HY-457: Assignment 1

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. Secret Sharing System
4. Notes

Motivation

Motivation

TechCrunch

Lee Enterprises ransomware attack halts freelance and contractor payments

The ransomware attack is affecting Lee's ability to pay outside vendors, including freelancers and contractors, TechCrunch has learned.

1 week ago

The Hacker News

Medusa Ransomware Hits 40+ Victims in 2025, Demands $100K–$15M Ransom

The threat actors behind the Medusa ransomware have claimed nearly 400 victims since it first emerged in January 2023, with the financially...

5 days ago

The Register

Qilin ransomware gang boasts of cyberattacks on cancer clinic, Ob-Gyn facility

Qilin – the "no regrets" ransomware crew wreaking havoc on the global healthcare industry – just claimed responsibility

6 days ago

The Record from Recorded Future News

Tata Technologies reports ransomware attack to Indian stock exchange

Tata Technologies reports ransomware attack to Indian stock exchange. Indian multinational engineering company Tata Technologies recently dealt...

1 month ago

Motivation

- **Hypothetical attack at a corporate infrastructure**
- **Ransomware: encryption to hold a victim's information at ransom**
  - You need to pay to access your data

- **Develop tools that will help study an attack**

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:**
  - Compute hash value
  - Extract strings
  - **Monitor system calls**
  - Create report with detailed information
- **Running:**

```
$ ./monitor application.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

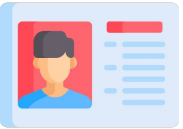
The **ptrace**() system call provides a means by which one process (the "tracer") may observe and control the execution of another process (the "tracee"), and examine and change the tracee's memory and registers. It is primarily used to implement breakpoint debugging and system call tracing.

Tracing

- **System calls:**
  - Programmatic way to request a service from the operating system
- **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

Implementation Details

- **Ptrace does not know system call symbolic names**
  - Works with system call numbers
- **Need to map names to numbers**



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

Implementation Details

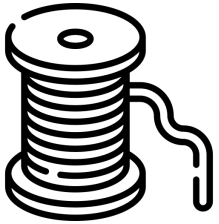
- **Deep inspection for 3 system calls**
  - sendto()
  - read()
  - write()

- **Read passed arguments**
  - Read registers
  - Read memory content

Depends on architecture!

- **Not exhaustive tests. Don’t care of other ways one can open a file**

Implementation



- **Need to examine content of files**
  - What if they are binary?
  - Not allowed to use the `strings` utility tool!

- **Need to extract all strings from file**
  - String == Sequence of **printable** characters

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

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 with a binary file

```
#!/bin/python3
import requests

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

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

Example

```
$ ./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'
```

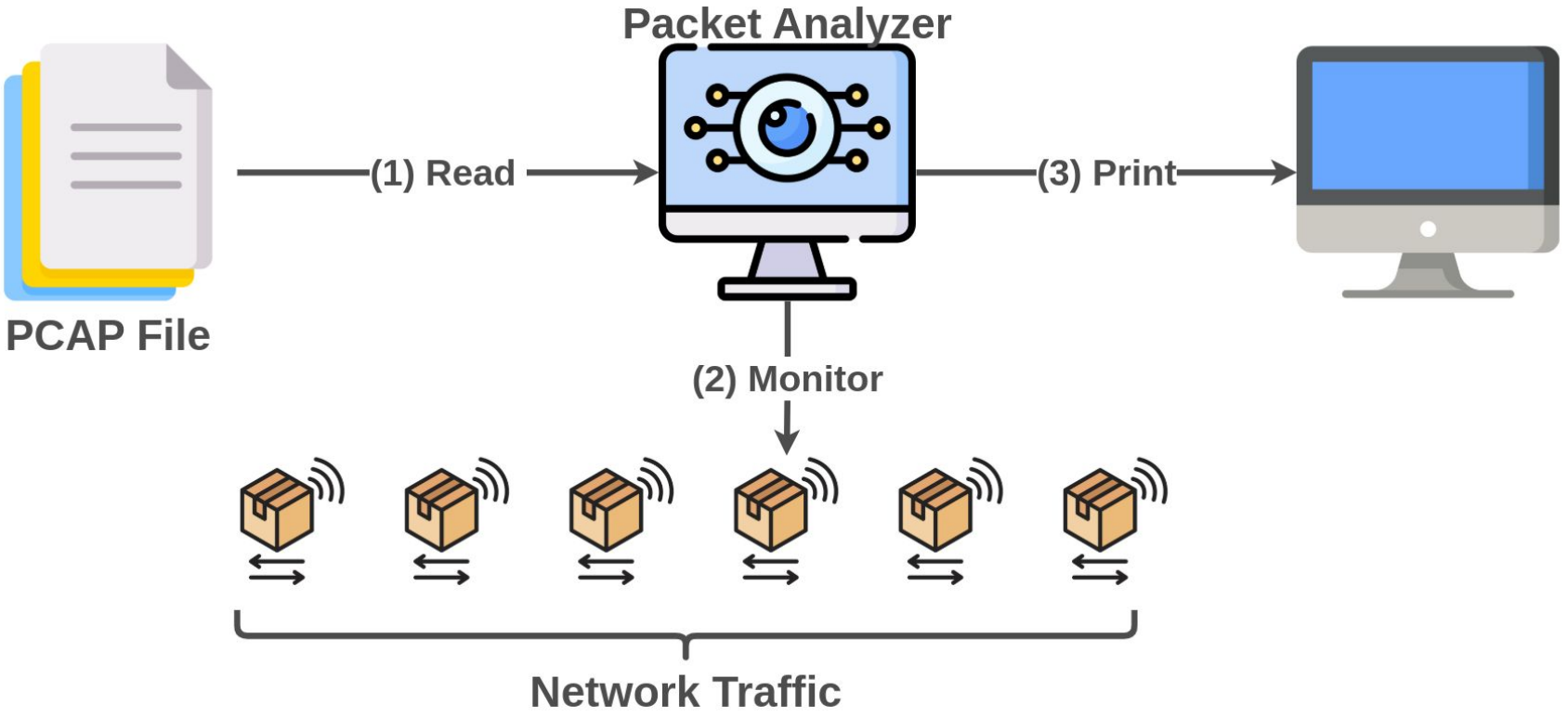
Network Traffic Analyzer

Network Traffic Analyzer

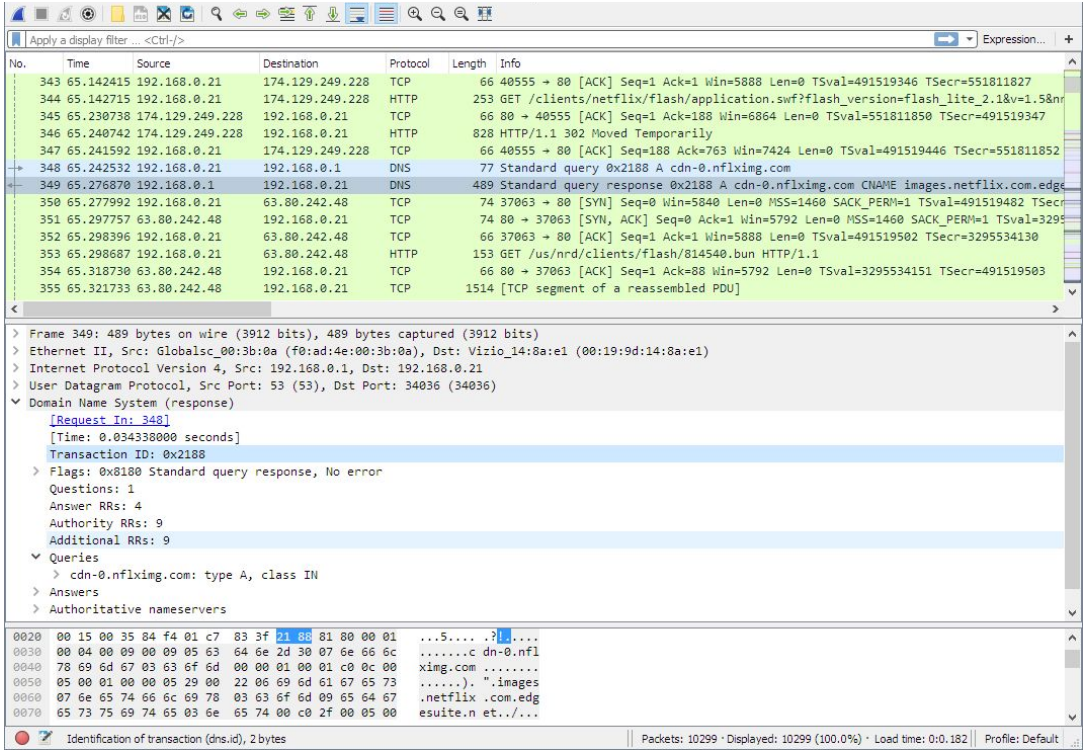
- Create a packet analyzer for network traffic

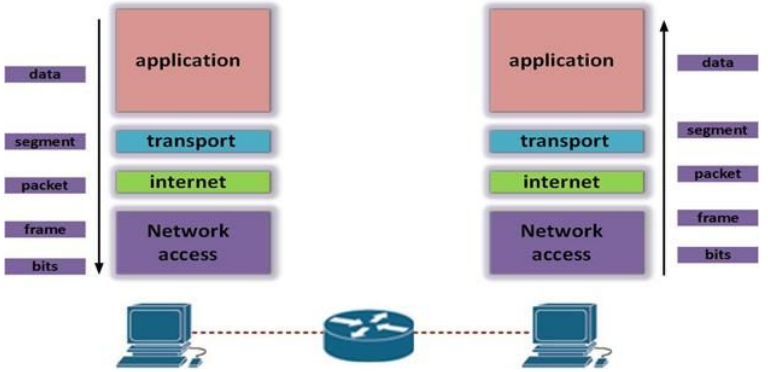
- Monitors network traffic, analyzes protocols and prints statistical information

Overview


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

Wireshark





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

Overview

```
$ 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

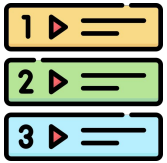
```
#include <pcap/pcap.h>
```

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**

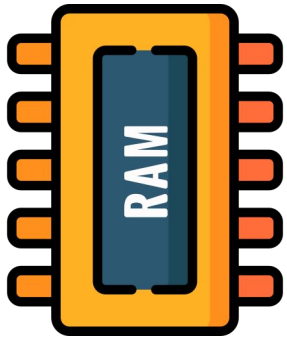
- **The headers you need to parse are:**
  - Ethernet
  - IP
  - TCP
  - UDP

Don’t forget about encapsulation!

Implementation Details

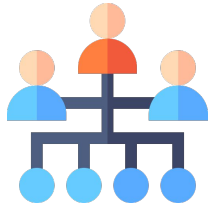
You need to store:

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

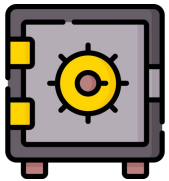
```
$ analyze ./traffic.pcap 192.168.153.1

[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

...
```

Secret Sharing

Motivation

- **Place important documents inside a “vault”**
  - Files inside the vault are encrypted and safe

- **No single individual can open the “vault” on their own**



Secret Sharing System

- **Implement a secret sharing mechanism**
  - In this case the secret would be the encryption key
  - No need to implement encryption

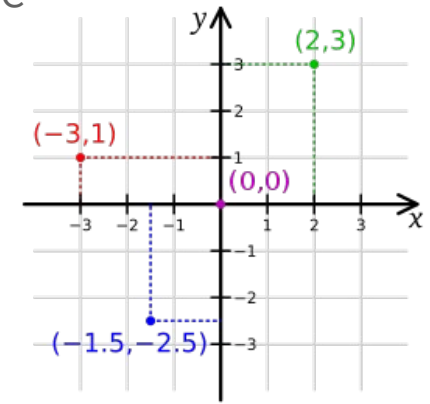
- **Distribute a secret among a group so that the secret cannot be revealed unless X people are present**

Overview

- Assume there are three friends: Alice, Bob and Carol
- The three people will share a secret number “c” by each taking a piece of the number
- Only when all three pieces are presented then all of them are able to reconstruct the secret number “c”

Introduction

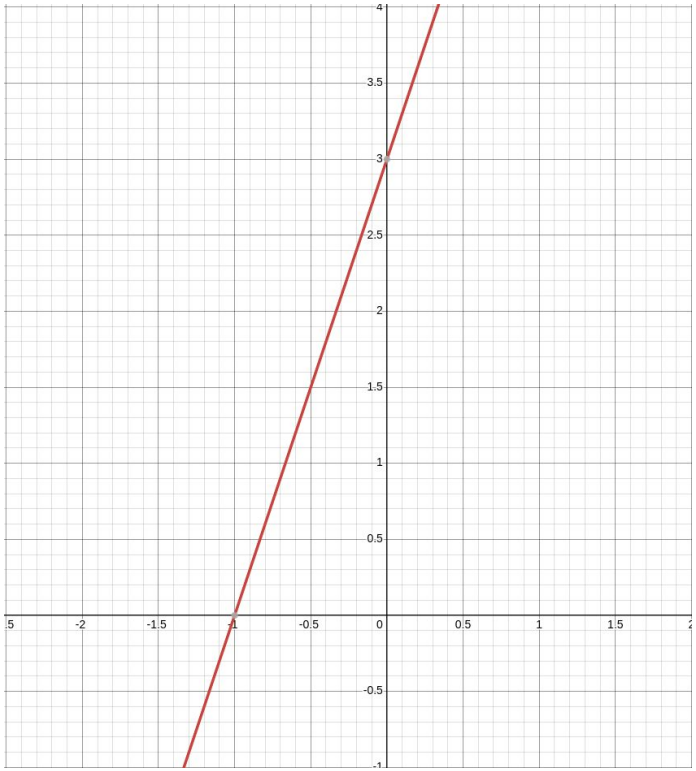
- **How can we achieve this?**
  - Let's assume the secret we want to share is a Euclidean line
- **How many lines are there?**
  - Infinite
- **What defines a line?**
  - Two points
  - We need to know them to reconstruct the line
- **Can one person on their own reconstruct the line?**



https://en.wikipedia.org/wiki/Euclidean_plane

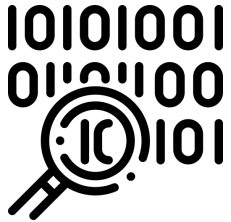
Euclidean Plane

$y = 3x + 3$



Simple Example

- **Let's assume that Alice and Bob want to share the secret number 72**
  - Since they are 2, the polynomial degree is 1
- **They randomly choose “a” to be 14 and “b” is the secret number**
  - $f(x) = a \cdot x + b = 14 \cdot x + 72$
- **They calculate f(1) = 86 and f(2) = 100**
  - Alice gets (1, 86)
  - Bob gets (2, 100)



Simple Example

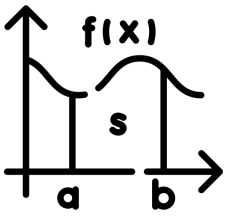
- To reconstruct the secret, they present their points and reconstruct the polynomial

$$\left.\begin{matrix} 86 = a + b \\ 100 = 2a + b \end{matrix}\right\} \left.\begin{matrix} a = 86 - b \\ 100 = 2(86 - b) + b \end{matrix}\right\} \left.\begin{matrix} 100 = 172 - 2b + b \end{matrix}\right\} \left.\begin{matrix} -72 = - b \end{matrix}\right\} b = 72$$

Implementation

- **Secret Sharing achieved by constructing the polynomial**
  $f(x) = a \cdot x^2 + b \cdot x + c$

- **Each person will take a point of the polynomial**
  - Alice: (1, f(1))
  - Bob: (2, f(2))
  - Carol: (3, f(3))



- **When all 3 points are presented, they reconstruct the polynomial to retrieve secret number “c”**

Implementation Details

- **Using the split option and a secret number the program generates the individual points**
  - $ ./vault split 9
    > (1, 16), (2, 27), (3, 42)
- **Using the join option and the 3 points, the program reconstructs the secret number**
  - $ ./vault join (1, 16), (2, 27), (3, 42)
    > 9
- **Generalize your solution for N friends.**
  - When any three present their points they are able to reconstruct the secret

Implementation Details

- **Allowed to look up how to solve system of linear equations with three variables**
  - Our focus is not maths ⇢ Cramer is your friend
- **No need to create parser for unlock function**
  - Use any format you like and assume the input will always be correct
- **Feel free to generate random numbers any way you like**
  - srand, rand
  - /dev/urandom

Advanced Example

- **Select a secret shared number “c”**
  - E.g. c is **9**
- **$f(x) = 2 \cdot x^2 + 5 \cdot x + 9$ where a, b were randomly generated**
- **When 3 shares (x_1, x_2, x_3) are present:**
  - $f(x_1) = a \cdot x_1^2 + b \cdot x_1 + c$ $\quad f(1) = 16$ $\quad 16 = a + b + c$ (1)
  - $f(x_2) = a \cdot x_2^2 + b \cdot x_2 + c$ $\quad f(2) = 27$ $\quad 27 = 4 \cdot a + 2 \cdot b + c$ (2)
  - $f(x_3) = a \cdot x_3^2 + b \cdot x_3 + c$ $\quad f(3) = 42$ $\quad 42 = 9 \cdot a + 3 \cdot b + c$ (3)

Advanced Example

$$
\begin{aligned}
16 &= a + b + c && (1)\\
27 &= 4\cdot a + 2\cdot b + c && (2)\\
42 &= 9\cdot a + 3\cdot b + c && (3)
\end{aligned}
$$

(3) - 3·(1)

$$
\begin{aligned}
42 &= 9\cdot a + 3\cdot b + c\\
-3\cdot 16 &= 3\cdot a + 3\cdot b + 3\cdot c\\
\hline
-6 &= 6\cdot a + 0 - 2\cdot c \quad \Leftrightarrow \quad \mathbf{a = (2\cdot c - 6)/6}
\end{aligned}
$$

$$
\begin{aligned}
16 &= a + b + c && (1)\\
27 &= 4\cdot a + 2\cdot b + c && (2)\\
42 &= 9\cdot a + 3\cdot b + c && (3)
\end{aligned}
$$

(2) - 2·(1)

$$
\begin{aligned}
27 &= 4*a + 2*b + c\\
-2*16 &= 2*a + 2*b + 2*c\\
\hline
-5 &= 2*a - c \quad \Leftrightarrow \quad \mathbf{a = (c - 5)/2}
\end{aligned}
$$

Advanced Example

- **We have computed that:**
  - a = (2·c -6)/6
  - a = (c - 5)/2

- **(2·c - 6)/6 = (c-5)/2 ⇔ c-5 = ⅔·c - 2 ⇔ ⅓ c = 3 ⇔ c = 9**

Notes

Notes

- **Your final implementation should be a one executable file per question**
  - Many source code files
- **Follow the execution instructions**
  - e.g. CLI arguments, arguments order
- **Allowed to use mentioned libraries**

Turnin

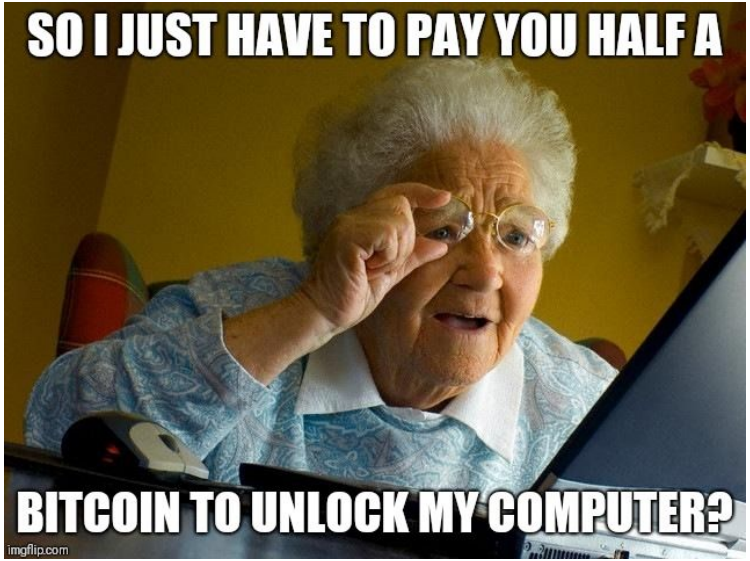
What to submit:

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


```
➢ turnin assignment_1@hy457 directory_name
```

Notes

- **This year assignment 1 is 25% of final grade**
- **Assignment 2 will be online after the Easter holidays**
- **40% Assignments:**
  - Assignment 1: 25%
  - Assignment 2: 15%
- **60% Final Exam**



Credit

Icons from FlatIcon, made by Freepik

Thank You!

papamano@csd.uoc.gr

Questions?