HY-457: Assignment 2

Implementation of a Software Security Suite

Papadogiannakis Manos
papamano@csd.uoc.gr

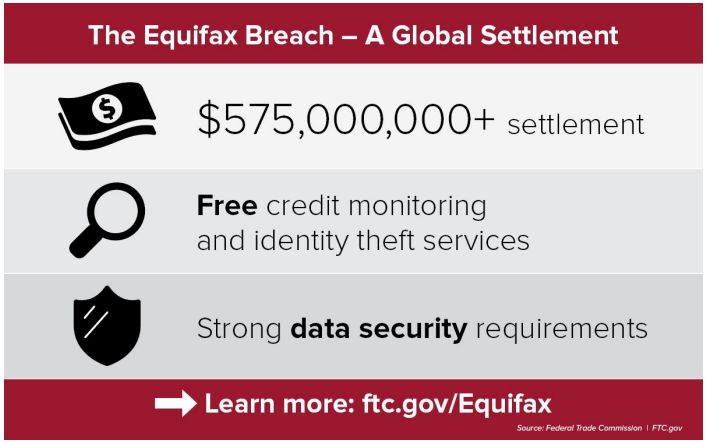
CS-457: Introduction to Information Security Systems
Computer Science Department
University of Crete

Outline

0. Motivation
1. Execution Monitoring
2. Network Traffic Analyzer
3. File Access Monitoring
4. Notes

Motivation

Real World Examples



Equifax to Pay $575 Million as Part of Settlement with FTC, CFPB, and States Related to 2017 Data Breach

https://www.ftc.gov/node/47878

Warnings (& Lessons) of the 2013 Target Data Breach

https://redriver.com/security/target-data-breach

Motivation

- **Attacks succeeded not because defenses didn't exist**
  - Systems weren't properly monitored, analyzed, or understood.
- **You are going to build tools that can detected/prevent these attacks**
  - Ideally

➢ **Network Monitoring**
  - Detecting suspicious connections

➢ **Traffic Analyzer**
  - Identifying pattern

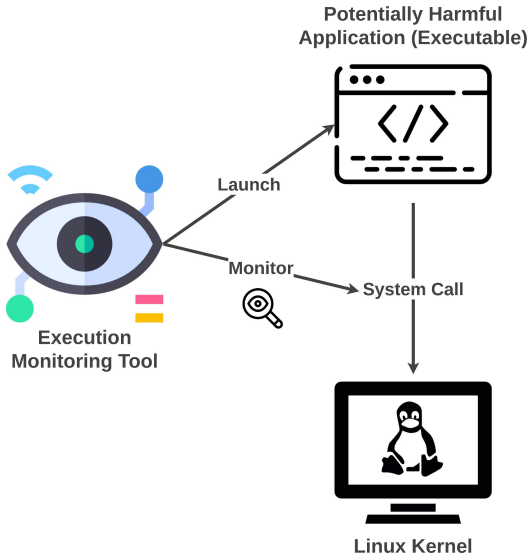
➢ **File Access Monitoring**
  - Detecting data misuse

Execution Monitoring

Execution Monitoring

- **Create a system that will monitor the behavior of an application (i.e. executable)**
- **You get an executable, you run it and report what it did**
  - The tracee will be executed!
- **Need to:**
  - **Monitor system calls**
  - Focus on network traffic
  - Create report with detailed information
- **Running:**

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



Tracing

- Use `ptrace()` to monitor system calls

```
PTRACE(2)              Linux Programmer's Manual              PTRACE(2)

NAME       top

       ptrace - process trace

SYNOPSIS       top

       #include <sys/ptrace.h>

       long ptrace(enum __ptrace_request request, pid_t pid,
                   void *addr, void *data);

DESCRIPTION       top

       The ptrace() system call provides a means by which one process
       (the "tracer") may observe and control the execution of another
       process (the "tracee"), and examine and change the tracee's
       memory and registers.  It is primarily used to implement
       breakpoint debugging and system call tracing.
```

Programmatic way to request a service from the operating system

Implementation Details

- **Whenever a system call is made, we need to know it (and log it)**
- **Deep inspection for 2 system calls**
  - `sendto()`
  - `connect()`
- **Read passed arguments** ← Depends on architecture!
  - Read registers
  - Read memory content
- **Not interested in exhaustive tests**
  - Don’t care of other ways one can contact the Web

Deep Inspection

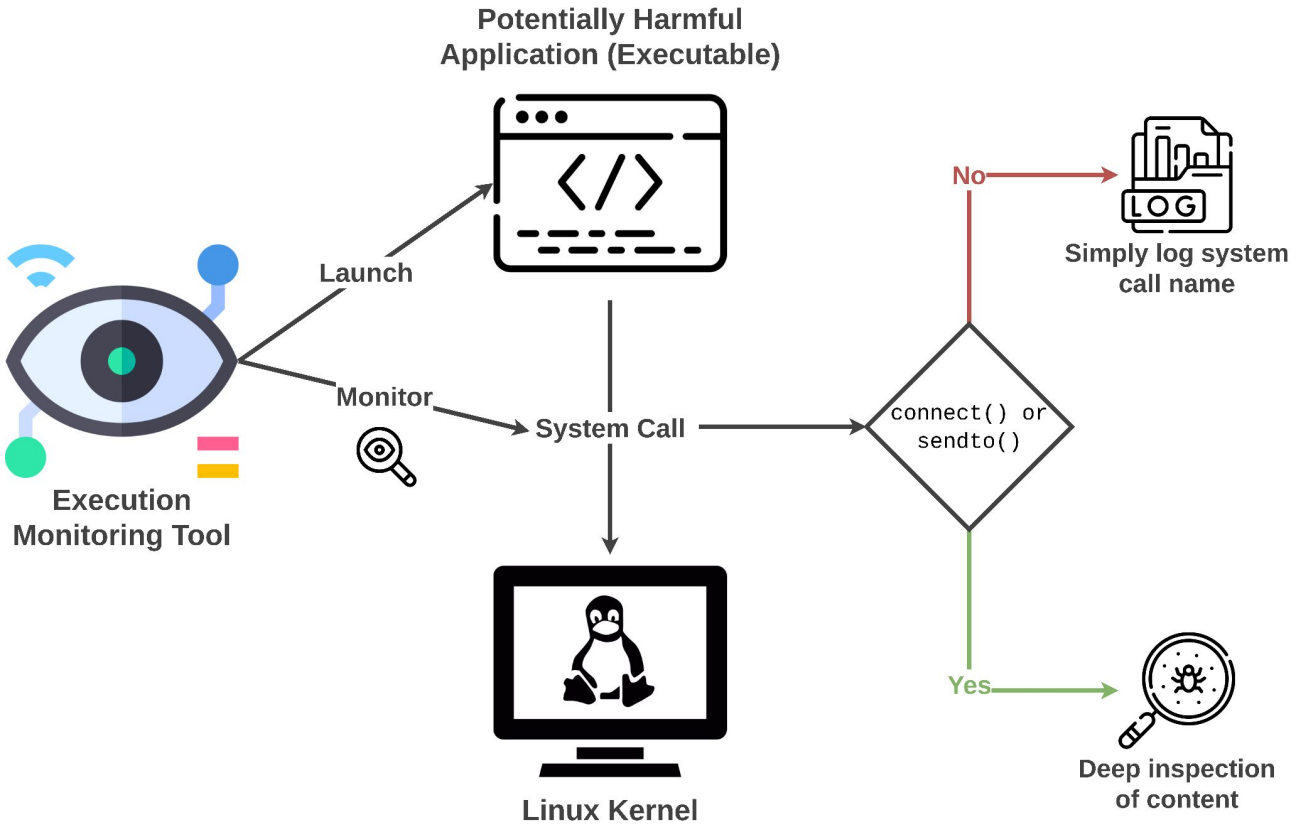
```c
/* 0. Message the will sent to server */
char *message = "Hello, server!";

/* 1. Create a UDP socket */
sockfd = socket(AF_INET, SOCK_DGRAM, 0);

/* 2. Set up the server address */
memset(&server_addr, 0, sizeof(server_addr));
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(8080);
inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

/* 3. Send the message */
sendto(sockfd,
       message,
       strlen(message),
       0,
       (struct sockaddr *)&server_addr,
       sizeof(server_addr));
```

Data: `Hello, server!`

Destination: *`127.0.0.1:8080`*

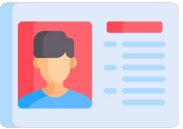
Deep inspection means we capture these arguments

Execution Flow



Implementation Details

- **Ptrace does not know system call symbolic names**
  - Works with system call numbers
- **Need to map names to numbers**
  - e.g., Syscall 0 is “`read`”. Syscall 5 is “`fstat`”



✓ **Important: System call numbers and their symbolic names may be different across architectures or kernel versions**
  - Your implementation should be compatible with CSD workstations!

Tracing

- **The tracee might be stopped at both the entry and the exit of system calls**
  - Need to carefully count calls
- **There are some system calls that are called when a program starts**
  - Not a bug
  - You will notice lots of `brk()`, `mmap()`, etc

Tracee

- You need to develop your own test applications
- You should submit enough tests to demonstrate the correct behavior of your system
- No need to create complex applications
  - But you need to test thoroughly!

```
#!/bin/python3
import requests

with open("temp.txt", "w") as file:
   file.write("hy457")

requests.get(url="http://google.com")
```

Example

```
$ ./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’
```

Example

```
$ cat report.txt

System Call Summary:
====================
brk (12): 47
mmap (9): 94
access (21): 1
openat (257): 260
getcwd (79): 2
gettid (186): 2
connect (42): 11
sendto (44): 4

Connection Summary:
===================
104.18.26.120:80
151.101.192.81:443
146.190.62.39: 80
```

```
Captured Data Summary:
======================
GET / HTTP/1.1
Host: www.google.com
User-Agent: python-requests/2.28.1
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

GET / HTTP/1.1
Host: example.com
Accept: */*
```

Reverse DNS lookup

- Determine the domain name associated with an IP address
- If the lookup succeeds you need to log the resolved hostname
- Utilize `getnameinfo()`
- Separate module in your application

Example

```
$ ./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...
```

You can test using `dig`

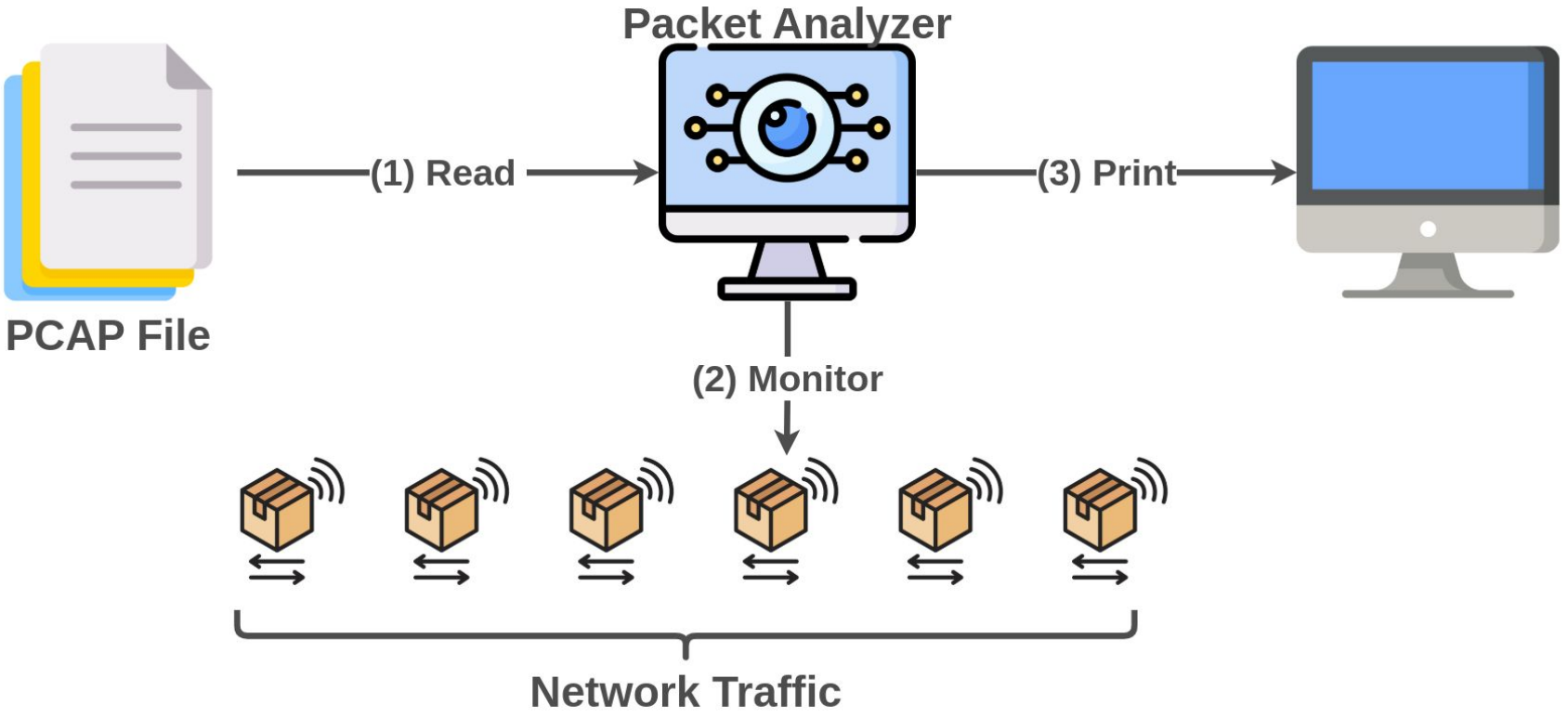
```
:~ $ dig -x 95.154.242.56

; <<>> DiG 9.18.44-1~deb12u1-Debian <<>> -x 95.154.242.56
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 16004
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;56.242.154.95.in-addr.arpa.    IN      PTR

;; ANSWER SECTION:
56.242.154.95.in-addr.arpa. 10800 IN    PTR     server15.webodns.com.

;; Query time: 63 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Tue Mar 24 13:26:59 EET 2026
;; MSG SIZE  rcvd: 89
```

Network Traffic Analyzer

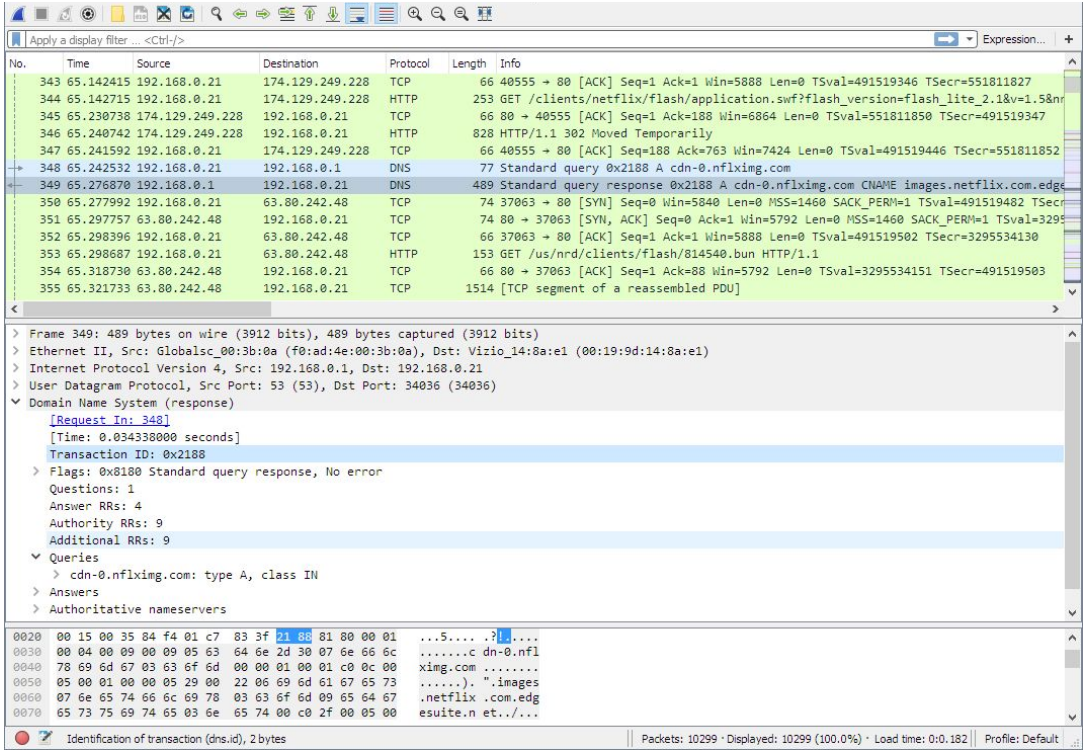
Network Traffic Analyzer

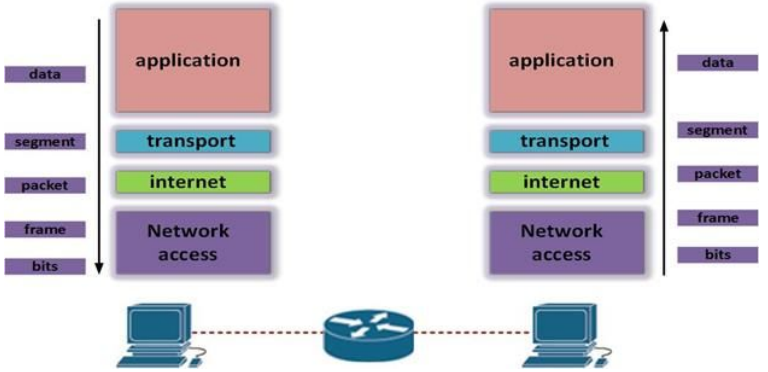
- Create a packet analyzer for network traffic
- Monitors network traffic, analyzes protocols and prints statistical information
- Focus only on captured traffic

Overview


Packet Analyzer
(1) Read
(3) Print
PCAP File
(2) Monitor
Network Traffic

Wireshark





https://www.ccnablog.com/tcpip-and-the-osi-model/

Overview

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

1. Read pcap file containing network traffic
2. Identify packets that match IP address
3. Collect information
4. Print statistics

Monitoring

- **Use the pcap library to process captured traffic**
  - No need to focus on real-time capture
  - You can test if you want (only on your personal workstation)

PCAP(3PCAP) MAN PAGE

Section: Misc. Reference Manual Pages (3PCAP)
Updated: 9 September 2020
Index Return to Main Contents

NAME

pcap - Packet Capture library

SYNOPSIS

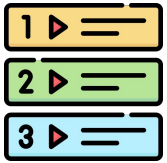
```
#include <pcap/pcap.h>
```

DESCRIPTION

The Packet Capture library provides a high level interface to packet capture systems. All packets on the network, even those destined for other hosts, are accessible through this mechanism. It also supports saving captured packets to a ``savefile'', and reading packets from a ``savefile''.

Monitoring

- **You need to iterate over packets**
  - e.g. `pcap_loop()`



- **You need to parse protocol headers to get IP addresses and ports**

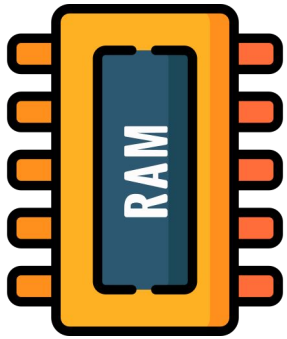
- **The headers you need to parse are:**
  - Ethernet
  - IP
  - TCP
  - UDP

Don’t forget about encapsulation!

Implementation Details

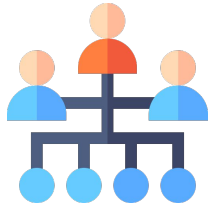
You need to store:

a. Source and destination IP addresses
b. Source and destination Ports
c. Type
d. Packet length



IP Addresses

- **Focus only on IPv4 addresses**
- **Dotted decimal notation**
  - 192.168.1.1
- **CIDR Notation**
  - Network Prefix and Host Identifier
  - 192.168.1.0/24 is equivalent to
    192.168.1.* or 192.168.1.0 - 255



Example

```
$ ./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

...
```

File Access Monitoring

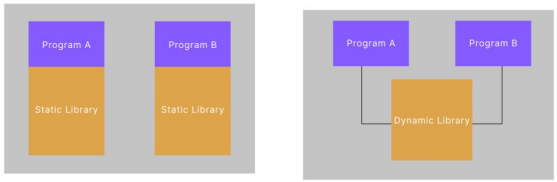
File Access Monitoring

- **Develop an Access Control Logging System**
- **Need to track each file access**
  - For this assignment only `fopen()` and `fwrite()`
- **Need to somehow know when these functions are called**
  - No ptrace in this task
  - Need another solution...

LD_PRELOAD

- **LD_PRELOAD allows you to inject your own code into a program at runtime**
  - Override functions it uses
- **Use dynamic linker to load your library before standard libraries**
  - Works on dynamically linked binaries
- **Control a programs behavior without changing its code**
  - No need to modify or recompile the target program
- **Can be used to override standard functions**
  - e.g., `fopen()`, `fwrite()`, `getaddrinfo()`

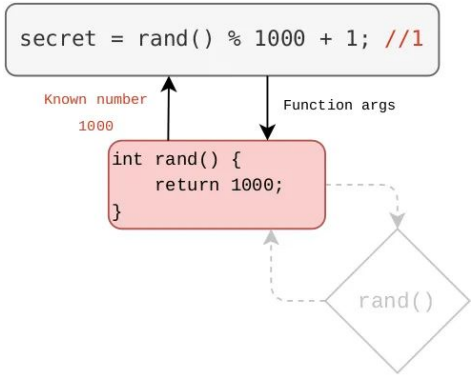
Static linking vs Dynamic linking



https://www.linkedin.com/pulse/static-libraries-vs-dynamic-ignacio-peralta

LD_PRELOAD (II)

- **Uses environment variable**
  - `LD_PRELOAD=yourlib.so ./program.out`
- **Monitoring:**
  - Log file access
  - Track network connections
- **Concept:**
  - Program calls `fopen("file.txt")`
  - Linker ensures your hooked `fopen()` is called
  - Log & Analyze
  - **Real `fopen()` is called**



https://infosecwriteups.com/a-gentle-introduction-to-function-hooking-using-ld-preload-1714124a6eb9

Example

1. Hook open()

```
#define _GNU_SOURCE
#include <stdio.h>
#include <dlfcn.h>

int open(const char *path, int flags, ...) {

    printf("[LD_PRELOAD] Opening file: %s\n", path);

    int (*orig)(const char*, int) = dlsym(RTLD_NEXT, "open");

    return orig(path, flags);

}
```

2. Compile library

```
gcc -shared -fPIC -o monitor.so monitor.c -ldl
```

3. Example Program

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>

int main() {

    int fd = open("example.txt", O_RDONLY);

    if (fd < 0) {

        perror("open failed");

        return 1;

    }

    printf("File opened successfully!\n");

    close(fd);

    return 0;

}
```

4. Run with LD_PRELOAD

```
$ LD_PRELOAD=./monitor.so ./test.out

[LD_PRELOAD] Opening file: example.txt
File opened successfully!
```

Logging

- **Every time one of the target functions is called, you need to log it**

- **What information do we want to know?**
  - 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

Example

```
$ 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
```

Hash Generators

- **Need to compute the hash value of all files**
  - Acts like a fingerprint or file signature
- **Allowed to use OpenSSL**
  - No need to reinvent the wheel
- **Read the docs!**
  - MD5, SHA256

```
$ cat my_secret_password.txt
Foobar

$ md5sum my_secret_password.txt
1b826051506f463f07307598fcf12fd6
```

Notes

Notes

- **Your final implementation should be a one executable and one shared library**
  - Many source code files
- **Follow the execution instructions**
  - e.g. CLI arguments, arguments order
- **Allowed to use mentioned libraries**
  - OpenSSL, LibPCAP

libcurl

- **You might want to use libcurl to develop demo applications**
- **Unfortunately it is not installed on CSD workstations**
  - You will get a `fatal error: curl/curl.h: No such file or directory`
- **Library is installed on the HY-457 home directory**
- **You can compile your applications using the following command:**

```
gcc -I/home/misc/courses/hy457/libcurl/lib/include
-L/home/misc/courses/hy457/libcurl/lib/lib test.c -lcurl
```

Turnin

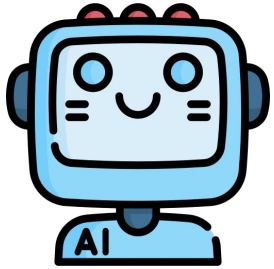
What to submit:

1. Source files
2. Test programs
3. Makefile
4. README

➢ Single **.zip** file submitted on eLearn

AI-Generated Code

- **Submitting AI-Generated code is not allowed**



- **The goal of this assignment is for you to learn**
  - We don’t need an AI agent to solve this assignment for us
  - We know that solutions exist
  - Your focus should be on learning the principles and techniques

- **Code review exam at the end of the semester**
  - You must be able to explain your implementation and design decisions
  - Not understanding your own code may result in **penalties**

Credit

Icons from FlatIcon, made by Freepik

Thank You!

papamano@csd.uoc.gr

Questions?