Skills For Hire Atlantic

Cybersecurity - Assignment 1 (WO5477 Rudy Im)

[Google Colab Link]

https://colab.research.google.com/drive/1CuoIn6yx2bWOILDYaD7nCw1unQRvFDNd

[Question 1]

What would happen if we always used the same encryption key without an Initialization Vector (IV) when encrypting messages?

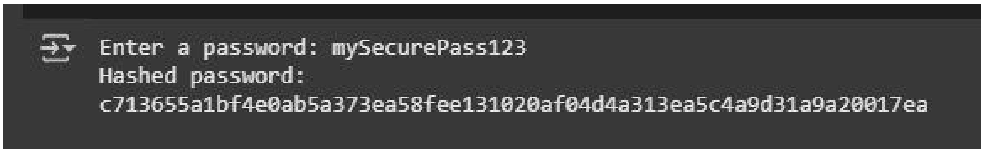
Even if you use the previous block to keep changing the key, the first block which doesn't have the previous block will be vulnerable. Also, without random vectors, some common data like long blank in text, or mono-colored background pixels in image, can easily expose the pattern.

What kind of attack could happen, and how does an Initialization Vector (IV) solve this problem?

If there is no initialization vector, the attacker can notice the pattern more easily, especially when the data is monolithic. Also, even if they could not figure out the encryption process, they can simply use pre-encrypted keywords to manipulate documents, if it is always encrypted to the same cipher.

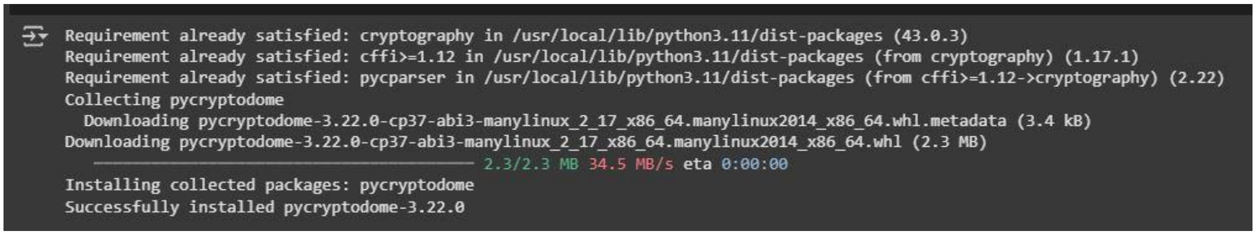
Google Colab Code + Output Screenshot

```
import hashlib

password_file = 'hashed_passwords.txt'

def set_password():
  password = input('Enter a password: ')
  byte_password = password.encode()
  hashed_password = hashlib.sha256(byte_password)
  print('Hashed password:')
  print(hashed_password.hexdigest())
  with open(password_file, 'a') as file:
    file.write(hashed_password.hexdigest() + '\n')

set_password()
```



```
Enter a password: mySecurePass123
Hashed password:
c713655a1bf4e0ab5a373ea58fee131020af04d4a313ea5c4a9d31a9a20017ea
```

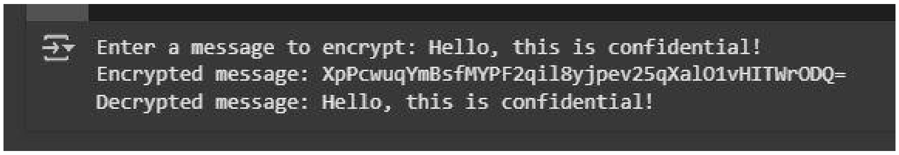
```
!pip install cryptography
!pip install pycryptodome
```



```
Requirement already satisfied: cryptography in /usr/local/lib/python3.11/dist-packages (43.0.3)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.11/dist-packages (from cryptography) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.12->cryptography) (2.22)
Collecting pycryptodome
  Downloading pycryptodome-3.22.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (3.4 kB)
Downloading pycryptodome-3.22.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 34.5 MB/s eta 0:00:00
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.22.0
```

```
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

import base64

password_file = 'hashed_passwords.txt'
message_file = 'encrypted_message.bin'
key_len = 16

def get_salt():
  return get_random_bytes(key_len)

def get_password():
  with open(password_file, 'r') as file:
    hashed_passwords = file.readlines()
  return hashed_passwords[-1].strip()

def get_key(password, salt):
  kdf = PBKDF2HMAC(
      algorithm=hashes.SHA256(),
      length=key_len,
      salt=salt,
      iterations=100000,
      backend=default_backend()
  )
  return kdf.derive(password.encode())

def pad(text):
    pad_len = key_len - (len(text) % key_len)
    return text + chr(pad_len) * pad_len

def unpad(text):
    pad_len = ord(text[-1])
    return text[:-pad_len]

def aes_cbc_encrypt(text, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return base64.b64encode(cipher.encrypt(pad(text).encode()))

def aes_cbc_decrypt(encrypted, key, iv):
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(base64.b64decode(encrypted)).decode())
```

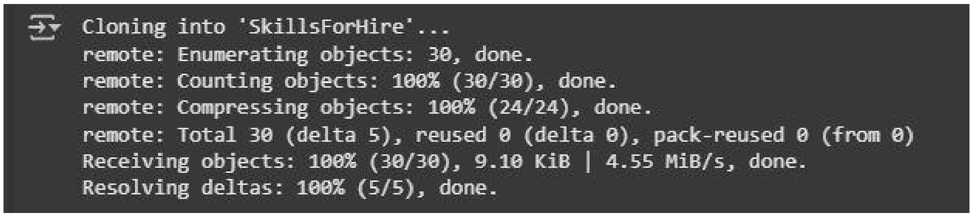
```
def encrypt_message():
  message = input('Enter a message to encrypt: ')
  salt = get_salt()
  key = get_key(get_password(), salt)
  encrypted_message = aes_cbc_encrypt(message, key, salt)
  print('Encrypted message:', encrypted_message.decode())
  with open(message_file, 'wb') as file:
    file.write(salt)
    file.write(encrypted_message)

def decrypt_message():
  with open(message_file, 'rb') as file:
    salt = file.read(key_len)
    encrypted_message = file.read()
  key = get_key(get_password(), salt)
  print('Decrypted message:', aes_cbc_decrypt(encrypted_message, key, salt))

encrypt_message()
decrypt_message()
```



```
Enter a message to encrypt: Hello, this is confidential!
Encrypted message: XpPcwuqYmBsfMYPF2qil8yjpev25qXalO1vHITWrODQ=
Decrypted message: Hello, this is confidential!
```

[File Import]

```
!git clone https://github.com/rudy-im/SkillsForHire.git
```



```
Cloning into 'SkillsForHire'...
remote: Enumerating objects: 30, done.
remote: Counting objects: 100% (30/30), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 30 (delta 5), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (30/30), 9.10 KiB | 4.55 MiB/s, done.
Resolving deltas: 100% (5/5), done.
```

[Question 2]

What is a brute force attack?

The method to try many different compositions to find the password that works.

Why do multiple failed login attempts within a short time indicate a potential brute force attack?

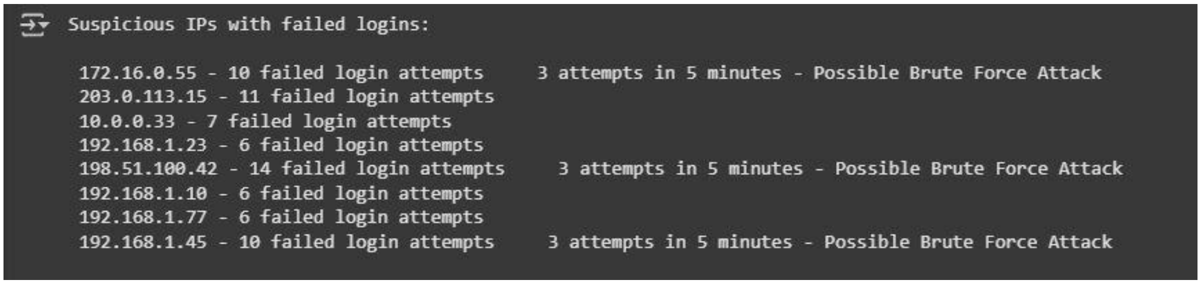
It may mean that the person doesn't know the actual password, and is just trying possible passwords that might work.

What are real-world countermeasures to prevent brute force attacks?

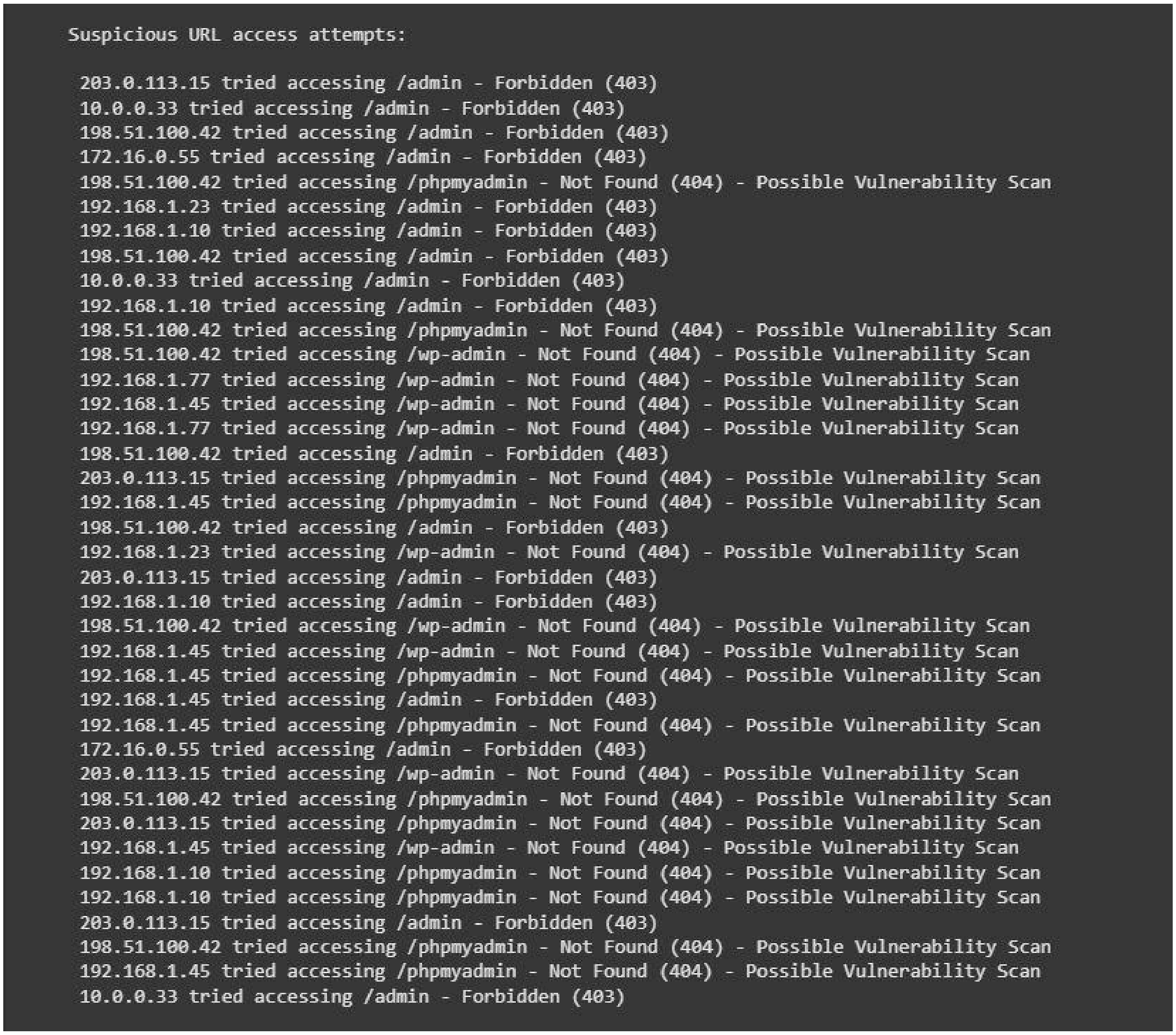
You can limit maximum failed attempts, and lock the account after that.

Google Colab Code + Output Screenshot

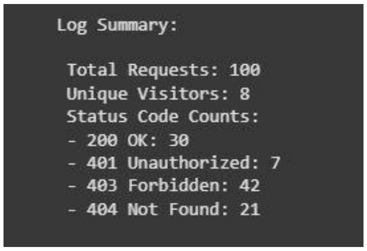
```
import re, datetime
from collections import Counter

log_file = '/content/SkillsForHire/CyberSecurity Assignment 1-2/access.log'

status_categories = {
    '200': 'OK',
    '301': 'Moved Permanently',
    '302': 'Found',
    '400': 'Bad Request',
    '401': 'Unauthorized',
    '403': 'Forbidden',
    '404': 'Not Found'
}

sensitive_pages = ['/admin', '/wp-admin', '/phpmyadmin']

suspicious_messages = {
    '404': 'Possible Vulnerability Scan',
    '200': 'Possible Unauthorized Access'
}

def get_datetime_instance(datetime_str):
  return datetime.datetime.strptime(datetime_str, '%d/%b/%Y:%H:%M:%S %z')

def log_to_dic(log):
  regex = re.compile(r'(?P<ip>\d+\.\d+\.\d+\.\d+) - -
\[(?P<datetime>\d+/.+/\d+:\d\d:\d\d:\d\d \+\d+)\] "(?P<method>[A-Z]+)
(?P<url>[/\-\w\d\.]+) (?P<protocol>[/\-\w\d\.]+)" (?P<status>\d+) "-"
"(?P<browser>.+)"')
  res = regex.search(log)
  if not res: return
  dic = res.groupdict()
  dic['datetime'] = get_datetime_instance(dic['datetime'])
  return dic

def get_attempts_while_watching(datetimes, watching_minutes):
  watching_seconds = watching_minutes * 60
  if len(datetimes) < 1: return 0
  timedeltas = []
  for i in range(len(datetimes)-1):
    timedeltas.append((datetimes[i+1]-datetimes[i]).total_seconds())
  max_attempts = 1
  for i in range(len(timedeltas)):
    watching = 0
    for j in range(i, len(timedeltas)):
      if watching + timedeltas[j] > watching_seconds: break
      watching += timedeltas[j]
    if j-i+1 > max_attempts: max_attempts = j-i+1
  return max_attempts

def analyze_failed(failed):
  watching_minutes = 5
  suspicious_count = 3
  print('Suspicious IPs with failed logins:')
  for ip, datetimes in failed.items():
    print('\n', ip, '-', len(datetimes), 'failed login attempts', end='')
    attempts = get_attempts_while_watching(datetimes, watching_minutes)
    if attempts < suspicious_count: continue
    print('    ', attempts, 'attempts in', watching_minutes, 'minutes - Possible Brute
Force Attack', end='')
  print('\n\n')

def analyze_suspicious(suspicious):
  print('Suspicious URL access attempts:')
  for log_dic in suspicious:
    print('\n', log_dic['ip'], 'tried accessing', log_dic['url'], end=' - ')
    if status_categories.get(log_dic['status']):
      print(status_categories.get(log_dic['status']), end=' ')
    print(f'({log_dic["status"]})', end='')
    if suspicious_messages.get(log_dic['status']):
      print(' -', suspicious_messages.get(log_dic['status']), end='')
  print('\n\n')

def analyze_log():
  failed = {}
  suspicious = []
  total_requests = 0
  visitors_counter = Counter()
  status_counter = Counter()
  with open(log_file, 'r') as file:
    for log in file:
      log_dic = log_to_dic(log)
      if not log_dic: continue
      if log_dic['status'] != '200':
        if not failed.get(log_dic['ip']): failed[log_dic['ip']] = []
        failed[log_dic['ip']].append(log_dic['datetime'])
      if list(filter(lambda sensitive: sensitive in log_dic['url'], sensitive_pages)):
        suspicious.append(log_dic)
      total_requests += 1
      visitors_counter.update([log_dic['ip']])
      status_counter.update([log_dic['status']])
  analyze_failed(failed)
  analyze_suspicious(suspicious)
  print("Log Summary:\n")
  print(' Total Requests:', total_requests)
  print(' Unique Visitors:', len(visitors_counter))
  print(' Status Code Counts:')
  for status, count in status_counter.items():
    print(' -', status, end='')
    if status_categories.get(status): print(' ' + status_categories.get(status),
end='')
    print(':', count)
  print('\n\n')

analyze_log()
```



```
Suspicious IPs with failed logins:

 172.16.0.55 - 10 failed login attempts     3 attempts in 5 minutes - Possible Brute Force Attack
 203.0.113.15 - 11 failed login attempts
 10.0.0.33 - 7 failed login attempts
 192.168.1.23 - 6 failed login attempts
 198.51.100.42 - 14 failed login attempts     3 attempts in 5 minutes - Possible Brute Force Attack
 192.168.1.10 - 6 failed login attempts
 192.168.1.77 - 6 failed login attempts
 192.168.1.45 - 10 failed login attempts     3 attempts in 5 minutes - Possible Brute Force Attack
```



```
Suspicious URL access attempts:

 203.0.113.15 tried accessing /admin - Forbidden (403)
 10.0.0.33 tried accessing /admin - Forbidden (403)
 198.51.100.42 tried accessing /admin - Forbidden (403)
 172.16.0.55 tried accessing /admin - Forbidden (403)
 198.51.100.42 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.23 tried accessing /admin - Forbidden (403)
 192.168.1.10 tried accessing /admin - Forbidden (403)
 198.51.100.42 tried accessing /admin - Forbidden (403)
 10.0.0.33 tried accessing /admin - Forbidden (403)
 192.168.1.10 tried accessing /admin - Forbidden (403)
 198.51.100.42 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 198.51.100.42 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.77 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.77 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 198.51.100.42 tried accessing /admin - Forbidden (403)
 203.0.113.15 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 198.51.100.42 tried accessing /admin - Forbidden (403)
 192.168.1.23 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 203.0.113.15 tried accessing /admin - Forbidden (403)
 192.168.1.10 tried accessing /admin - Forbidden (403)
 198.51.100.42 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /admin - Forbidden (403)
 192.168.1.45 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 172.16.0.55 tried accessing /admin - Forbidden (403)
 203.0.113.15 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 198.51.100.42 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 203.0.113.15 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /wp-admin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.10 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.10 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 203.0.113.15 tried accessing /admin - Forbidden (403)
 198.51.100.42 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 192.168.1.45 tried accessing /phpmyadmin - Not Found (404) - Possible Vulnerability Scan
 10.0.0.33 tried accessing /admin - Forbidden (403)
```



```
Log Summary:

 Total Requests: 100
 Unique Visitors: 8
 Status Code Counts:
 - 200 OK: 30
 - 401 Unauthorized: 7
 - 403 Forbidden: 42
 - 404 Not Found: 21
```

[Question 3]

What are the limitations of YARA and signature-based malware detection?

If the attacker modifies the file or signature to negate the known rule, it will not be detected.

How do modern attackers bypass signature-based detection?

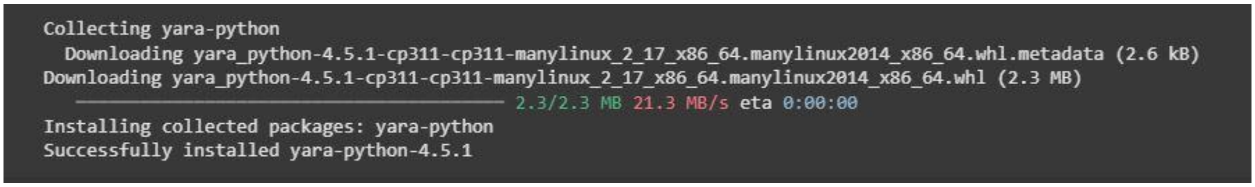
They can use salt-based encryption to encrypt the file differently every time. Also, they can modify the code by adding meaningless operations, reordering functions, or change it to equivalent but different code. Or they can even make it fileless and inject malicious code to legitimate software. Nowadays, AI can help it even more sophisticatedly.

What other cybersecurity techniques can be used to detect malware more effectively?

Instead of checking the file, you can check the behavior or access log. Looking for suspicious traits by heuristic analysis can be another way. Or you can run the program in an isolated environment called a sandbox, which is like a virtual system, to see how it operates. Just like attackers do, security agents can use AI to detect suspicious behavior, or share information globally to learn attackers better.

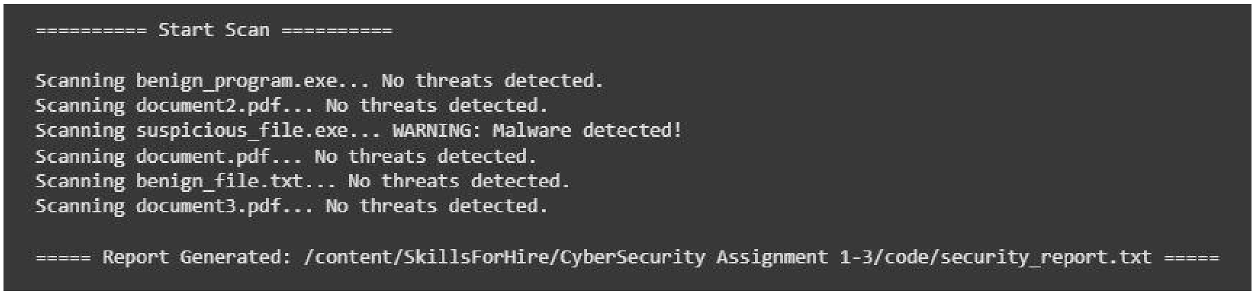
Google Colab Code + Output Screenshot

```
!apt-get update
!apt-get install -y libyara-dev
!pip install yara-python

!cp /usr/lib/x86_64-linux-gnu/libyara.so /usr/lib/libyara.so
```



```
Collecting yara-python
  Downloading yara_python-4.5.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (2.6 kB)
Downloading yara_python-4.5.1-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (2.3 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.3/2.3 MB 21.3 MB/s eta 0:00:00
Installing collected packages: yara-python
Successfully installed yara-python-4.5.1
```

```
import datetime, os
import yara

project_dir = '/content/SkillsForHire/CyberSecurity Assignment 1-3'
yara_rule = yara.compile(filepath = project_dir + '/code/malware_rule.yar')
report_file = project_dir + '/code/security_report.txt'

def yara_scan(file):
  print('Scanning ' + file.split('/')[-1] + '...', end=' ')
  if yara_rule.match(filepath = file):
    print('WARNING: Malware detected!')
    return True
  else:
    print('No threats detected.')
    return False

def gen_report(dic):
  with open(report_file, 'a') as report:
    now = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    report.write('\n\nMalware Detection Report - [' + now + ']\n\n')
    report.write('Files Scanned:\n')

    count = 1
    for file, result in dic.items():
      report.write(str(count) + '. ' + file + ' → ')
      count += 1
      if result:
        report.write('WARNING: Malware detected!\n')
      else:
        report.write('No threats detected.\n')

def scan_dir(folder):
  if not os.path.isdir(folder):
    print('Not a valid path!!')
    return

  print('\n\n========== Start Scan ==========\n')
  result = {}

  for item in os.listdir(folder):
    path = os.path.join(folder, item)
    if os.path.isfile(path):
      result[item] = yara_scan(path)

  gen_report(result)
  print('\n===== Report Generated: ' + report_file + ' =====\n')


scan_dir(project_dir)
```



```
========== Start Scan ==========

Scanning benign_program.exe... No threats detected.
Scanning document2.pdf... No threats detected.
Scanning suspicious_file.exe... WARNING: Malware detected!
Scanning document.pdf... No threats detected.
Scanning benign_file.txt... No threats detected.
Scanning document3.pdf... No threats detected.

===== Report Generated: /content/SkillsForHire/CyberSecurity Assignment 1-3/code/security_report.txt =====
```