affected Users | This vulnerability can cause damage to all users of the affected application. Any user interacting with the application's input fields—such as form inputs or search bars—has a chance to unintentionally cause the server to execute malicious commands. | 8/10 |
| Discoverability | The vulnerability is relatively easy to discover through manual testing or automated scanning techniques. Security researchers, penetration testers, and malicious actors can identify the vulnerability by inspecting the application's input validation mechanisms and observing its response to unexpected inputs | 8/10 |
| DREAD SCORE | (Risk Level: High) | Overall score 44/50 |

Video Recording Link:

<https://youtu.be/C1qeEhyQfI>