Introduction to Cybersecurity

Lecture #4: Reverse engineering I

Anton Semenkin

CTF sections

- Web
- Crypto
- Stegano
- Reverse
- Pwn
- Ppc
- Recon
- Misc
- Forensics

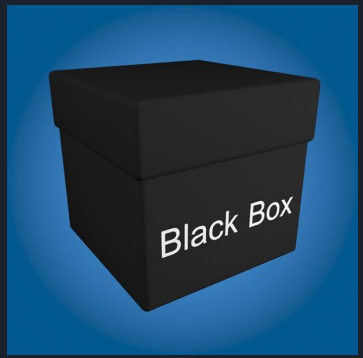
CTF sections

- Web
- Crypto
- Stegano
- Reverse
- Pwn
- Ppc
- Recon
- Misc
- Forensics

What is it?

- Forward engineering
  - Come up with idea
  - Choose tools
  - Create an architecture
  - Develop application

- Reverse engineering
  - You have a black box (desktop / mobile app)
  - Study how this box works



Why?

Why?

Make better

Find hidden functions

Get +400

Adapt for new hardware

Flex

Make your own

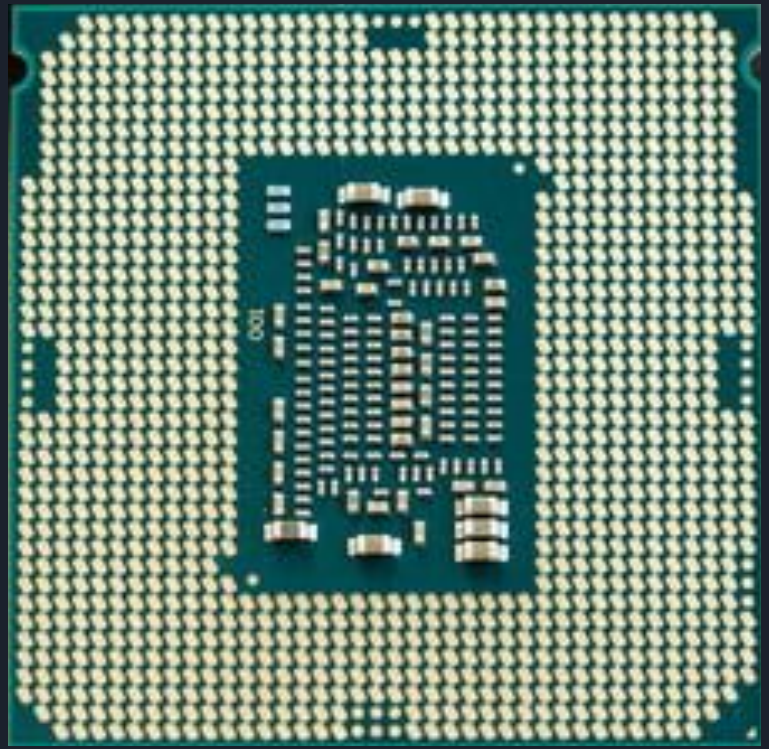
Find mistakes

Bugbounty

Harm (illegal)

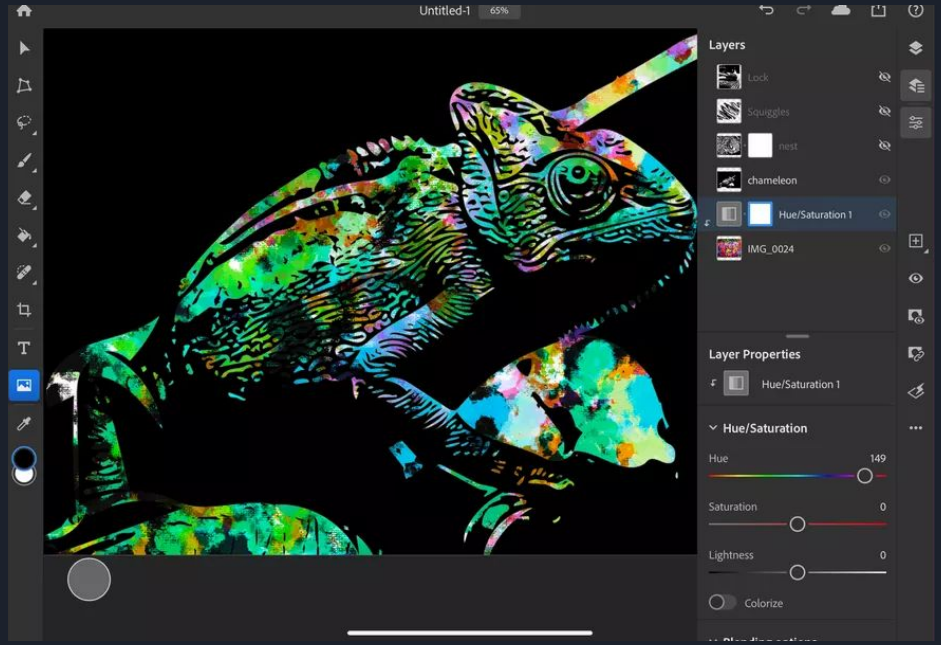
Hardware reverse

Hardware reverse

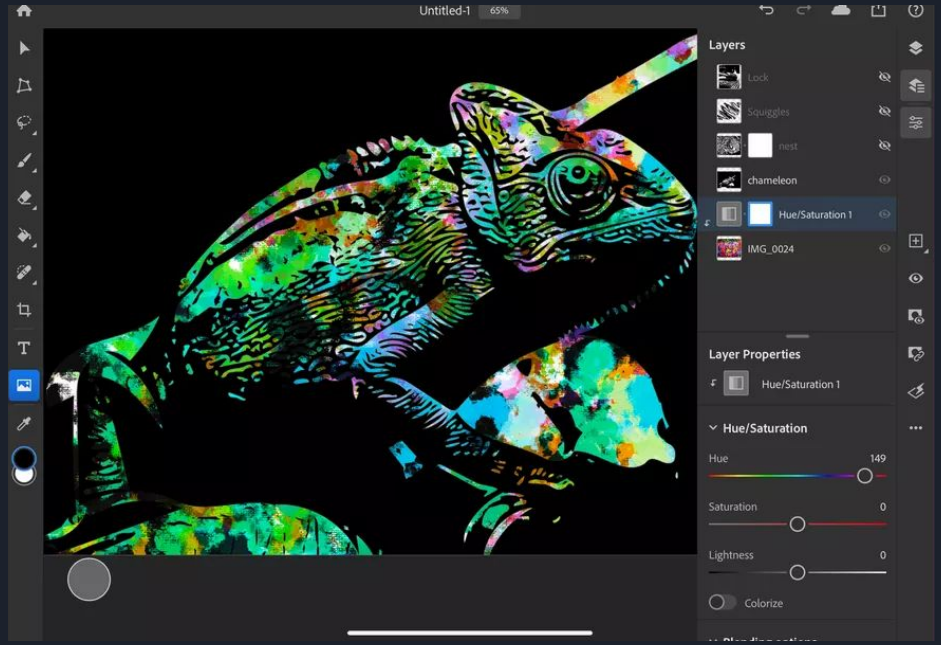


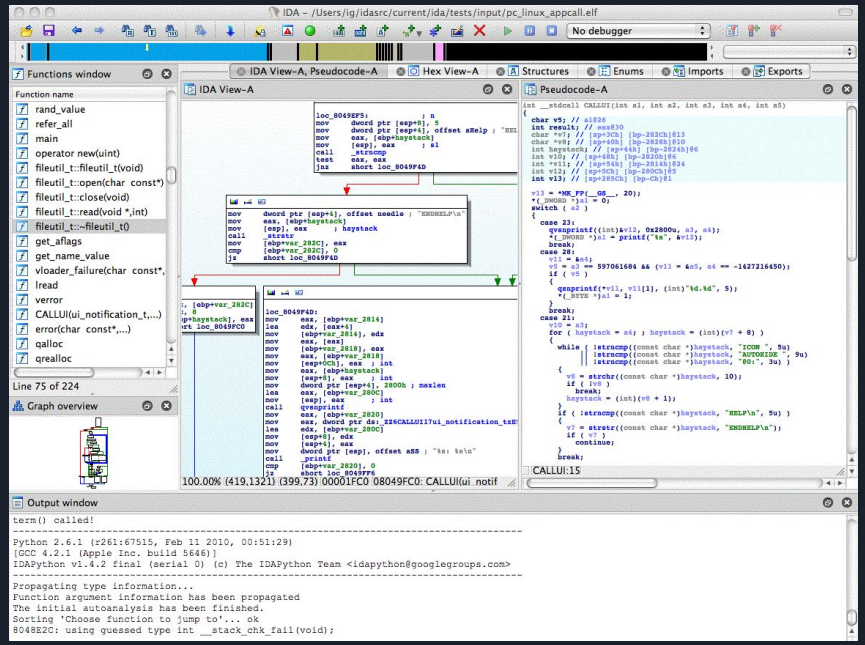
Hardware reverse

Software reverse



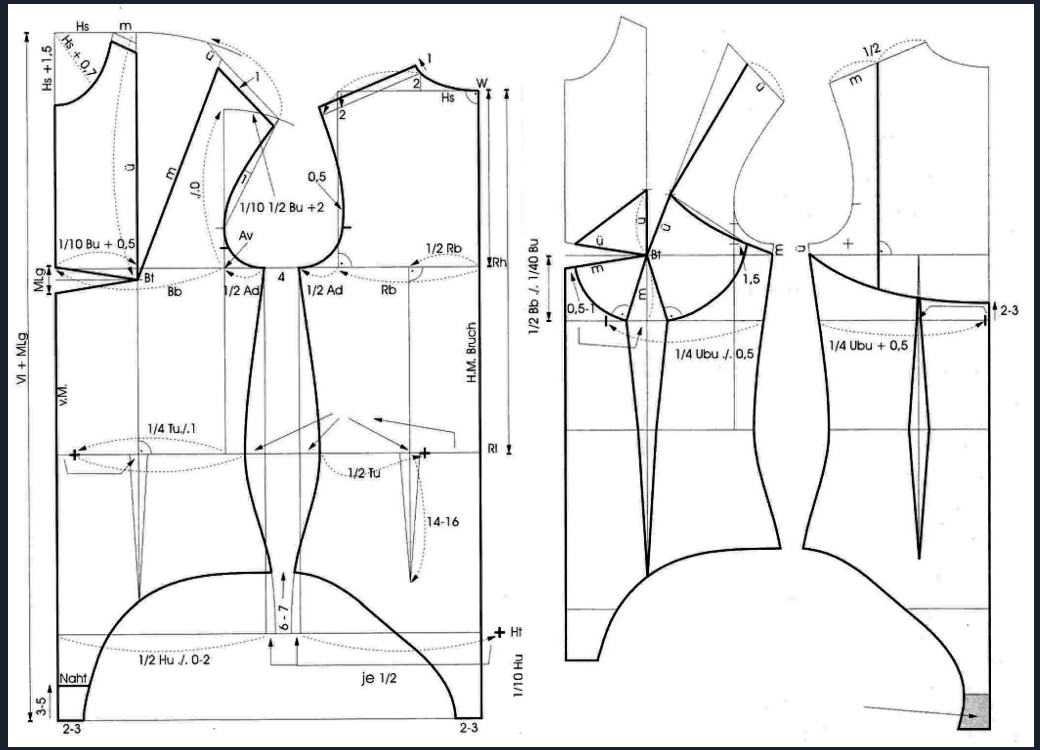
Software reverse





Clothes reverse

Clothes reverse



Clothes reverse

Time to play

https://overthewire.org/wargames/bandit/

Reverse is hard

What is the program?

- Code point of view
  - Operations
  - Custom functions
  - Library functions
  - System calls
- OS point of view
  - Executable and Linkable Format
  - Scripts (#!/usr/local/bin/my_brainfuck_interp, #!/bin/zsh)
  - Old formats (a.out, Common Object File Format)

ELF

- Other programs
- Dynamic libraries (*.so)
- Object files (*.o)

Executable file has only one entry point (*_start* or *main*), but the library has many (all exportable functions)

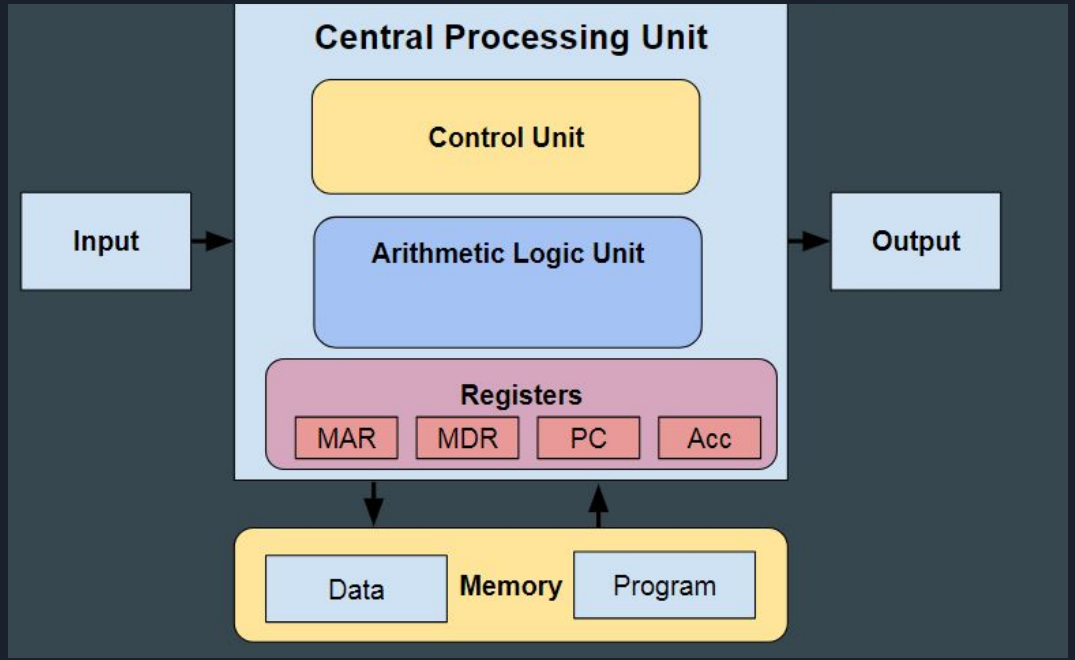
Demo

How computer works

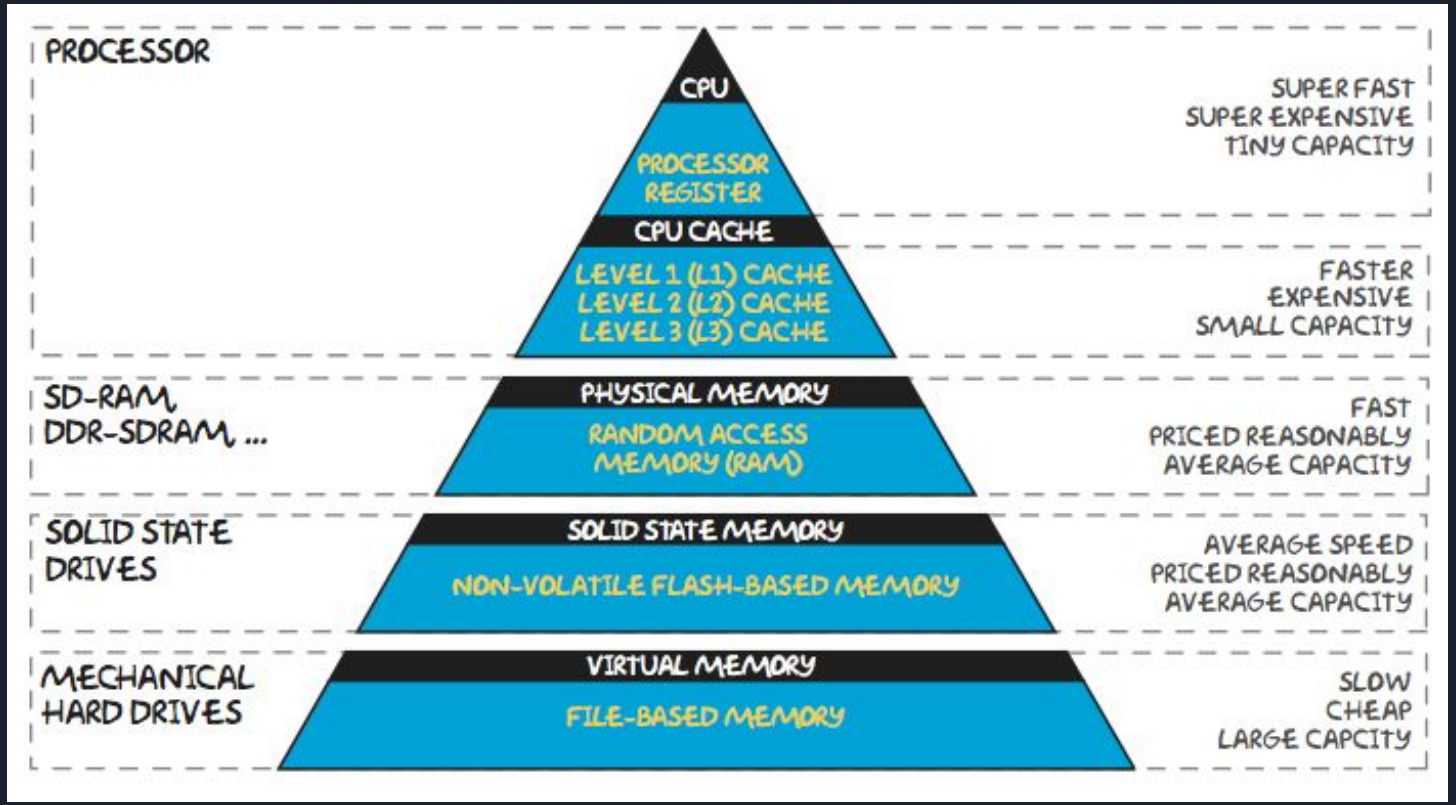
- Von Neumann architecture
- Memory hierarchy
- Process memory

Von Neumann architecture

- Processing unit
- Control unit
- Memory unit
- Input / output devices

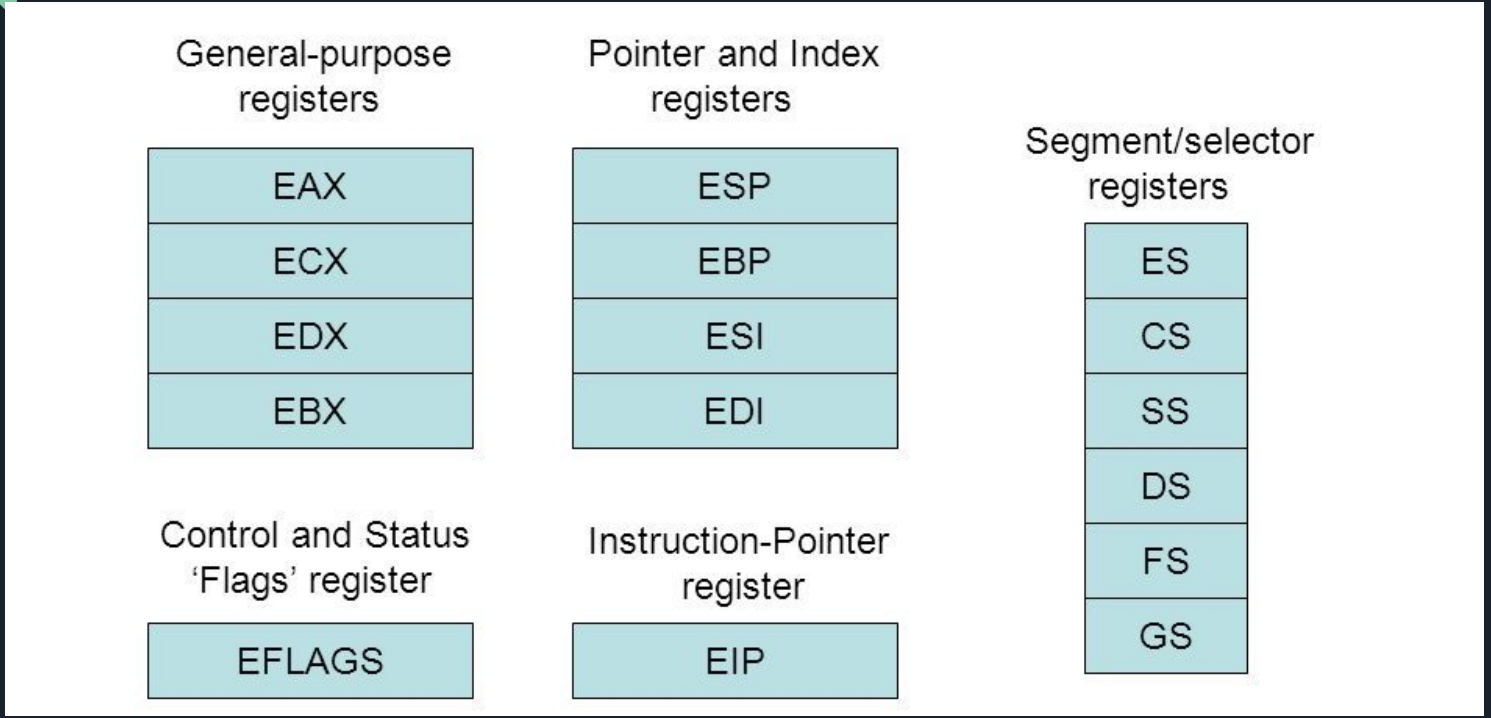


Memory hierarchy



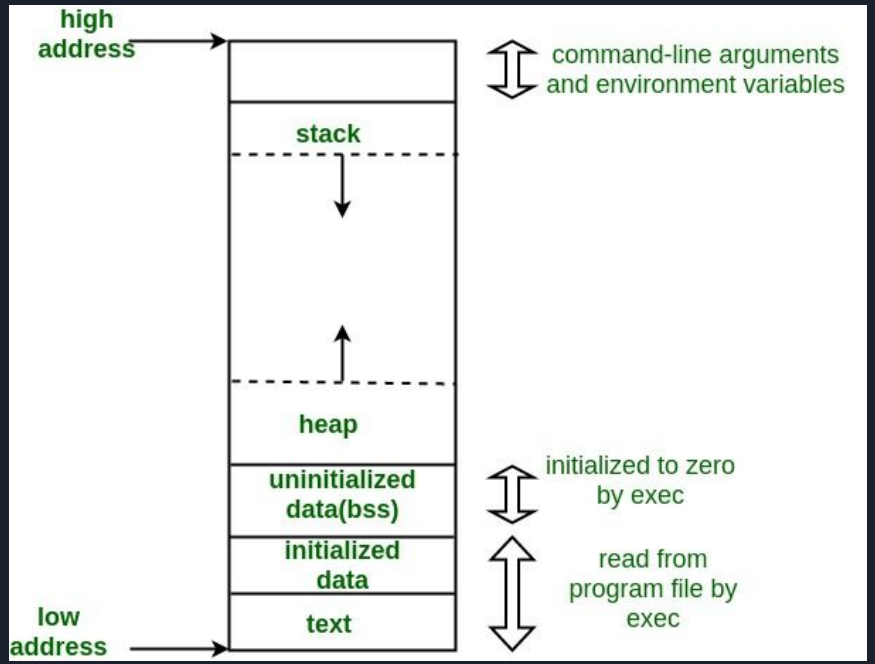
x64 VS x32?

x32 processor registers



Memory layout

- Static
  - .bss, .data
- Dynamic
  - heap
- Automatic
  - stack



https://www.geeksforgeeks.org/memory-layout-of-c-program/

Demo

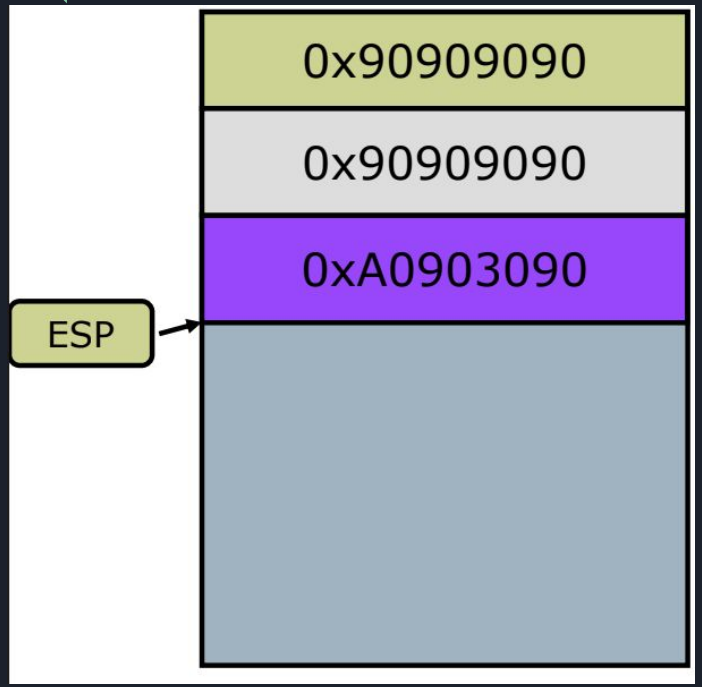
stack VS heap

Stack

- Last-in-first-out (LIFO) structure
- Contains temporary (local) function variables
- Managed and optimized by CPU (not by you)
- Functions clears stack after exit*

*For more information read about calling conventions

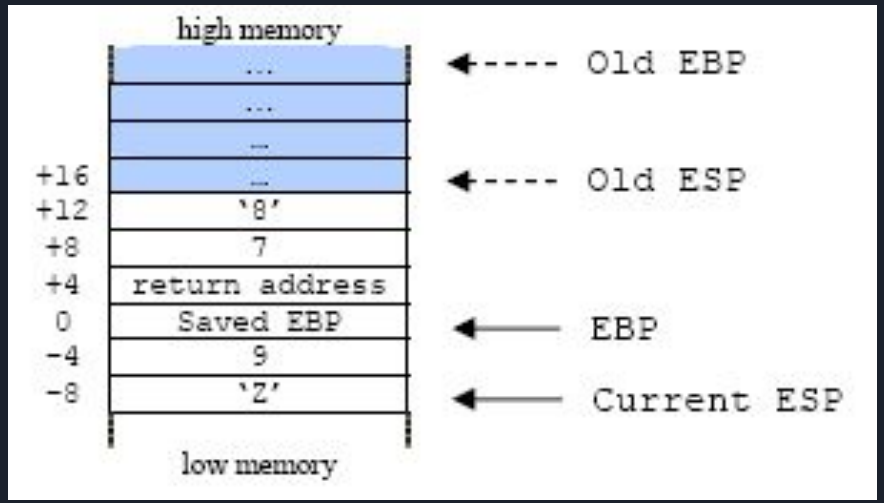
Stack: in-depth view



- Push - put element on the top of the stack
- Pop - take top value from stack
- ESP - pointer to the top of the stack

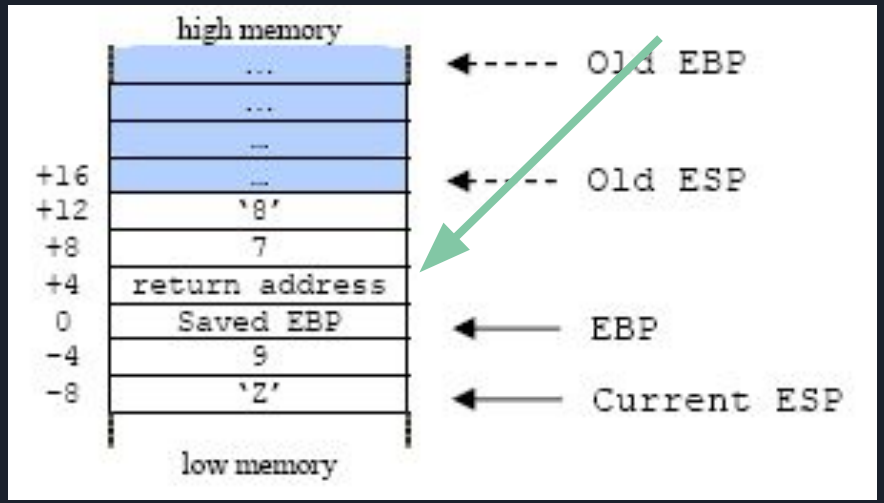
Stack: in-depth view

```c
#include <stdio.h>

int MyFunc(int parameter1, char parameter2) {

    int local1 = 9;

    char local2 = 'Z';

    return 0;
}


int main(int argc, char **argv) {

    MyFunc(7, '8');
    return 0;
}
```



Stack: in-depth view

```c
#include <stdio.h>

int MyFunc(int parameter1, char parameter2) {

    int local1 = 9;

    char local2 = 'Z';

    return 0;
}


int main(int argc, char **argv) {

    MyFunc(7, '8');
    return 0;
}
```

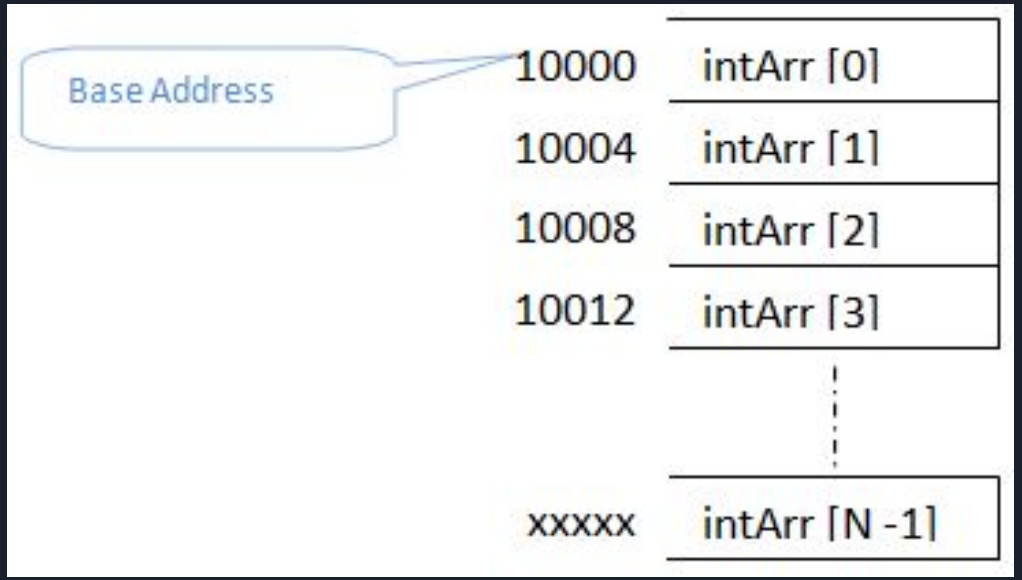


Demo

What’s wrong?

```
int func(char *user) {
    fprintf(stdout, user);
}
```

Arrays in stack



https://www.geeksforgeeks.org/buffer-overflow-attack-with-example/

Heap

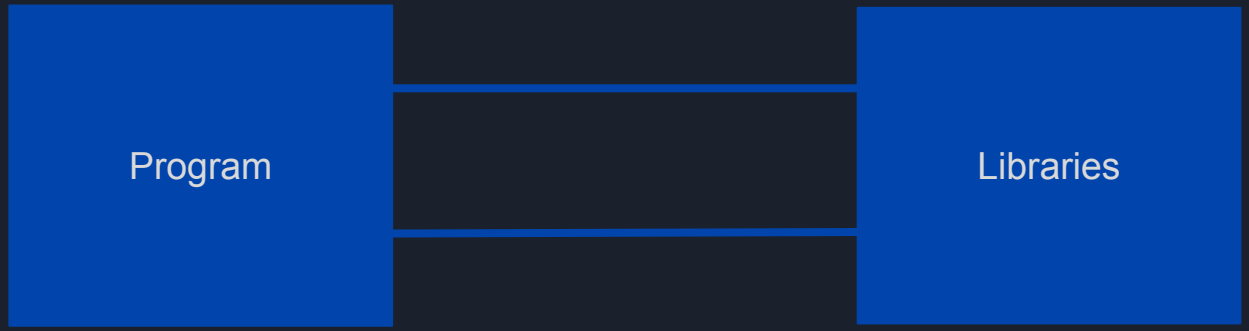
- Managed by you
- Global variables
- malloc(), calloc()
- YOU are responsible to free() the memory you allocated
- If you fail to free the space, you've got **memory leak**

For hardcore reversers:
https://www.contextis.com/en/resources/white-papers/glibc-adventures-the-forgotten-chunks

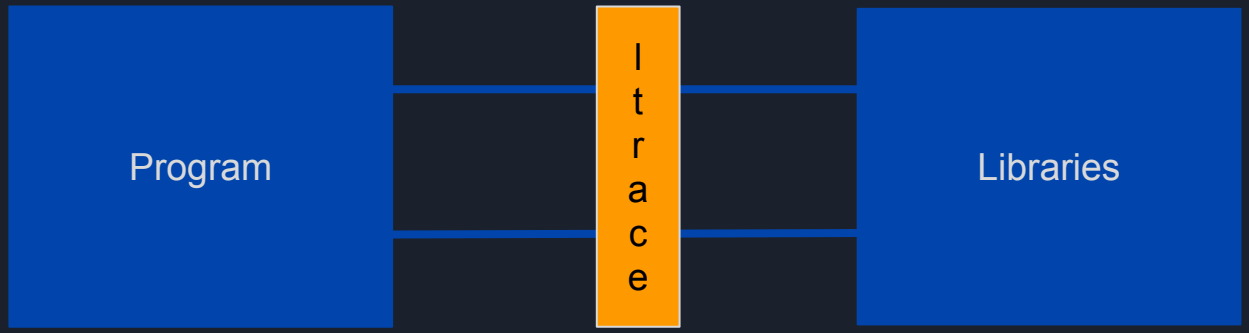
Reverse levels

- Source code analysis
- Tracing
- Static analysis
- Binary patching

Tracing: functions interception

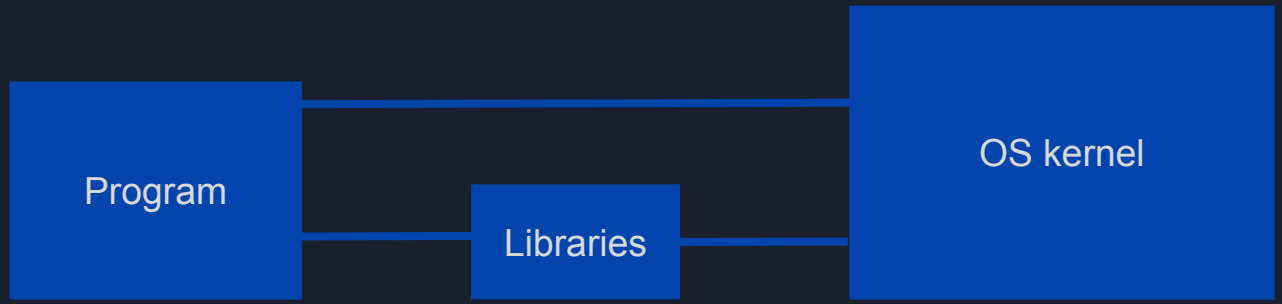


Tracing: functions interception

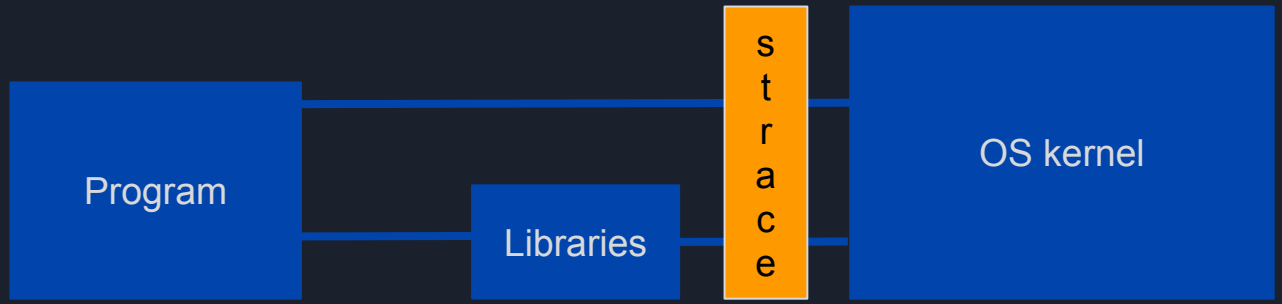


Demo

Tracing programs: syscalls interception



Tracing programs: syscalls interception



Demo