Building a Custom Honeypot

Objectives:

- Detect Threats: Identify and monitor unauthorized access and malicious behavior.
- Analyze Attacks: Study attacker methods to improve understanding of threat patterns.
- Strengthen Defenses: Use insights to enhance system security and patch vulnerabilities.
- Divert Attackers: Act as a decoy to protect critical systems from direct attacks.
- Support Incident Response: Provide detailed logs to facilitate swift threat mitigation.

Expected Outcomes:

- Increased visibility of cyber threats and malicious activity.
- Enhanced defenses through updated security tools and patched vulnerabilities.
- Valuable intelligence on attacker behavior and trends.
- Reduced risk to critical assets via decoy mechanisms.
- Faster and more efficient response to cyber incidents.
- Improved training for security teams in handling threats.

Cowrie Honey-Pot

Using cowrie Honey Pot:

Cowrie is a medium to high interaction SSH and Telnet honeypot designed to log brute force attacks and the shell interaction performed by the attacker. In medium interaction mode (shell) it emulates a UNIX system in Python, in high interaction mode (proxy) it functions as an SSH and telnet proxy to observe attacker behavior to another system.

How to use cowrie:

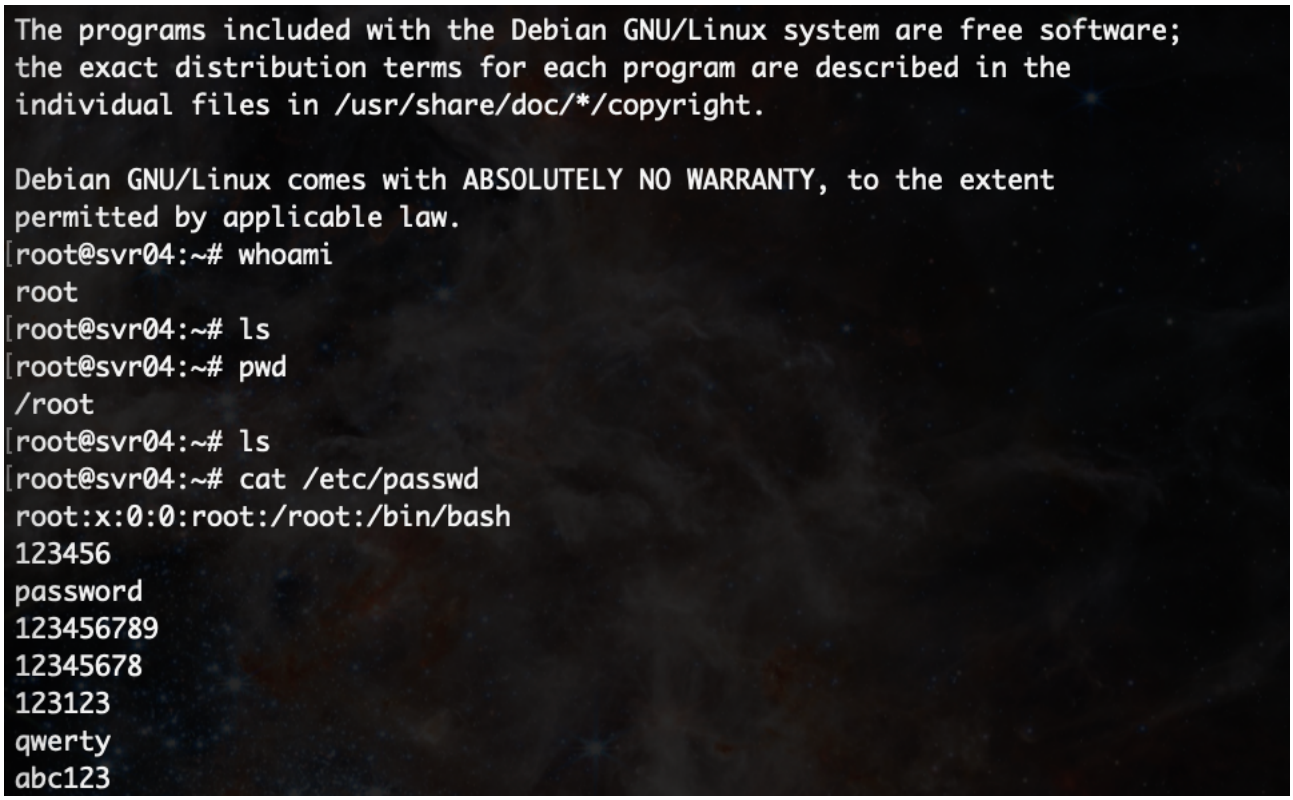
to start cowrie we use the command “bin/cowrie start” and to close cowrie we use the command “bin/cowrie stop” NOTE THAT YOU HAVE TO BE IN THE INSALL DIRECTORY

Configrated programs :

1- SQL Injection detection
2- XSS and CSRF attack detection
3- Automated Script Detection (SQL, XSS, CSRF)
4- Brute Force Attack Detection
5- Directory Traversal Attack Detection
6- Remote Command Execution Detection
7- HTTP Request Flood Detection (DoS Attack)
8 - Malicious User-Agent Detection
9- detect IPs

Stimualte the attack same PC Different Terminals

1-Navigate to the install directory
2- bin/cowrie start (starting the honey pot)
3- tail -f var/log/cowrie/cowrie.log (viewing the logs) we will view each input he types
4- from another device or in my case another terminal we will use this command “ssh root@localhost -p 2222” to establish connection with the server.
5- now we search for the password file using the command “cat /etc/passwd”



Attacking using another Device

1-We ran the command “docker run -p 2222:2222 cowrie/cowrie:latest” to host the server
2- from another computer we scanned the network using nmap -sV IP to locate the SSH

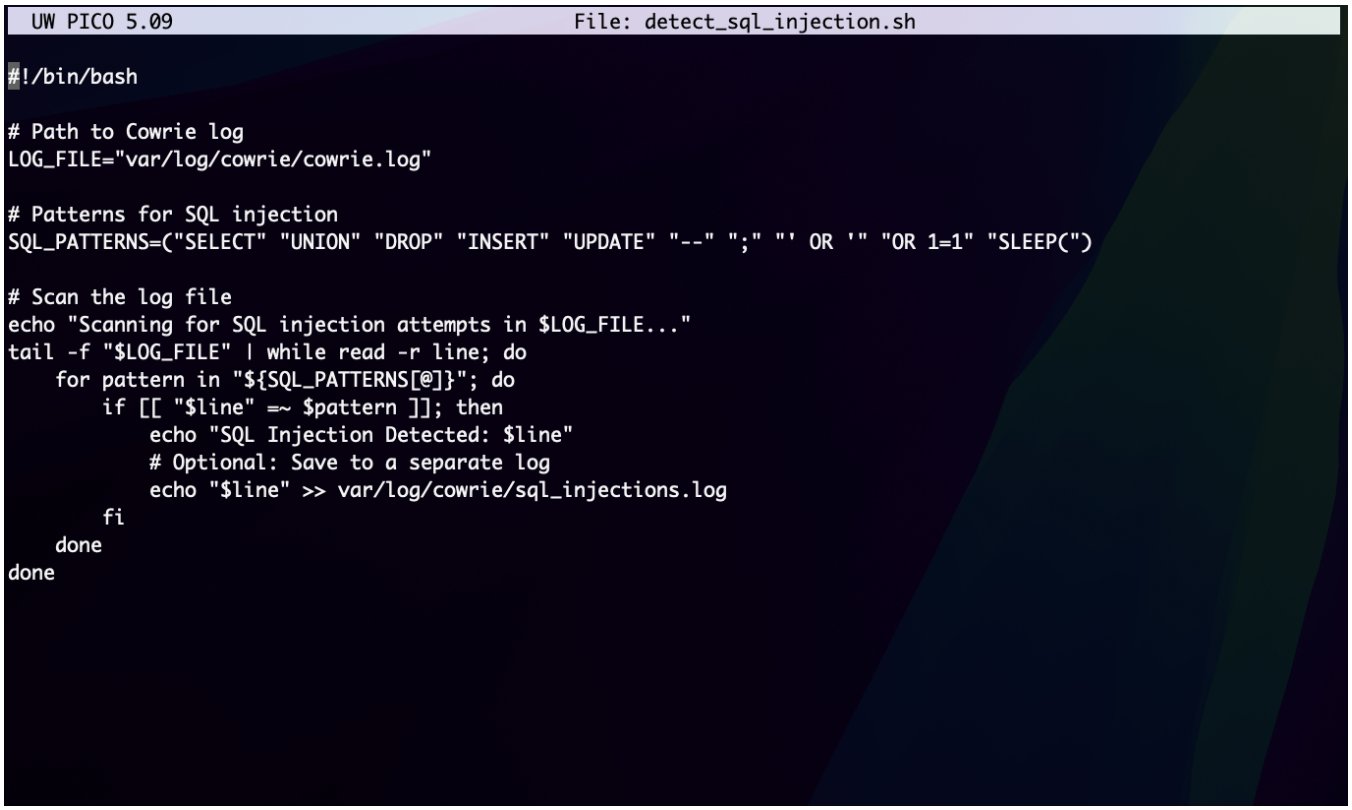
```
PS C:\Users\MoHamed\Desktop> nmap -sV 192.168.87.*/24nmap -sV 192.168.87.*/2nmap
/1nmap -sV 192.168.87.*/nmap -sV 192
Starting Nmap 7.95 ( https://nmap.org ) at 2024-11-23 16:37 Egypt Standard Time
Nmap scan report for 192.168.87.18
Host is up (0.043s latency).
Not shown: 994 closed tcp ports (reset)
PORT     STATE SERVICE       VERSION
88/tcp   open  kerberos-sec  Heimdal Kerberos (server time: 2024-11-23 14:37:58Z)
445/tcp  open  microsoft-ds?
2222/tcp open  ssh           OpenSSH 6.0p1 Debian 4+deb7u2 (protocol 2.0)
5000/tcp open  rtsp
5900/tcp open  vnc           Apple remote desktop vnc
7000/tcp open  rtsp
```

3-we ran the command ssh root@DEVICE-IP -p PORT to connect to the server
4-now that we are connected to the server (Honey Pot) we can freely search the system in the other side we will use the command **tail -f cowrie.log** to view the logs that the attacker is typing

```
2024-11-23T14:31:41.234869Z [twisted.conch.ssh.session#info] Getting shell
2024-11-23T14:31:52.198492Z [HoneyPotSSHTransport,7,192.168.87.148] CMD: cd /etc
2024-11-23T14:31:52.212383Z [HoneyPotSSHTransport,7,192.168.87.148] Command found: cd /etc
2024-11-23T14:31:53.571803Z [HoneyPotSSHTransport,7,192.168.87.148] CMD: ls
2024-11-23T14:31:53.584897Z [HoneyPotSSHTransport,7,192.168.87.148] Command found: ls
2024-11-23T14:32:01.447531Z [HoneyPotSSHTransport,7,192.168.87.148] CMD: cat profile
2024-11-23T14:32:01.460047Z [HoneyPotSSHTransport,7,192.168.87.148] Command found: cat profile
2024-11-23T14:32:01.535093Z [HoneyPotSSHTransport,7,192.168.87.148] CMD:
2024-11-23T14:32:15.313355Z [HoneyPotSSHTransport,7,192.168.87.148] CMD: cat issue
2024-11-23T14:32:15.326863Z [HoneyPotSSHTransport,7,192.168.87.148] Command found: cat issue
2024-11-23T14:34:41.187411Z [-] Timeout reached in HoneyPotSSHTransport
2024-11-23T14:34:41.197031Z [HoneyPotSSHTransport,7,192.168.87.148] Closing TTY Log: var/lib/cowr
ie/tty/ade9cf1143de7664fe004a2a0d4ce9f10c7ec6d860c3c53bcc5f797ed1e8e645 after 179 seconds
2024-11-23T14:34:41.211973Z [HoneyPotSSHTransport,7,192.168.87.148] avatar root logging out
2024-11-23T14:34:41.212702Z [cowrie.ssh.transport.HoneyPotSSHTransport#info] connection lost
2024-11-23T14:34:41.213365Z [HoneyPotSSHTransport,7,192.168.87.148] Connection lost after 183 sec
onds
2024-11-23T14:37:52.602770Z [cowrie.ssh.factory.CowrieSSHFactory] New connection: 192.168.87.148:
30797 (192.168.87.18:2222) [session: e8ebac646516]
2024-11-23T14:37:52.639029Z [cowrie.ssh.transport.HoneyPotSSHTransport#info] connection lost
2024-11-23T14:37:52.640235Z [HoneyPotSSHTransport,8,192.168.87.148] Connection lost after 0 secon
ds
```

Identify vulnerabilities such as SQL injection

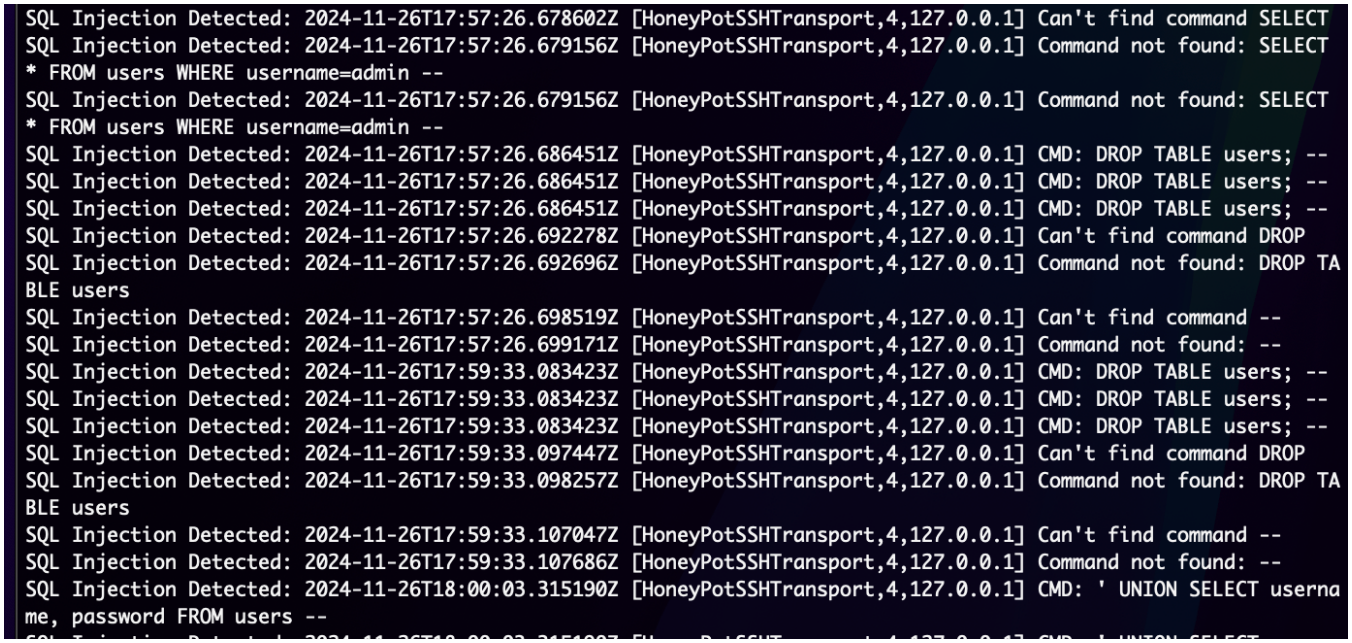
We will create a .sh file to put our program that will detect SQL injections, first will will navigate to the main Cowrie directory and create a new file “nano detect_sql_injection.sh” we will insert the following code into the file.



Now we will have to make script executable using the command “chmod +x detect_sql_injection.sh”

Now we can run our script or program using the command “./detect_sql_injection.sh”

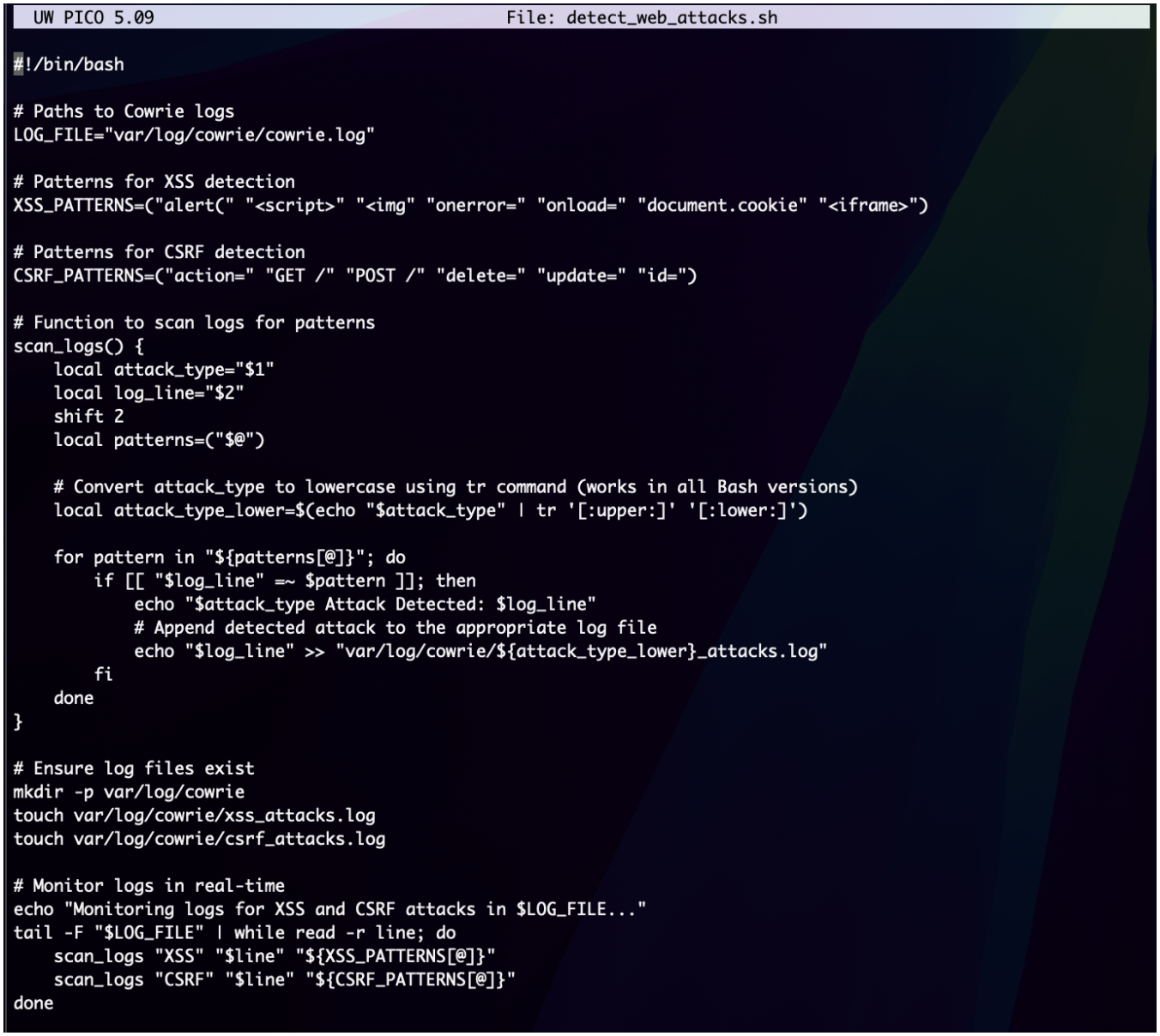
We can test the program by injecting some SQL querrys “' OR '1'='1 --
SELECT * FROM users WHERE username='admin' --
DROP TABLE users; --”



We can see that the system works well by logging each SQL injection Querry it recives.

Identify vulnerabilities such as XSS and CSRF

We will create a .sh file to put our program that will detect XSS and CSRF attacks, first will will navigate to the main Cowrie directory and create a new file “nano detect_web_attacks.sh” we will insert the following code into the file.



```
  UW PICO 5.09                                File: detect_web_attacks.sh

#!/bin/bash

# Paths to Cowrie logs
LOG_FILE="var/log/cowrie/cowrie.log"

# Patterns for XSS detection
XSS_PATTERNS=("alert(" "<script>" "<img" "onerror=" "onload=" "document.cookie" "<iframe>")

# Patterns for CSRF detection
CSRF_PATTERNS=("action=" "GET /" "POST /" "delete=" "update=" "id=")

# Function to scan logs for patterns
scan_logs() {
    local attack_type="$1"
    local log_line="$2"
    shift 2
    local patterns=("$@")

    # Convert attack_type to lowercase using tr command (works in all Bash versions)
    local attack_type_lower=$(echo "$attack_type" | tr '[:upper:]' '[:lower:]')

    for pattern in "${patterns[@]}"; do
        if [[ "$log_line" =~ $pattern ]]; then
            echo "$attack_type Attack Detected: $log_line"
            # Append detected attack to the appropriate log file
            echo "$log_line" >> "var/log/cowrie/${attack_type_lower}_attacks.log"
        fi
    done
}

# Ensure log files exist
mkdir -p var/log/cowrie
touch var/log/cowrie/xss_attacks.log
touch var/log/cowrie/csrf_attacks.log

# Monitor logs in real-time
echo "Monitoring logs for XSS and CSRF attacks in $LOG_FILE..."
tail -F "$LOG_FILE" | while read -r line; do
    scan_logs "XSS" "$line" "${XSS_PATTERNS[@]}"
    scan_logs "CSRF" "$line" "${CSRF_PATTERNS[@]}"
done
```

Now we will have to make script executable using the command “chmod +x detect_web_attacks.sh”

We can run our script or program using the command ./detect_web_attacks.sh”.

We can test our program by using the following commands,

XSS Payloads:

"<script>alert('XSS')</script>"
"<img src='x' onerror='alert(1)'>"

CSRF Payloads:

"GET /delete?id=123 HTTP/1.1"
"POST /update?id=456 HTTP/1.1"

```
[bash-3.2$ ./detect_web_attacks.sh
Monitoring logs for XSS and CSRF attacks in var/log/cowrie/cowrie.log...
XSS Attack Detected: 2024-11-26T18:24:12.094198Z [HoneyPotSSHTransport,7,127.0.0.1] CMD: "<script>alert('XSS')</s
cript>"
XSS Attack Detected: 2024-11-26T18:24:12.104543Z [HoneyPotSSHTransport,7,127.0.0.1] Can't find command <script>al
ert('XSS')</script>
XSS Attack Detected: 2024-11-26T18:24:12.105185Z [HoneyPotSSHTransport,7,127.0.0.1] Command not found: <script>al
ert('XSS')</script>
XSS Attack Detected: 2024-11-26T18:24:12.115473Z [HoneyPotSSHTransport,7,127.0.0.1] CMD: "<img src='x' onerror='a
lert(1)'>"
XSS Attack Detected: 2024-11-26T18:24:12.115473Z [HoneyPotSSHTransport,7,127.0.0.1] CMD: "<img src='x' onerror='a
lert(1)'>"
XSS Attack Detected: 2024-11-26T18:26:14.927922Z [HoneyPotSSHTransport,7,127.0.0.1] CMD: "<script>alert('XSS')</s
cript>"
XSS Attack Detected: 2024-11-26T18:26:14.941435Z [HoneyPotSSHTransport,7,127.0.0.1] Can't find command <script>al
ert('XSS')</script>
XSS Attack Detected: 2024-11-26T18:26:14.942270Z [HoneyPotSSHTransport,7,127.0.0.1] Command not found: <script>al
ert('XSS')</script>
XSS Attack Detected: 2024-11-26T18:26:16.883744Z [HoneyPotSSHTransport,7,127.0.0.1] CMD: "<script>alert('XSS')</s
cript>"
XSS Attack Detected: 2024-11-26T18:26:16.896431Z [HoneyPotSSHTransport,7,127.0.0.1] Can't find command <script>al
ert('XSS')</script>
XSS Attack Detected: 2024-11-26T18:26:16.897474Z [HoneyPotSSHTransport,7,127.0.0.1] Command not found: <script>al
ert('XSS')</script>
```

The program is working as expected and Logging the attcaks.

LOGS:
cat var/log/cowrie/xss_attacks.log
cat var/log/cowrie/csrf_attacks.log

Automated Script Detection

Bonus Feature

Why not run only one program and the program detect all the attacks automaticly (SQL, XSS, CSRF), we have made one program that include all the previous tecniques to identift the attack and record them.

We will create a .sh file to put our program that will detect SQL, XSS and CSRF attacks, first will will navigate to the main Cowrie directory and create a new file “nano automate_detection.sh” we will insert the following code into the file.

Now we will have to make script executable using the command “chmod +x automate_detection.sh

We can test our program by using the following commands,

XSS Payloads:

"<script>alert('XSS')</script>"
"<img src='x' onerror='alert(1)'>"

CSRF Payloads:

"GET /delete?id=123 HTTP/1.1"
"POST /update?id=456 HTTP/1.1"

SQL INJECTIONS:

' OR '1'='1 --
' UNION SELECT username, password FROM users --
' OR SLEEP(5) --

ssh root@localhost -p 2222
' OR '1'='1 --
SELECT * FROM users WHERE username='admin' --
DROP TABLE users; --

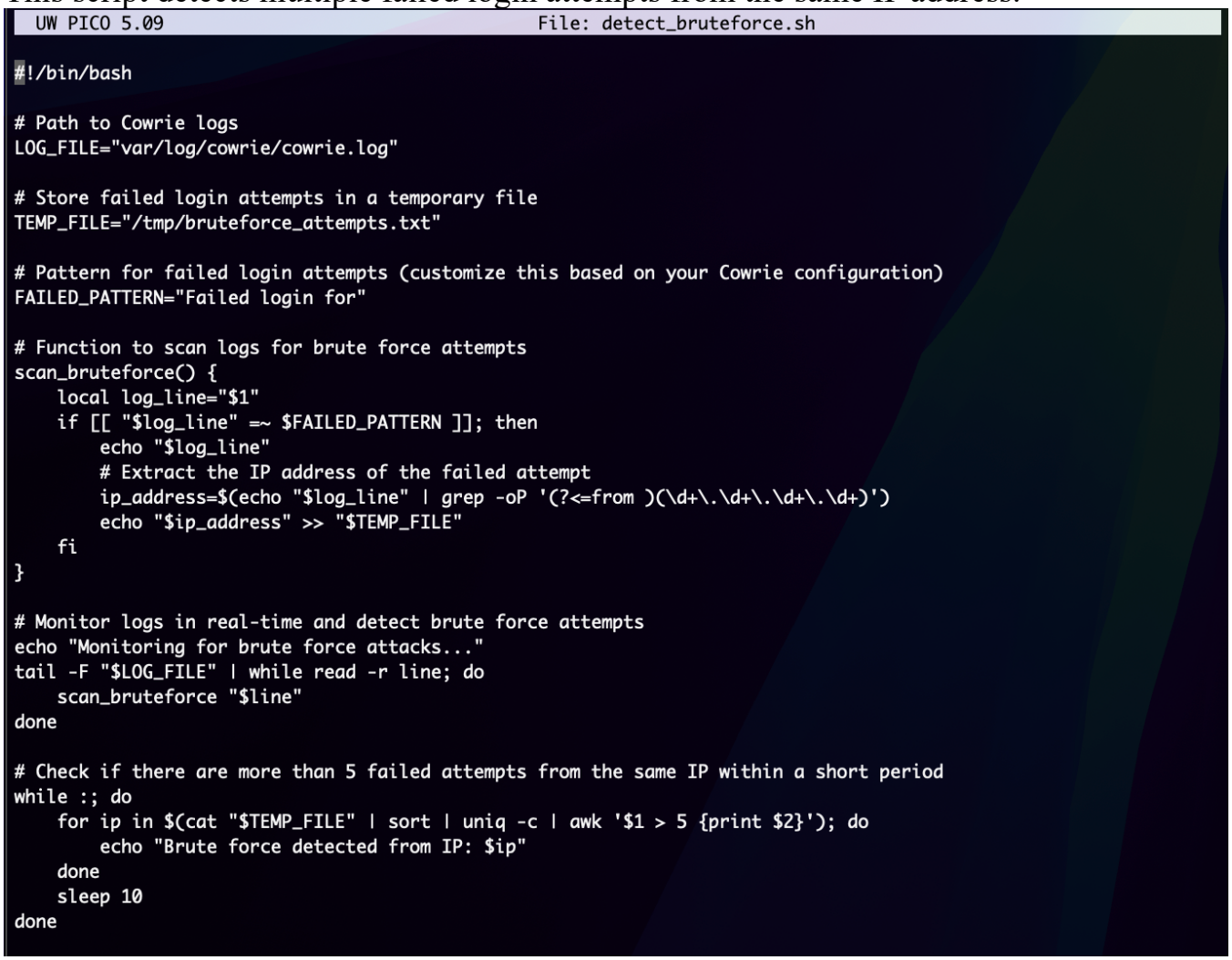
THE Automated PROGRAM

```
  UW PICO 5.09                          File: automate_detection.sh

#!/bin/bash

# Start Cowrie honeypot
echo "Starting Cowrie Honeypot..."
bin/cowrie start &  # Start Cowrie in the background

# Paths to Cowrie logs
LOG_FILE="var/log/cowrie/cowrie.log"

# Patterns for SQL Injection detection
SQL_PATTERNS=("UNION SELECT" "SELECT * FROM" "DROP TABLE" "INSERT INTO" "OR 1=1" "' OR '" "--" "#")

# Patterns for XSS detection
XSS_PATTERNS=("alert(" "<script>" "<img" "onerror=" "onload=" "document.cookie" "<iframe>")

# Patterns for CSRF detection
CSRF_PATTERNS=("action=" "GET /" "POST /" "delete=" "update=" "id=")

# Function to scan logs for patterns and save to log files
scan_logs() {
    local attack_type="$1"
    local log_line="$2"
    shift 2
    local patterns=("$@")

    # Convert attack_type to lowercase using tr command
    local attack_type_lower=$(echo "$attack_type" | tr '[:upper:]' '[:lower:]')

    for pattern in "${patterns[@]}"; do
        if [[ "$log_line" =~ $pattern ]]; then
            echo "$attack_type Attack Detected: $log_line"
            # Append detected attack to the appropriate log file
            echo "$log_line" >> "var/log/cowrie/${attack_type_lower}_attacks.log"
        fi
    done
}

# Ensure log files exist
mkdir -p var/log/cowrie
touch var/log/cowrie/sql_injection_attacks.log
touch var/log/cowrie/xss_attacks.log
touch var/log/cowrie/csrf_attacks.log

# Monitor logs in real-time
echo "Monitoring logs for SQL Injection, XSS, and CSRF attacks in $LOG_FILE..."
tail -F "$LOG_FILE" | while read -r line; do
    scan_logs "SQL Injection" "$line" "${SQL_PATTERNS[@]}"
    scan_logs "XSS" "$line" "${XSS_PATTERNS[@]}"
    scan_logs "CSRF" "$line" "${CSRF_PATTERNS[@]}"

^G Get Help      ^O WriteOut      ^R Read File     ^Y Prev Pg       ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where is      ^V Next Pg       ^U UnCut Text    ^T To Spell
```

DETECTION PROGRAMS

1- **Brute Force Attack Detection**

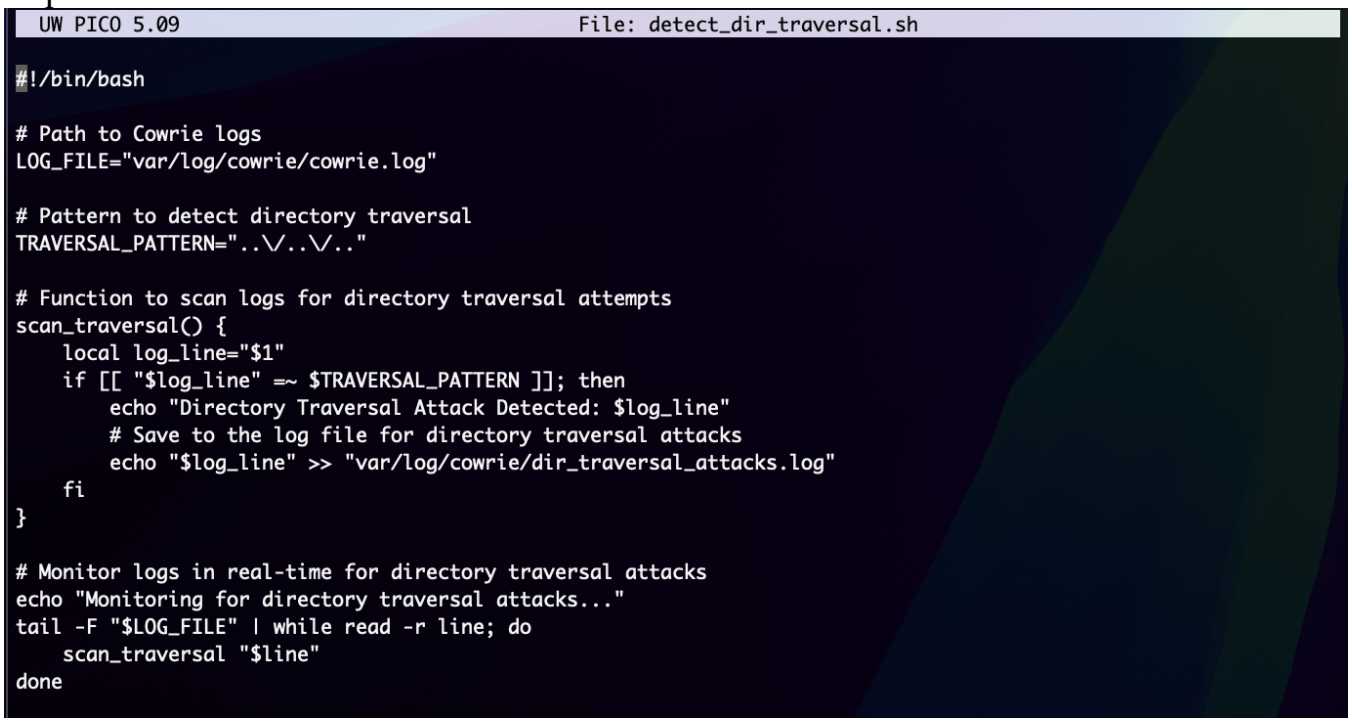
This script detects multiple failed login attempts from the same IP address.



```
  UW PICO 5.09                          File: detect_bruteforce.sh

#!/bin/bash

# Path to Cowrie logs
LOG_FILE="var/log/cowrie/cowrie.log"

# Store failed login attempts in a temporary file
TEMP_FILE="/tmp/bruteforce_attempts.txt"

# Pattern for failed login attempts (customize this based on your Cowrie configuration)
FAILED_PATTERN="Failed login for"

# Function to scan logs for brute force attempts
scan_bruteforce() {
    local log_line="$1"
    if [[ "$log_line" =~ $FAILED_PATTERN ]]; then
        echo "$log_line"
        # Extract the IP address of the failed attempt
        ip_address=$(echo "$log_line" | grep -oP '(?<=from )(\d+\.\d+\.\d+\.\d+)')
        echo "$ip_address" >> "$TEMP_FILE"
    fi
}

# Monitor logs in real-time and detect brute force attempts
echo "Monitoring for brute force attacks..."
tail -F "$LOG_FILE" | while read -r line; do
    scan_bruteforce "$line"
done

# Check if there are more than 5 failed attempts from the same IP within a short period
while :; do
    for ip in $(cat "$TEMP_FILE" | sort | uniq -c | awk '$1 > 5 {print $2}'); do
        echo "Brute force detected from IP: $ip"
    done
    sleep 10
done
```

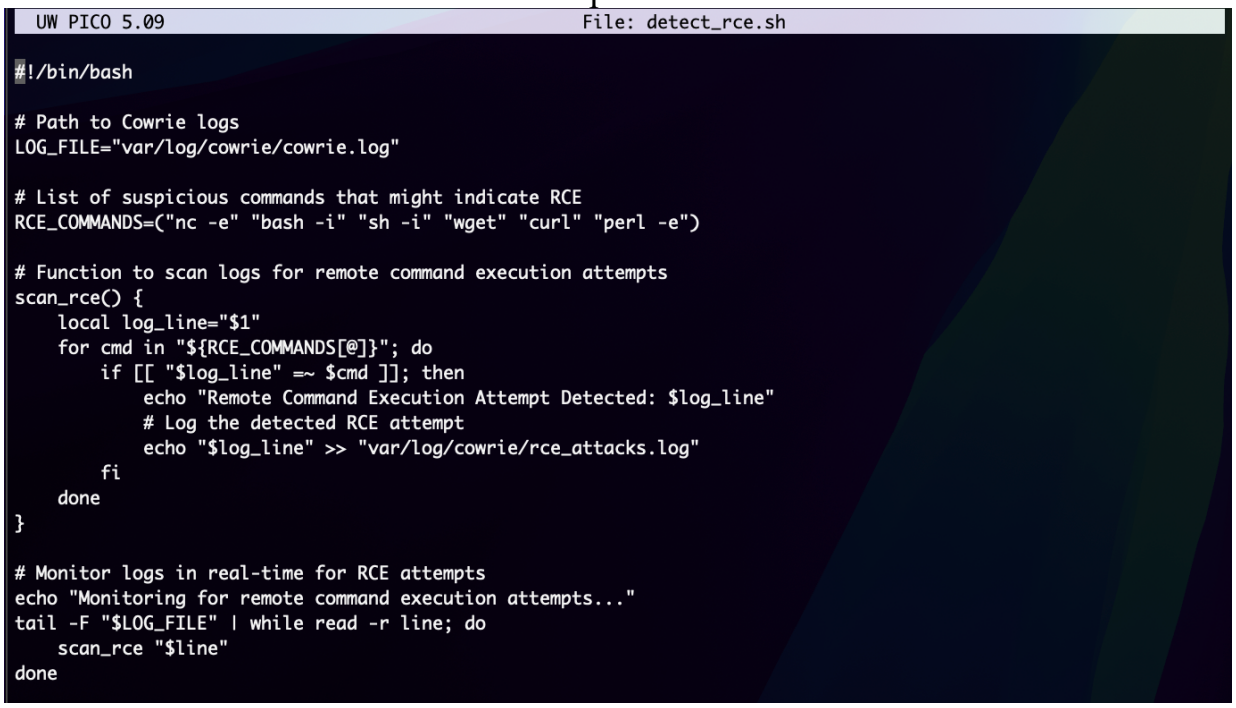
2- **Directory Traversal Attack Detection**

Directory traversal attacks try to access files outside the intended directory by using sequences like ../../.



3- **Remote Command Execution Detection**

Attackers may attempt to execute arbitrary commands remotely on the server. This script will detect unusual commands or attempts to execute shell commands.



4- **HTTP Request Flood Detection (DoS Attack)**

Denial of Service (DoS) attacks flood the target system with requests to overwhelm it.

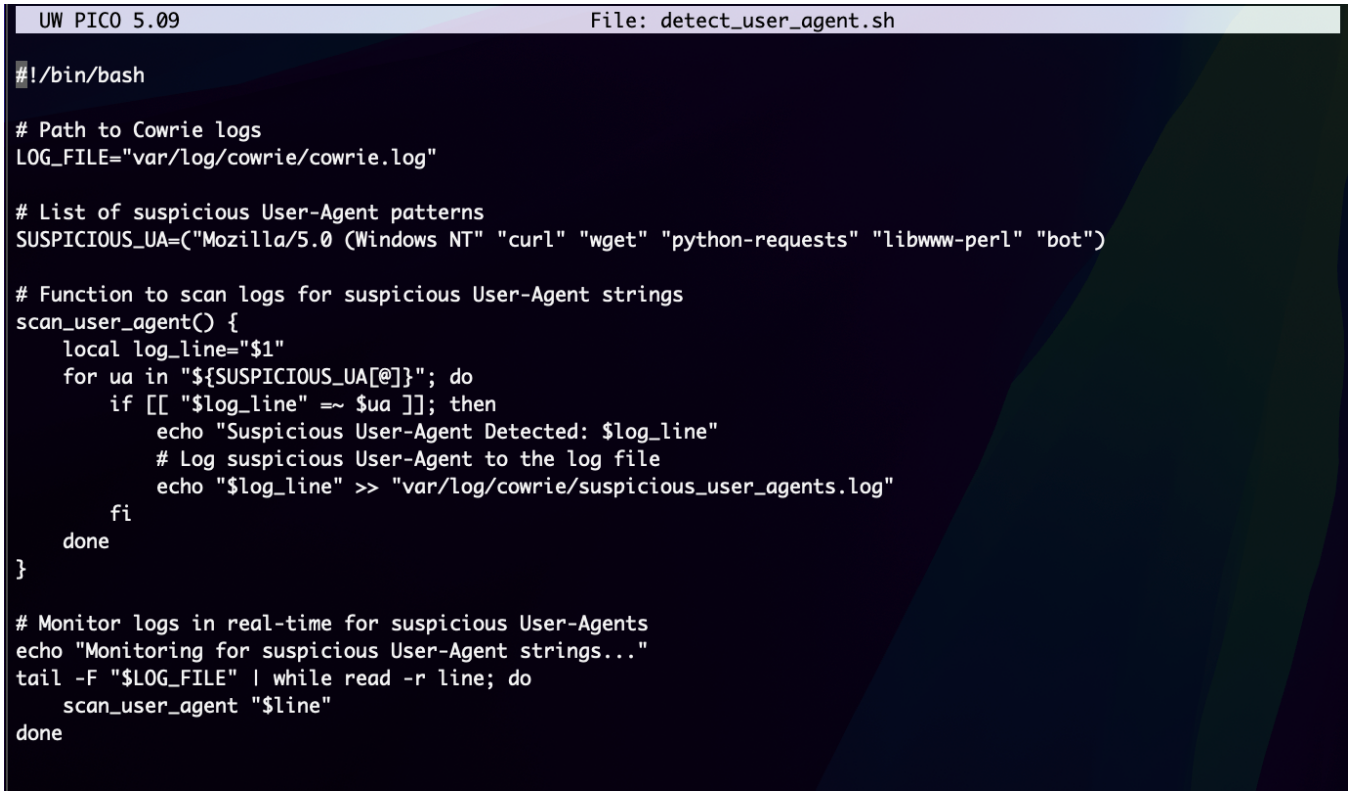
```
  UW PICO 5.09                                File: detect_dos.sh

#!/bin/bash

# Path to Cowrie logs
LOG_FILE="var/log/cowrie/cowrie.log"

# Store request counts per IP in a temporary file
TEMP_FILE="/tmp/dos_requests.txt"

# Threshold for request count
THRESHOLD=100

# Function to scan logs for repeated SSH connection attempts
scan_dos() {
    local log_line="$1"
    # Match connection attempts, and extract the IP address (adjust the pattern if necessary)
    ip_address=$(echo "$log_line" | grep -oE 'Connection from [0-9]+\.[0-9]+\.[0-9]+\.[0-9]+' | awk '{print $3}')
    if [ -n "$ip_address" ]; then
        echo "$ip_address" >> "$TEMP_FILE"
    fi
}

# Monitor logs in real-time for connection attempts
echo "Monitoring for Denial of Service (DoS) attacks..."
tail -F "$LOG_FILE" | while read -r line; do
    scan_dos "$line"
done &  # Run the above monitoring process in the background

# Check if any IP exceeds the threshold (e.g., 100 requests in a short period)
while :; do
    for ip in $(cat "$TEMP_FILE" | sort | uniq -c | awk '$1 > '$THRESHOLD' {print $2}'); do
        echo "DoS attack detected from IP: $ip"
    done
    sleep 10
done
```

LOGS:


```
c_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.failed","input":"while true","message":"Command not found: while true","sensor":"192.1
68.1.13","timestamp":"2024-11-26T19:36:28.675849Z","src_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.input","input":"while true; do ssh root@localhost -p 2222; done","message":"CMD: while
 true; do ssh root@localhost -p 2222; done","sensor":"192.168.1.13","timestamp":"2024-11-26T19:36:29.071941Z","sr
c_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.failed","input":"while true","message":"Command not found: while true","sensor":"192.1
68.1.13","timestamp":"2024-11-26T19:36:29.080366Z","src_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.input","input":"while true; do ssh root@localhost -p 2222; done","message":"CMD: while
 true; do ssh root@localhost -p 2222; done","sensor":"192.168.1.13","timestamp":"2024-11-26T19:36:29.269132Z","sr
c_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.failed","input":"while true","message":"Command not found: while true","sensor":"192.1
68.1.13","timestamp":"2024-11-26T19:36:29.280285Z","src_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.input","input":"while true; do ssh root@localhost -p 2222; done","message":"CMD: while
 true; do ssh root@localhost -p 2222; done","sensor":"192.168.1.13","timestamp":"2024-11-26T19:36:29.405527Z","sr
c_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.command.failed","input":"while true","message":"Command not found: while true","sensor":"192.1
68.1.13","timestamp":"2024-11-26T19:36:29.413662Z","src_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.log.closed","ttylog":"var/lib/cowrie/tty/11588e2206d83c0ebb5a087fbe3984cee09cecf31adc3c99ab64d
9f2353426d7","size":1074,"shasum":"11588e2206d83c0ebb5a087fbe3984cee09cecf31adc3c99ab64d9f2353426d7","duplicate":
false,"duration":179.9840681552887,"message":"Closing TTY Log: var/lib/cowrie/tty/11588e2206d83c0ebb5a087fbe3984c
ee09cecf31adc3c99ab64d9f2353426d7 after 179 seconds","sensor":"192.168.1.13","timestamp":"2024-11-26T19:39:20.836
982Z","src_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.session.closed","duration":183.12270402908325,"message":"Connection lost after 183 seconds","s
ensor":"192.168.1.13","timestamp":"2024-11-26T19:39:20.858576Z","src_ip":"127.0.0.1","session":"10a6644528dc"}
{"eventid":"cowrie.session.connect","src_ip":"127.0.0.1","src_port":52797,"dst_ip":"127.0.0.1","dst_port":2222,"s
ession":"d25e29567371","protocol":"ssh","message":"New connection: 127.0.0.1:52797 (127.0.0.1:2222) [session: d25
e29567371]","sensor":"192.168.1.13","timestamp":"2024-11-26T19:39:20.870631Z"}
{"eventid":"cowrie.client.version","version":"SSH-2.0-OpenSSH_9.8","message":"Remote SSH version: SSH-2.0-OpenSSH
_9.8","sensor":"192.168.1.13","timestamp":"2024-11-26T19:39:20.880624Z","src_ip":"127.0.0.1","session":"d25e29567
371"}
```

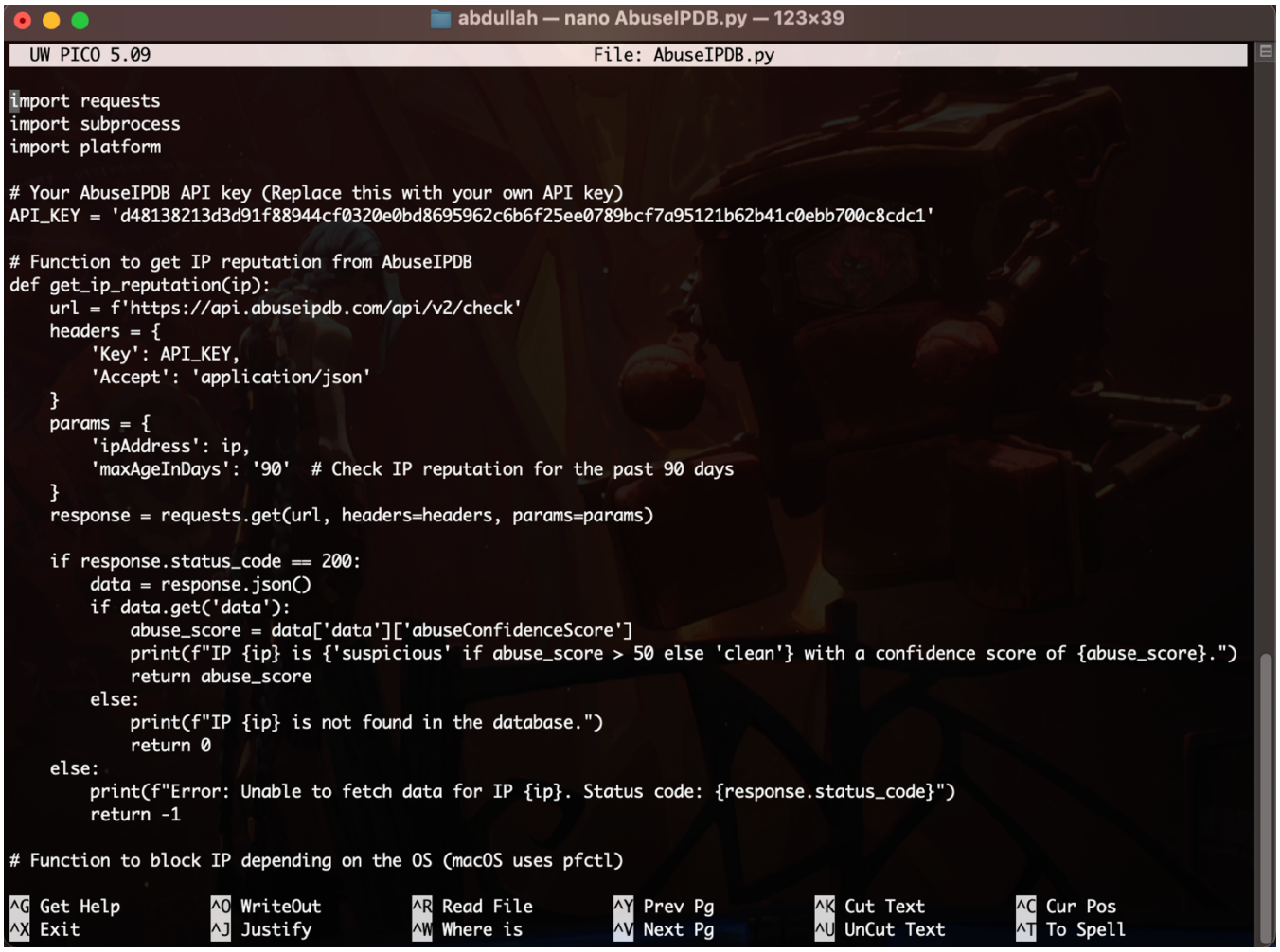
5- **Malicious User-Agent Detection**

Attackers often use custom User-Agent strings to disguise automated bots or exploit vulnerabilities.



```
  UW PICO 5.09                                File: detect_user_agent.sh

#!/bin/bash

# Path to Cowrie logs
LOG_FILE="var/log/cowrie/cowrie.log"

# List of suspicious User-Agent patterns
SUSPICIOUS_UA=("Mozilla/5.0 (Windows NT" "curl" "wget" "python-requests" "libwww-perl" "bot")

# Function to scan logs for suspicious User-Agent strings
scan_user_agent() {
    local log_line="$1"
    for ua in "${SUSPICIOUS_UA[@]}"; do
        if [[ "$log_line" =~ $ua ]]; then
            echo "Suspicious User-Agent Detected: $log_line"
            # Log suspicious User-Agent to the log file
            echo "$log_line" >> "var/log/cowrie/suspicious_user_agents.log"
        fi
    done
}

# Monitor logs in real-time for suspicious User-Agents
echo "Monitoring for suspicious User-Agent strings..."
tail -F "$LOG_FILE" | while read -r line; do
    scan_user_agent "$line"
done
```

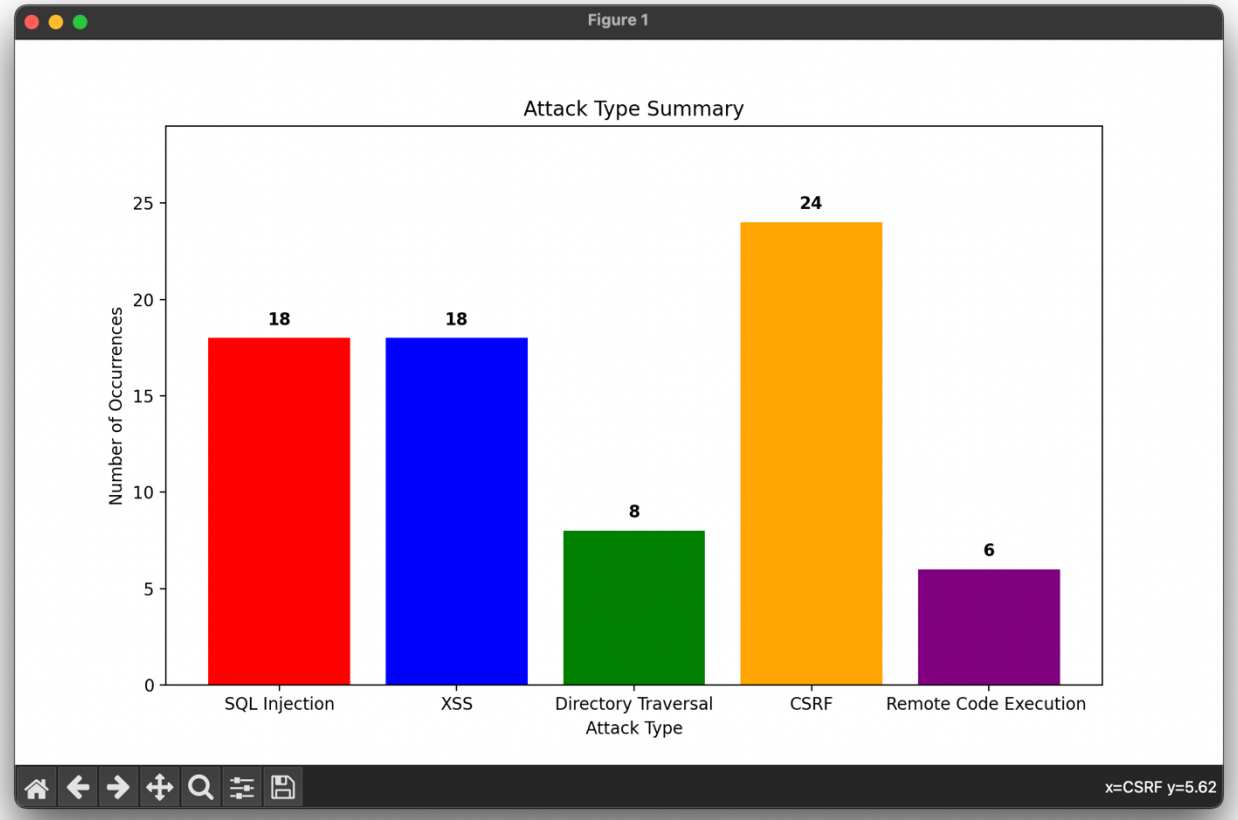
6-**Detect IPs**



This Threat Intelligence Integration program receives IP addresses then it checks The AbuseIPDB (Database) and if the IP is found dangers it automatically blocks it from ever establishing a connection with the device again.

Command to run: sudo python AbuseIPDB.py

Data Collected from the Honey-pot



We can see an attack summary of all the Logs combined through our program in a bar chart