HY-457: Introduction to Information Security Systems

Computer Science Department

Spring Semester 2026

Assignment 2

“**Implementation of a Software Security Suite**”

Tutorial: 26/03/2026
Deadline: 06/05/2026

Introduction

Modern computer systems run dozens or even hundreds of applications simultaneously. While this flexibility is powerful, it also creates significant security challenges. Applications and networks are constantly exposed to threats, from ransomware and data breaches to insider misuse. Malicious or compromised programs may access sensitive files, contact suspicious network hosts, or exfiltrate data without the user's knowledge. Detecting and preventing these attacks requires active monitoring of system behavior, network traffic, and file access.

You are a cybersecurity engineer working for a Security and Integrity company. One morning, the company notices unusual behavior on a developer’s workstation. Files are being accessed without authorization, unknown programs are making network connections, and sensitive configuration data might be leaking. Your supervisor has asked you to develop a lightweight monitoring tool for a Linux system. Your goal is to build components that observe system activity without modifying the applications themselves. This replicates the approach used by many security tools, such as host-based intrusion detection systems and endpoint monitoring platforms. Your tools will provide real-time insight into what a system is doing, helping prevent or mitigate harmful actions.

Your focus is:

- **File activity**. Track when files are opened, read, or written, log any denied access, and record metadata such as UID and file hash.
- **DNS activity** [1]. Intercept DNS queries from applications to discover which hosts they try to contact.
- **Network traffic**. Capture outgoing connection attempts and resolve remote IP addresses, both from live monitoring and packet capture logs.

1. Execution & Network 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. Dynamic Monitoring: Execute the PHA software and monitor its real-time behavior using **`ptrace`** [2].
2. 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.
3. Network Traffic Monitoring: Observe and record the application's network activity in real time, focusing on unencrypted communication and potential external connections.
4. Post-Execution Analysis: Once the PHA completes its execution, your system should generate and store a comprehensive statistical report, summarizing key findings (e.g., system calls invoked).

Implementation Details

For this task, you are only allowed to use the C standard library. You will make use of **`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()`** [3] 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 **`connect()`** [4] and **`sendto()`** [5] system calls. Your software should be able to capture information each time one of the two system calls is invoked. For the `connect()` system call you need to identify the IP address and the port the application connected to. For the `sendto()` system call you need to check whether the sent data corresponds to an HTTP request. If so, extract and log the host (domain name) of the request.

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, and (ii) the network traffic generated by the application (i.e., IP addresses, ports and domain names).

A sample output is the following:

```
$ ./security-suite --monitor ./pha.out

[INFO] [20-Mar-26 13:53:43] Application Started with argument 'pha.out'
[INFO] [20-Mar-26 13:53:43] Initialized data structures
[INFO] [20-Mar-26 13:53:45] Running...
[INFO] [20-Mar-26 13:53:45] Subprocess called 'mmap' for the first time
...
[INFO] [20-Mar-26 13:53:46] Subprocess interacted with host 'google.com'
...
[INFO] [20-Mar-26 13:53:53] Subprocess exited
[INFO] [20-Mar-26 13:53:53] Stored report to 'report.txt'
```

Next, your software should provide the appropriate functionality to enrich the IP address information collected in by performing reverse DNS resolution. After identifying the IP addresses contacted by the potentially harmful application, your task is to determine whether these addresses correspond to known domain names. To do this, you will use the `getnameinfo()` [6] function provided by the system networking libraries. This function allows you to convert a socket address structure into a human-readable hostname.

For each captured IP address, construct the appropriate socket address structure and call `getnameinfo()` to attempt a reverse lookup. If the lookup succeeds, record the resolved hostname along with the original IP address. If the lookup fails or no hostname is available, your program should handle the error and indicate that no reverse DNS entry was found. This step will help you better understand the infrastructure behind the network connections made by the application, and may reveal clues about the ownership or purpose of the contacted servers.

A sample output is the following:

```
$ ./security-suite --resolve ./report.txt

[INFO] [20-Mar-26 13:55:12] Application Started with argument 'report.txt'
[INFO] [20-Mar-26 13:55:12] Running...
[INFO] [20-Mar-26 13:55:12] Extracted 4 IP addresses from 'report.txt'
...
[INFO] [20-Mar-26 13:55:12] Address 95.154.242.56 corresponds to server15.webodns.com
...
[INFO] [20-Mar-26 13:55:13] Address 104.18.26.120 lookup failed: Name or service not known
...
[INFO] [20-Mar-26 13:55:13] Exiting...
```

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. 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.
5. Develop a module that utilizes `getnameinfo()` to perform reverse IP lookup.

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. Traffic Processing: Read and process the input file, extracting relevant network data.
2. Packet Analysis: Iterate through Ethernet, IP, TCP, and UDP packets, identifying key details about each one.
3. 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** [7] 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 [8] (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:

```
$ ./security-suite --analyze ./traffic.pcap 192.168.153.1

[INFO] [20-Mar-26 14:12:27] Application Started with argument
'traffic.pcap'
[INFO] [20-Mar-26 14:12:28] Initialized data structures
[INFO] [20-Mar-26 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. File Access Monitoring

Your third and final task is to ensure that highly sensitive files are not accessed or modified by the potentially harmful application. Towards that extent, you will need to develop an access control logging system that will keep track of all file accesses and modifications. For this task you will use the `LD_PRELOAD` [9] mechanism to dynamically intercept selected library calls made by an application at runtime. Your goal is to monitor file access activity and basic network resolution behavior without modifying the source code of the target application. To achieve this, you will implement a shared library that overrides specific standard library functions and logs relevant information whenever they are invoked.

First, intercept file-related operations by overriding the `fopen()` [10] and `fwrite()` [11] functions. Each time one of these functions is called, your wrapper function should record the event in a log file. For every file access, the log must contain the following information.

- UID of the user executing the program
- The name of the file being accessed
- Date of the event
- Time of the event
- Action performed (open, read, or write)
- A flag indicating whether the operation failed due to insufficient permissions or other access restrictions (i.e. action denied)
- A cryptographic hash computed over the file contents

For this task, you are only allowed to use the C standard library and the OpenSSL library for computing file hash values [12, 13]. The cryptographic hash of the file (i.e. fingerprint) should be computed after the file has been modified.

In addition to file monitoring, you will also intercept the `getaddrinfo()` [14] function to observe which domain names the application attempts to resolve. Your wrapper should log each hostname provided to `getaddrinfo`, allowing you to identify the remote services or domains that the monitored application is trying to contact.

Your shared library should be compiled as a shared object (.so) and loaded using the `LD_PRELOAD` environment variable before executing the target application. The resulting log file should provide a chronological record of both file access events and attempted domain name resolutions. When implementing your wrappers, remember to call the original library functions so that the application's behavior is preserved.

A sample output is the following:

```
$ LD_PRELOAD=./logger.so ./pha.out
...
$ cat ./logger.report.txt

UID   file_name      timestamp          Action    Denied    Hash
==========================================================================
==
1000  /tmp/foo.txt   2026-03-20 20:37  open      0
57dc9a450ce410...
1000  /tmp/foo.txt   2026-03-20 20:37  write     0
b87225ff7daf52...
1000  /var/back/1.gz 2026-03-20 20:41  write     1         cace0bb6e26d77…

Network Traffic
[2026-03-29 20:27:11] example.com
[2026-03-29 20:27:12] bbc.com
[2026-03-29 20:27:18] malicious-domain.com
```

Implementation Steps

1. Create a shared library that will be loaded before the standard C library using the LD_PRELOAD mechanism. This library will contain wrapper implementations of the functions you want to monitor. At first, your wrappers should simply print a message.
2. Implement wrapper functions for the required library calls. The wrappers should collect the required information and then invoke the original functions so that the program continues to operate normally. Within each wrapper, first dynamically resolve the address of the original function from the standard libraries
3. For every intercepted call, gather the required data (e.g. the user identifier (UID) of the running process, the file name being accessed, etc).
4. Develop a module that computes the hash value of the accessed file to uniquely identify its contents.
5. Store all collected information in a structured log file. Each log entry should correspond to a single intercepted call and contain the required fields. Ensure that entries are appended in chronological order.

Notes

1. For this assignment, you must submit two components: (i) an executable program that performs the required analysis and processing tasks described in the previous parts, and (ii) a shared monitoring library that is loaded using LD_PRELOAD.

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. 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.

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

11. You can submit the assignment on the course's eLearn page. Please submit a single .zip file that contains your source code.

References

[1] https://en.wikipedia.org/wiki/Domain_Name_System
[2] https://linux.die.net/man/2/ptrace
[3] https://linux.die.net/man/2/fork
[4] https://linux.die.net/man/2/connect
[5] https://linux.die.net/man/2/sendto
[6] https://linux.die.net/man/3/getnameinfo
[7] https://www.tcpdump.org/manpages/pcap.3pcap.html
[8] https://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing
[9] https://linux.die.net/man/8/ld.so
[10] https://linux.die.net/man/3/fopen
[11] https://linux.die.net/man/3/fwrite
[12] https://en.wikipedia.org/wiki/MD5
[13] https://en.wikipedia.org/wiki/SHA-2
[14] https://linux.die.net/man/3/getaddrinfo