

Cybersecurity - Red Teaming Techniques

*Author* : Berrahia Hamssa oumaima

---

# *Ethical Cybersecurity Lab Report*

\_\_\_\_\_Academic Year : 2024-2025\_\_\_\_\_

**Subject:** Cybersecurity - Red Teaming Techniques

**Project :** Brute force attack Simulation in a Controlled lab Environment

# *Table of Contents*

1. Introduction .....	3
-----------------------	---

## *Theoretical part*

2. Historical Introduction .....	5
3. Types of Brute Force Attacks .....	6
4. Tools used .....	7

## *Practical Execution part*

5. Step 0: Reconnaissance & Wordlist Generation .....	9
6. Step 1: Environment Setup .....	11
7. Step 2: Launching the Attack .....	12

## *Defense Against Brute Force Attacks*

8. Instagram's Real-World Defensive Reactions .....	18
9. Conclusion .....	21

## *Introduction :*

As a Cybersecurity enthusiast, you have probably heard of an attack known as the **brute force attack** — a basic yet powerful technique used by attackers to guess and crack passwords, login credentials, and encryption keys. In this report, I simulate a brute force attack in a safe and controlled environment in order to explore how weak authentication systems can be exploited in real-world scenarios.

## *Objective :*

The main objective of this lab is to simulate a brute force attack in a virtual environment, includes a brief explanation of the theoretical concept behind the attack, the tools used to perform it, and common protection methods that can be applied to secure systems against such threats.

## *Environment Setup :*

- **Attacker Machine:** Kali Linux (VirtualBox)
- **Target Username:** testuser
- **Wordlist Used:** wordlist.txt

# *Theoretical part*

## -----Historical Introduction-----

The **brute force attack** is one of the oldest and most fundamental techniques in the history of Cryptography and Cybersecurity. Its origins can be traced back to the early days of encrypted communication, when attackers had no choice but to manually try every possible key or combination to decrypt a message.

The core idea — systematically attempting a large number of possible keys until the correct one is found — remains conceptually the same. With the rise of computers, however, brute force attacks became smarter, faster, and fully automated. Today, they pose a significant threat to the security of systems and sensitive information, especially when weak authentication mechanisms are in place.

- Given this background of brute force attacks, we must explore the different types and variations of this technique. While the basic principle remains the same — trying multiple combinations to gain access — attackers have developed various methods to optimize and adapt brute force strategies for different targets and scenarios.

## -----***TYPES OF BRUTE FORCE ATTACK***-----

Type	Definition	positives	negatives
<b><i>Dictionary</i></b>	Uses a predefined list of <b>common</b> or leaked passwords to guess the correct one.	Faster than pure brute force; targets <b>weak/common</b> passwords.	<b>Limited</b> to what's in the list; fails if password is unique.
<b><i>Hybrid</i></b>	<b>Combines</b> dictionary attack with variations - adding numbers or symbols-	More effective than dictionary alone.	<b>Slower</b> than dictionary; still misses complex passwords.
<b><i>Rainbow table</i></b>	Uses precomputed <b>hashes</b> to reverse-engineer password hashes.	Very fast <b>hash cracking</b> once table is ready.	Requires large storage, only works if <b>hash is unsalted</b> .
<b><i>Credential stuffing</i></b>	Reuses <b>real leaked</b> username-password combos on other services.	<b>High success rate</b> if reused credentials exist.	Depends on previous breaches, less effective with <b>2FA</b> .

### ***Additional explanation :***

**\*Salted Hash\***: A technique in which random data (a "salt") is added to passwords before hashing. This prevents attackers from using precomputed tables like rainbow tables.

**\*2FA\*** : Two-Factor Authentication

## -----*Tools Used*-----

- **Hydra**  
A fast and flexible brute force tool that supports various protocols such as SSH, FTP, HTTP, Telnet, and more.
- **Medusa**  
Similar to Hydra but known for speed and parallel testing. It supports modular login brute force across many services.
- **Ncrack**  
A high-speed network authentication cracking tool developed by the Nmap project. It focuses on scalability and performance.
- **John the Ripper**  
Mainly used for offline password cracking (especially hashes), but it can also perform dictionary attacks.
- **Patator**  
A multi-purpose brute-forcer with a lot of flexibility. It allows more customized and scriptable attacks.
- **Burp Suite (Intruder module)**  
Can be used for web-based brute force attacks on login forms, especially with custom payloads.
- **CrackMapExec**  
Useful for brute forcing SMB and RDP services in internal network environments.

*→ In the next section, I will experiment with some of these tools in a controlled lab environment to observe their behavior, speed, and effectiveness.*

## *Practical Execution part*



## -----Instagram-----

To simulate a real brute force attack in a safe and controlled environment, I used a script targeting an Instagram account I created specifically for testing purposes. Below are the steps followed during the process.

---

### Step 0: Reconnaissance & Wordlist Customization

The first step in almost any cyberattack is to **know the victim**—their personal information, lifestyle, and anything that might hold value. In this case, I simulated a typical attacker's mindset by gathering personal details about the target. This phase, known as **reconnaissance**, is critical as it helps in building a more targeted and realistic wordlist, which significantly increases the chances of a successful brute force attack.

To collect personal and behavioral data (such as names, birth dates, hobbies, pet names, etc.), attackers can use a wide range of tools. One such tool is **Maltego**, a powerful open-source intelligence (OSINT) tool used for mapping relationships between people, domains, companies, emails, and more. While Maltego is often used legally for investigation and threat intelligence, in attacker hands it becomes a tool to **track and trace** individuals or organizations.

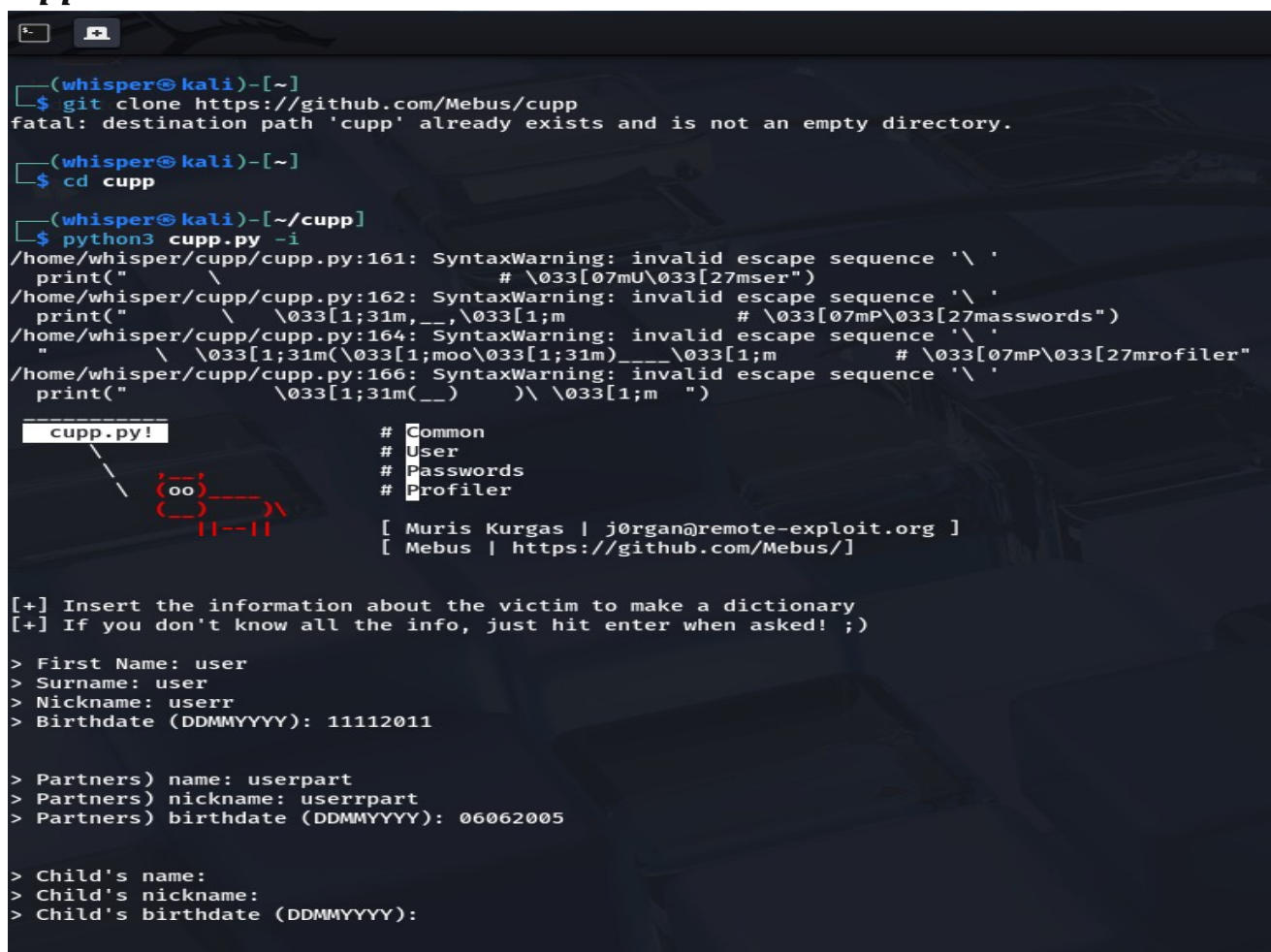
In this simulation, I did not target a real person. Instead, I skipped the actual reconnaissance phase and moved directly to the wordlist customization using fake sample information to simulate a realistic scenario.

## custom wordlist Generation :

Following the simulated reconnaissance phase , I used the tool **Cupp** (Common User Passwords Profiler) to generate a custom wordlist based on fake sample information.

Cupp works by asking for several personal details and then generates password combinations that are commonly used by individuals based on that data.

### *cupp simulation :*



```
(whisper@kali)-[~]
$ git clone https://github.com/Mebus/cupp
fatal: destination path 'cupp' already exists and is not an empty directory.

(whisper@kali)-[~]
$ cd cupp

(whisper@kali)-[~/cupp]
$ python3 cupp.py -i
/home/whisper/cupp/cupp.py:161: SyntaxWarning: invalid escape sequence '\ '
print("
# \033[07mU\033[27mser")
/home/whisper/cupp/cupp.py:162: SyntaxWarning: invalid escape sequence '\ '
print("
\033[1;31m,__,\033[1;m
# \033[07mP\033[27masswords")
/home/whisper/cupp/cupp.py:164: SyntaxWarning: invalid escape sequence '\ '
"
\033[1;31m(\033[1;moo\033[1;31m)____\033[1;m
# \033[07mP\033[27mrofiler"
/home/whisper/cupp/cupp.py:166: SyntaxWarning: invalid escape sequence '\ '
print("
\033[1;31m(__)
)\ \033[1;m ")

cupp.py!
# Common
# User
# Passwords
# Profiler
[ Muris Kurgas | j0rgan@remote-exploit.org ]
[ Mebus | https://github.com/Mebus/ ]

[+] Insert the information about the victim to make a dictionary
[+] If you don't know all the info, just hit enter when asked! ;)

> First Name: user
> Surname: user
> Nickname: userr
> Birthdate (DDMMYYYY): 11112011

> Partners) name: userpart
> Partners) nickname: userrpart
> Partners) birthdate (DDMMYYYY): 06062005

> Child's name:
> Child's nickname:
> Child's birthdate (DDMMYYYY):
```

✍ The generated wordlist was saved as `user.txt` and used in the brute force attack phase.

You can use other tools like mentalist (GUI ) or crunch (CLI )

## Step 1 : Preparing the Environment

To ensure that the brute force attack was conducted safely and ethically, I created a controlled environment using a virtual machine and a test account. This allowed me to simulate a realistic attack scenario without targeting any real users or violating any policies.


Component	Description
Attacker Machine	Kali Linux (running on VirtualBox)
Network Masking	TOR network
Target	Test Instagram account
Wordlist	target_wordlist.txt (generated via Cupp)
Script Used	Custom Python brute force script

### Kali Linux virtual machine running, connected through Tor :

```
(whisper@kali)-[~]
└─$ curl --socks5 127.0.0.1:9050 https://check.torproject.org/
<!doctype html>
<html lang="en_US">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>

  Congratulations. This browser is configured to use Tor.

</title>
<link rel="icon" type="image/x-icon" href="/torcheck/img/tor-not.png" />
<style>
  html { height: 100%; }
  body {
    height: 100%;
    text-align: center;
```

 The use of TOR helps in anonymizing the IP address and avoiding quick blocking.

Command : `sudo systemctl start tor@default`

## Step 2: Launching the Attack

After setting up the environment and generating a custom wordlist using Cupp, I launched the brute force attack using a Python script designed to automate the login attempts against the Instagram platform.

### Tools and Setup:

- **Attacking Machine:** Kali Linux VM
- **VPN:** Windscribe
- **Anonymity Layer:** TOR
- **Wordlist:** wordlist.txt
- **Script:** Custom Python script designed to simulate Instagram login attempts
- **Target:** A test account created for this experiment

### 2-1 Windscribe connected to a U.S. server :

```
(whisper@kali)-[~/SocialMediaHackingToolkit/dependencies]
$ windscribe connect us
Your account has ran out of bandwidth for this month
You can either upgrade to Pro or wait for your free data to reset
You can upgrade via our website here: https://windscribe.com/upgrade

(whisper@kali)-[~/SocialMediaHackingToolkit/dependencies]
$ windscribe logout
Logged Out, Disconnecting
Firewall Disabled
DISCONNECTED

(whisper@kali)-[~/SocialMediaHackingToolkit/dependencies]
$ windscribe login
Windscribe Username:
Windscribe Password:
Logged In

(whisper@kali)-[~/SocialMediaHackingToolkit/dependencies]
$ windscribe connect us
Connecting to US East New York Grand Central (UDP:443)
Firewall Enabled
Connected to US East New York Grand Central
Your IP changed from 197.204.140.24 to 66.232.126.72

(whisper@kali)-[~/SocialMediaHackingToolkit/dependencies]
$
```

## 2-2 Script Design and Functionality :

A key part of this project involved writing and modifying a Python script that simulates a brute force attack using a wordlist.

The script attempts login using each password and interprets Instagram's responses to determine whether the attempt was successful, blocked, or requires verification ( 2FA or checkpoint).

### Example about functions from the script :

This function randomly selects a country code from a predefined list and connects to the corresponding VPN server via the Windscribe CLI.

The purpose is to simulate attacker behavior by changing the public IP address regularly during the brute force attack, helping to bypass temporary bans or rate-limiting mechanisms applied by the server :

```
#windscribe vpn random country
codeList = ["TR", "US-C", "US", "US-W", "CA", "CA-W", "FR", "DE", "NL", "NO", "RO", "CH", "GB", "HK"]
choiceCode = random.choice(codeList)

def change_ip():
    if "Arch" in distro.linux_distribution()[0]:
        os.system("sudo systemctl start windscribe")
        os.system("\nwindscribe connect " + choiceCode)
```

Additionally, the script rotates VPN connections (via Windscribe CLI) after a number of failed attempts, and inserts randomized delays between login requests to avoid simple detection.

**→ All execution was done in a controlled environment with a test account, and the full script is not shared for ethical reasons**

## 2-3 script execution :

```
(myenv)-(whisper@kali)-[~/SocialMediaHackingToolkit]
$ chmod +x linux.sh

(myenv)-(whisper@kali)-[~/SocialMediaHackingToolkit]
$ ./linux.sh
```

Script response showing "checkpoint\_required" and "needs\_verification" :

```
[choice] wordlist.txt
[🔑] Trying: [REDACTED]
[⚠️] Error for '[REDACTED]': login error: "fail" status, message "".
[👤] Result: unknown
[?] Unknown error, continuing...
[⌚] Waiting 2.67s...

[🔑] Trying: [REDACTED]
[⚠️] Error for '[REDACTED]': login error: "fail" status, message "".
[👤] Result: unknown
[?] Unknown error, continuing...
[⌚] Waiting 2.06s...

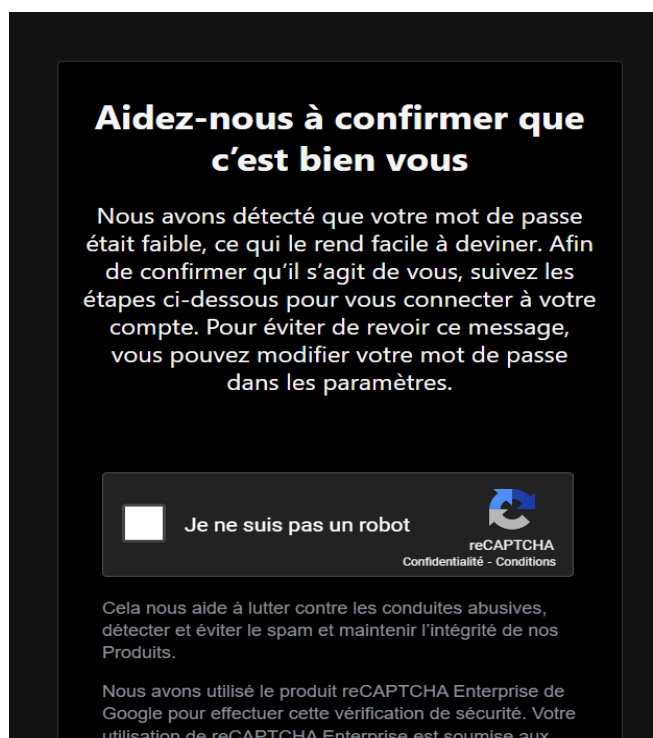
[🔑] Trying: I love you
[⚠️] Error for 'I love you': login error: "fail" status, message "".
[👤] Result: unknown
[?] Unknown error, continuing...
[⌚] Waiting 1.54s...

[🔑] Trying: [REDACTED]
[⚠️] Error for '[REDACTED]': login error: "fail" status, message "".
[👤] Result: unknown
[?] Unknown error, continuing...
[⌚] Waiting 1.48s...

[🔑] Trying: !!!!12Jps
[⚠️] Error for '!!!!12Jps': login: checkpoint required. point your browser
rwvrbkg/ffc_qabhnvic6zdvctlmiv7jsmkpogfimbqtmkobffurim8avrxv1n7zzndfcsc
[👤] Password is correct but needs verification (2FA or checkpoint).
[👤] Result: needs_verification

[✅] Password is correct: !!!!12Jps
[👤] But requires 2FA or checkpoint to continue.
```





After the brute force script successfully identified the correct password, I attempted to log in manually to verify the result. At that point, Instagram triggered a **CAPTCHA** challenge to confirm that the login attempt was not automated.

This protection layer is part of Instagram's real-time detection system, which assesses login behavior based on factors like timing, location, and device fingerprint.

Even though the **credentials were correct**, Instagram blocked access to the account without solving the **CAPTCHA**.

→ After identifying the correct password during the simulation, I attempted to log in from a different device. Instagram immediately flagged the attempt as suspicious and blocked access, requesting a password reset.

**This response confirms that:**

- Instagram uses **device and behavior-based detection**.
- The system **recognizes unusual login patterns**, even with correct credentials.
- **Multi-layered security** plays a crucial role in preventing account compromise.

## Suspicious login attempt



## We blocked a suspicious login attempt

Web | 6/17/25 2:05 PM  
New York, New York

Somebody tried to log in to your account with your password. We were able to stop them before they got access, but you should change your password to keep your account secure.



# *Defense Against Brute Force Attacks*

## Instagram's Real-World Defensive Reactions :

During the brute force simulation, Instagram responded to my login attempts with multiple layered defense mechanisms. These reactions varied depending on the behavior of the script, the IP used, and the success of each password attempt. Below is a breakdown of the main protective responses I encountered :

### **1. CAPTCHA After Manual Login**

After the script successfully identified the correct password, I tried logging in manually from a browser. Instagram immediately presented a CAPTCHA challenge, preventing further access until I proved I was human. This demonstrates a smart anti-automation layer that activates only when suspicious behavior is detected.

---

### **2. Checkpoint and 2FA Protection**

When the correct password was detected by the script, the response did not allow full login. Instead, Instagram redirected to a checkpoint page requiring additional verification, such as email or phone confirmation. This step ensured that even with the correct credentials, unauthorized users could not bypass two-factor protection.

---

### **3. Device and IP Behavior Analysis**

When I attempted to log in from a different device or browser, Instagram flagged the session as suspicious. It sent an alert to the account's email and blocked the login, asking for a password reset. This clearly indicates that

Instagram uses behavioral analytics — comparing device fingerprints and IP reputation — to detect unusual access patterns.

---

#### 4. Rate Limiting and Temporary Blocking

After several login attempts, even with different passwords and IPs, Instagram started blocking access or returning inconsistent error responses. This behavior is consistent with rate-limiting policies, which are designed to slow down or temporarily suspend repeated failed attempts.

---

#### 5. Timing Pattern Detection

One of the most subtle yet powerful techniques used by Instagram is **timing analysis**. During the simulation, I noticed that login attempts with fixed delays became less effective, and server behavior changed after several periodic attempts.

This suggests that Instagram monitors the **intervals between login requests**. Automated tools that send requests at consistent time intervals are easier to detect.

To avoid this, I implemented **randomized delays** (between 1–3 seconds) between login attempts in the script, simulating a more human-like interaction pattern.

This adjustment helped reduce detection and improved the realism of the attack — further confirming that **timing behavior** is part of Instagram's defense model.

---

#### 6. Redirection to Invalid or Broken Pages

During several test runs, especially when the script behavior became more repetitive or obvious, Instagram started redirecting the session to **non-functional or undefined pages**. These pages either had no content or returned unexpected HTML structures.

This appears to be a **deliberate anti-automation tactic**. By serving broken or misleading content to suspicious sessions, Instagram attempts to disrupt scripted flows that rely on specific elements (like forms, buttons, or URLs).


Such redirection is difficult to detect programmatically, especially if the script expects a normal login page or success page. This forced me to update the script's logic to rely not only on the page URL but also on HTML content patterns and error messages.

→ **This experience reinforced how platforms employ non-obvious defenses — not just by blocking, but by confusing and trapping automation attempts.**

---

## 7. UI Element Interference

Another interesting defense mechanism encountered during the simulation was **UI interference**. When attempting to automate the login process using Playwright, the script failed to click the “Login” button, even though the element was fully visible and stable.



```
[▲] Error: Page.click: Timeout 30000ms exceeded.
```

The error log indicated that:

“A child `<span>` or a layered `<div>` was intercepting pointer events, preventing the click action.”

This suggests that Instagram deliberately places invisible or overlapping elements on key buttons to **block automation tools**. The script had to be modified using `force=True` in the click method, and even then, success was inconsistent.

→ **This kind of front-end defense targets behavior-based automation and is especially effective against naïve scripts that rely on DOM stability alone.**

## ***Final Thoughts :***

This simulation was not just a technical exercise, but a deep dive into how real-world systems behave under targeted pressure. It showed that brute force attacks, while conceptually simple, face numerous challenges when confronted with modern, layered defenses.

From CAPTCHA to behavioral fingerprinting and UI-level traps, platforms like Instagram demonstrate how security is no longer about passwords — but about **context, behavior, and adaptation**.

Writing this report helped me understand not only how attackers think, but more importantly, how defenders build intelligent countermeasures.