HY-457: Introduction to Information Security Systems

Computer Science Department

Spring Semester 2025

Assignment 1

“Implementation of a Software Security Suite”

Tutorial: 20/03/2024
Deadline: 10/04/2024

Introduction

There has been a growing number of discussions regarding cybersecurity attacks targeting corporate networks, with increasing concerns about the sophistication and impact of such threats. Various security experts have reported that a particular strain of malware has been actively encrypting files, deleting critical backup data, and even removing virtual hard disk drives. Furthermore, there are indications that sensitive and valuable information has been exfiltrated, potentially leading to serious data breaches and financial losses.

Upon further investigation, this attack has been identified as the Medusa Ransomware Attack [1]. Ransomware is a type of malicious software designed to lock a user’s personal and corporate files, rendering them inaccessible until a ransom is paid to the attackers [2]. In these cases, the affected files are encrypted using advanced cryptographic techniques, and the only way to retrieve them is by obtaining a decryption key, which the attackers offer in exchange for a specified sum of money. Victims are often left with limited options; either comply with the ransom demand or risk losing their data permanently.

As a cybersecurity engineer working for a Security and Integrity company, your supervisor has informed you that one of the organization’s enterprise honeypots has been compromised by Medusa Ransomware. This presents a valuable opportunity to analyze the malware’s behavior and understand its attack vectors. You have been assigned the critical task of developing a software security suite that will assist researchers in studying this new ransomware attack. The goal is to gain insights into its methods, identify potential weaknesses, and explore ways to mitigate the threat.

1. Execution Monitoring

Your first task is to develop a monitoring execution environment designed to observe and analyze the behavior of Potentially Harmful Applications (**PHA**). This system will generate detailed reports on all significant actions performed by the application during its execution. Please note that the objective of your system is strictly observation and reporting. It should not interfere with the application's execution by blocking, deleting, or quarantining any detected threats. To achieve this, your system should support the following functionalities:

1. File Integrity Analysis: Compute and display the MD5 [3] and SHA-256 [4] cryptographic hashes of the PHA, ensuring a unique fingerprint for file integrity verification.
2. Static Analysis: Extract and store all readable strings found within the file (similar to the UNIX **`strings`** [5] command), providing insights into commands, URLs, or embedded text.
3. Dynamic Monitoring: Execute the PHA software and monitor its real-time behavior using **`ptrace`** [6].
4. System Call Tracking: Capture and log all system calls made by the PHA during execution, offering visibility into how it interacts with the operating system.
5. Network Traffic Monitoring: Observe and record the application's network activity in real time, focusing on unencrypted communication and potential external connections.
6. File System Interaction Tracking: Monitor all file system access attempts, detecting which files and directories the PHA interacts with during execution.
7. Post-Execution Analysis: Once the PHA completes its execution, your system should generate and store a comprehensive statistical report, summarizing key findings such as system calls invoked, accessed or modified files.

Implementation Details

For this task, you are only allowed to use the C standard library and the OpenSSL library for computing file hash values. More information about OpenSSL can be found in its man pages [3, 4]. Each time your application runs, it must compute the MD5 and SHA-256 hash values of the PHA (which is provided as a CLI argument).

You will also use **`ptrace()`**, a crucial system call that allows a process to observe and control the execution of another process. More details about ptrace can be found in its man page: https://man7.org/linux/man-pages/man2/ptrace.2.html. You must use the correct parameters to trace the executable (tracee) and its system calls. Your implementation should utilize **`fork()`** [7] to execute and monitor the target executable. While your system may internally work with system call numbers, the final report should also include their symbolic names.

Additionally, your system should monitor the PHA's network traffic and file system access in real time, focusing only on unencrypted traffic. Specifically, for network monitoring, you need to track the **`sendto()`** [8] system call. Each time it is invoked, check whether the sent data corresponds to an HTTP request. If so, extract and log the host (domain name) of the request. For

File System monitoring you need to track the **`read()`** [9] and **`write()`** [10] system calls, collecting the names of the accessed files.

Once the PHA completes execution, your application must generate a detailed execution report containing (i) a list of all system calls made, along with their call frequencies, (ii) the paths of files opened or modified by the PHA, including their last access timestamps, and (iii) the content sent by the PHA over the network.

A sample output is the following:

```
$ monitor ./pha.out

[INFO] [20-Mar-25 13:53:43] Application Started with argument ‘pha.out’
[INFO] [20-Mar-25 13:53:43] MD5 hash:    7c1cadb6887373dacb595c47166bfbd9
[INFO] [20-Mar-25 13:53:43] SHA256 hash: 6d89b6cdd650e689ef35710b7e......
[INFO] [20-Mar-25 13:53:43] Initialized data structures
[INFO] [20-Mar-25 13:53:45] Running...
[INFO] [20-Mar-25 13:53:45] Subprocess called ‘mmap’ for the first time
...
[INFO] [20-Mar-25 13:53:46] Subprocess interacted with host ‘google.com’
...
[INFO] [20-Mar-25 13:53:47] Subprocess made file access (read) `/etc/cfg`
[INFO] [20-Mar-25 13:53:47] Subprocess made file access (write) `/etc/cfg`
...
[INFO] [20-Mar-25 13:53:53] Subprocess exited
[INFO] [20-Mar-25 13:53:53] Stored report to ‘report.txt’
```

Implementation Steps

1. Develop a module which receives an executable as a CLI argument and executes it using **`fork()`**.
2. Develop a module which is able to trace the system calls of the tracee using **`ptrace()`**.
3. Implement the needed functionality in order to read the arguments the tracee passes to a system call (focus only on the required ones).
4. Implement the needed functionality so that you can extract the filename of a given file descriptor.
5. Develop a module which stores the statistical information that you need in memory. You are free to implement any data structure that you may need.

2. Network Traffic Analyzer

Your second task is to process and analyze the network traffic generated by an infected workstation. To accomplish this, you must develop a Network Traffic Analyzer that systematically monitors network traffic and inspects network and transport layer protocols to detect potential suspicious activity. This system will help security researchers gain insights into how the infected workstation interacts with external entities.

You are required to use the C programming language to develop this system. Unlike real-time network monitoring tools, your system will work in near real-time, meaning that it will analyze pre-captured network traffic instead of actively capturing live traffic. This approach allows for detailed analysis without affecting ongoing network operations. Your application should support the following functionalities:

1. File Hashing: Compute and display the MD5 and SHA-256 hash values of the provided network traffic file to verify its integrity.
2. Traffic Processing: Read and process the input file, extracting relevant network data.
3. Packet Analysis: Iterate through Ethernet, IP, TCP, and UDP packets, identifying key details about each one.
4. Communication Insights: List all IP addresses the workstation interacted with, including the number of packets sent, ports used, protocols used (e.g., TCP, UDP), network volume.

Implementation Details

For this task, you are going to use the **pcap** [11] library, a fundamental library that provides an interface to capture and process network traffic. More information about the library can be found in its man page: https://www.tcpdump.org/manpages/pcap.3pcap.html. You need to read the documentation in order to understand how the library works and how you can use it to achieve your goal. You are allowed to use any function or structure provided by this library but you are **not allowed** to use any other third-party library. You should focus on how you can process network traffic that has already been captured in a savefile. You can find test capture files here. You should only process Ethernet, IP, TCP and UDP packets.

Your application should accept two CLI arguments. The first argument will be the file containing the captured network traffic. This should be a PCAP file. The second argument will be an IP address that will be used as a filter. Your application should focus only on traffic generated by this IP address (i.e. source IP address). Please note that you are free to support either fixed IP addresses (e.g. 153.165.23.98), or addresses that follow the Classless Inter-Domain Routing (CIDR) notation [12] (e.g. 192.0.2.0/24), or both. Also note that your implementation should only support IPv4 addresses.

A sample output is the following:

```
$ analyze ./traffic.pcap 192.168.153.1

[INFO] [20-Mar-25 14:12:27] Application Started with argument
'traffic.pcap'
[INFO] [20-Mar-25 14:12:27] MD5 hash:
cd6bc10414b9d3fdb8fba52579f654ad
[INFO] [20-Mar-25 14:12:28] SHA256 hash:
11760f811e5e82d293ac01..........
[INFO] [20-Mar-25 14:12:28] Initialized data structures
[INFO] [20-Mar-25 14:12:28] Filtering traffic of 192.168.153.1

IP: 142.162.123.43
5 outgoing packets [15623 Bytes]
Source Ports: 19981, 20010, 20024, 20034, 20036
Destination Ports: 80
Protocols: TCP

IP: 192.168.153.130
40 outgoing packets [637193 Bytes]
Source Ports: 3372, 19407, 19938
Destination Ports: 20041, 20042, 20043
Protocols: UDP, TCP

...
```

Implementation Steps

1. Develop a module which parses the network traffic file and iterates over all captured packets. You are required to use the pcap library for this step.
2. Develop a module which parses the headers of a packet (e.g. IP, TCP) and extracts the required information from them (e.g. source IP address).
3. Develop a module that can store information about network hosts (e.g. IP address, number of packets, ports, protocols, etc). You are free to implement any data structure that you may need.
4. Change your implementation so your system can filter network traffic based on the source IP address and group network traffic based on the destination IP.

3. Secret Sharing System

Your third and final task is to ensure that highly sensitive files remain secure from ransomware by placing them into a secure digital vault. Access to the vault is controlled with a secret password, but you want to ensure that no single individual can access it. Instead, multiple trusted stakeholders must collaborate to unlock the vault. You are required to implement a solution using a **secret sharing scheme** [13].

Secret sharing works by splitting private information into smaller pieces - or shares - and then distributing those shares amongst a group or network. Each individual share is useless on its own but when all the shares are together, they reconstruct an original secret. The original secret in our case is the secret password that controls the vault. Requiring all shares to reconstruct the original secret every time the vault needs to be opened, seems impractical and inefficient. Instead, a threshold of minimum shares must be set to avoid unpredicted shareholder behavior, intended or not. This approach ensures that the vault can only be accessed when trusted individuals work together, enhancing security and preventing unauthorized access.

Implementation Details

For this task you are asked to implement a secret sharing mechanism that unlocks the digital vault when **at least three stakeholders** are present. There are ten members in the organization's board. Only if any three (or more) of them are present, the vault can be opened. Otherwise, it remains sealed. To share the password among the members of the board you have to implement a secret sharing method that relies on polynomial interpolation. More specifically you will write a C program that:

1. Constructs a 2nd degree polynomial $f(x) = a_2 \cdot x^2 + a_1 \cdot x + a_0$ where a_0 is the password and a_1, a_2 are randomly generated numbers. Note that if the secret has to be reconstructed by k entities, the polynomial degree must be k-1.
2. Gives each member of the board a tuple in the form of $(x_n, f(x_n))$. The first member of the board would take f(1) the second f(2), the third f(3), and the last one would take f(10). Note that f(0) results in $f(0) = a_2 \cdot 0 + a \cdot 0 + a_0 \Leftrightarrow f(0) = a_0$. Hence, f(0) is the secret password that is splitted into pieces and must not be shared.
3. Provides two operation modes. One that splits the password and shares it to the board members, and one that reconstructs the password given at least 3 shares.

```
$ vault split 9
```

Will use 9 as the secret password and create 10 points (x,f(x)). Each point is a share. Each of these shares are distributed to the board's members.

```
$ vault join (3,42) (5,84) (9,216)
```

Will recreate the password. Note that in order to obtain the password you must have already created the points from step 1. Each share must be in the form of (x,f(x)).

Sample Execution

Appropriate printing is required for both operation modes of the program. Specifically you have to print the created points for the given key. For the password reconstruction mode, you have to print the calculated password and the result of comparing it with the password that was given during the split phase.

```
./vault split 9
    Randomly selected a = 2, b = 5
    Point (1,16)
    Point (2,27)
    Point (3,42)
    Point (4,61)
    Point (5,84)
    Point (6,111)
    Point (7,142)
    Point (8,177)
    Point (9,216)
    Point (10,259)
```

```
./vault join (3,42) (5,84) (9,216)
    Share count is 3
    f(3) = 42
    f(5) = 84
    f(9) = 216
    Computed that:
    a = 2
    b = 5
    Password is: 9
```

Implementation Steps

1. Implement a module that computes the polynomial for each value x
2. Implement a module that parses the CLI arguments. Note that each share will be given in the form of a tuple (x,y) where y is the value of the polynomial for the respective x.
3. Implement a module that solves a system of three equations with three unknown variables.
4. Combine the modules of the previous steps and create the required applications.

Notes

1. You should create a different executable for each part of this assignment. In the end, you will submit the source code of three different applications.

2. This is not a group assignment. Each student should submit their own implementation and you are not allowed to work with each other. If you decide to use hosting services or version control systems (e.g. Git), do not forget to mark your repository as private.

3. Implement all the tasks of this assignment using the C programming language. You are free to develop and test your implementation on your personal device, however, the final version should work on CSD workstations.

4. You can use the course's mailing list for any questions related to this assignment. Please provide a clear subject when sending an email. Do not send any private messages to the teaching assistants. Other students may have the same question.

5. Do not send code snippets or files of your implementation to the mailing list. If you do so, your assignment will not be accepted and you will not be graded for the assignment.

6. For each task of this assignment, you need to provide a Makefile. The makefile should contain at least three rules. One that builds your system, one that runs it using your own test files and one that deletes build files (e.g. object files). Please clean the directories before submitting your assignment.

7. You should provide your own tests to demonstrate that your implementation is correct. This applies to all tasks and might involve creating simple executables.

8. You should provide a short README file that describes what parts of the assignment you have implemented, what you implemented differently and anything else that you consider important. Please keep this file relatively short.

9. Each task of this assignment should be implemented in a separate subdirectory.

10. Follow the steps described above and implement the assignment incrementally. This will be especially helpful for you. You can develop small building blocks and then integrate them into the final application. This will also help you with identifying and solving bugs.

11. Note that the submitted code will be tested for plagiarism using appropriate software.

12. You can submit the assignment by executing the command **turnin assignment_1@hy457 directory_name** where directory_name is the directory that contains the source code.

References

[1]https://www.bitdefender.com/en-us/blog/businessinsights/medusa-ransomware-a-growing-threat-with-a-bold-online-presence
[2] https://en.wikipedia.org/wiki/Ransomware
[3] https://en.wikipedia.org/wiki/MD5
[4] https://en.wikipedia.org/wiki/SHA-2
[4] https://www.openssl.org/docs/manmaster/
[5] https://linux.die.net/man/1/strings
[6] https://linux.die.net/man/2/ptrace
[7] https://linux.die.net/man/2/fork
[8] https://linux.die.net/man/2/sendto
[9] https://linux.die.net/man/2/read
[10] https://linux.die.net/man/2/write
[11] https://www.tcpdump.org/manpages/pcap.3pcap.html
[12] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing
[13] https://en.wikipedia.org/wiki/Shamir%27s_secret_sharing