Learn to Hack

* but ethically

Check in!

Raise Hands!

- Casual event
- If anything is not working or you have any questions, please raise your hands!

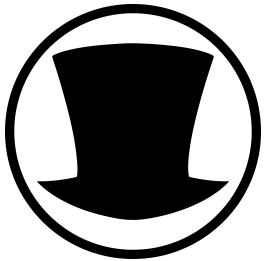
Event Outline

- Hacking/cybersecurity and why it's important
- Event sandbox
- Setting up sandbox
- Introduction to concepts (process, stack, stack frame, strcpy,memcpy)
- Introduction to **Exploit 1**
- Perform **Exploit 1**
- Introduction to **Exploit 2**
- Perform **Exploit 2**
- Introduction to **Exploit 3**
- Perform **Exploit 3**
- Conclusion

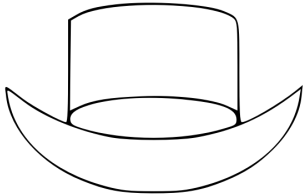
What is hacking?

- Exploiting a technical vulnerability for some motivation
- Not just bad guys
- Not all illegal
- $10.25 trillion in damage by 2025

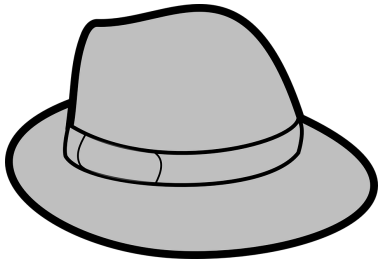
Hacking motivations (Hats)



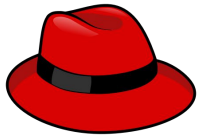
- Bad guys
- The stereotype
- Malicious/personal gain
- Cyber theft
- Ransomware

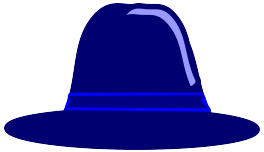


- Good guys
- Penetration tests
- Vulnerability assessments
- “Ethical Hacking

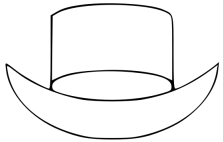


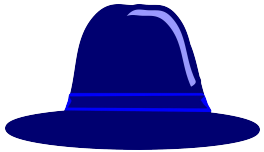
- Eh? Guys
- Search for vulnerabilities
- Request money for details/fix
- Notify company/community





Cybersecurity





- Securing a system to stop the bad guys!
- Incorporates **design** and **defenses**
- Detailed knowledge of computer systems
- “Think like a hacker”
- Exploit testbeds

Importance

- Dependence on computers -> gravity of attacks -> importance to defend
- Notable examples:
  - Mirai
  - Stuxnet
  - Colonial pipeline
  - Recent attacks on US hospitals and supply chain factories
- “Cyber warfare”
  - Russia…Anonymous

Today’s sandbox

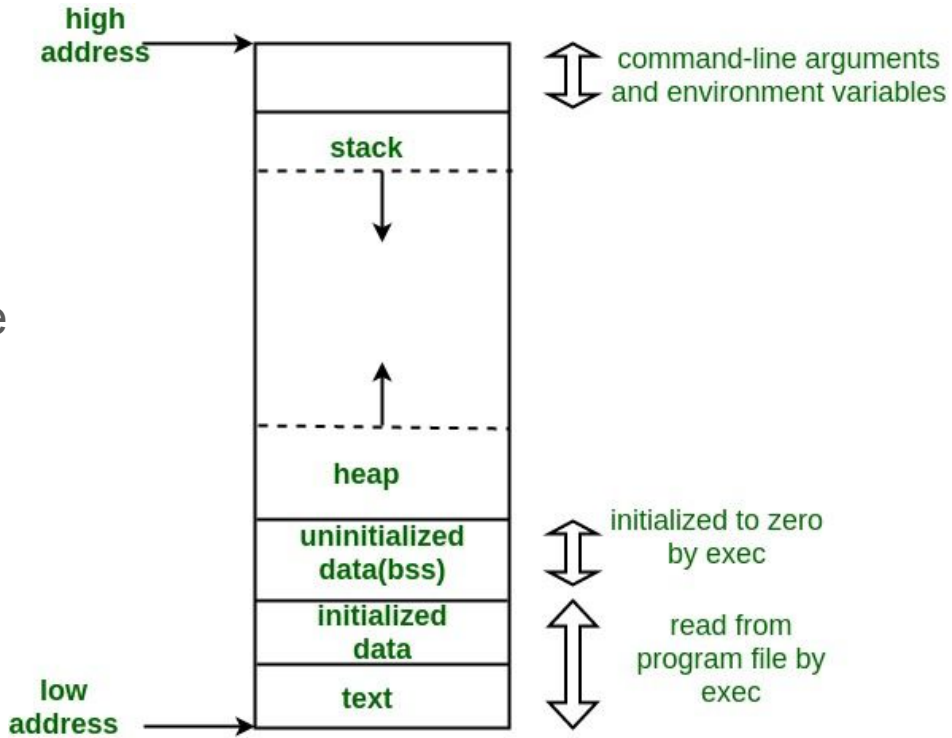
- Practicing on local machine UNSAFE
- Two levels of isolation/protection for this event
  1) **Amazon Machine Instances** (AMI)
     a) Virtual machine
     b) Completely remote
     c) 40 identical AMIs for this event
        i) All event software installed already!
  2) **Docker container** (in each AMI)
     a) Almost a VM (shares kernel and other system resources of host machine)

Setting up your sandbox

- Follow “Getting Set Up” instructions here to get your environment set up

What is a process?

- A running program
- Virtual memory
- Process virtual memory compone
  - Stack
  - Heap
  - Code
- Registers

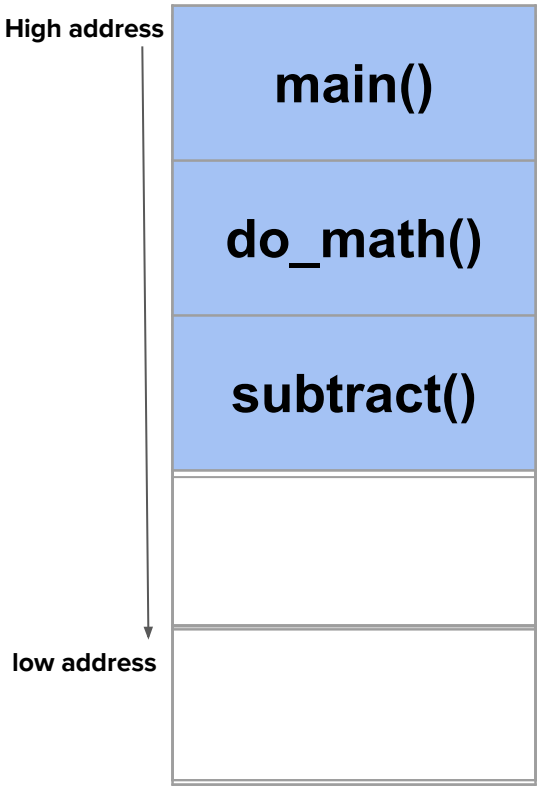


Stack Frame

- Memory allocated on stack for function when it is *called*
  - Popped off stack when function *returns*

Stack frame

```c
#include <stdio.h>

int add(int a, int b) {
    return a + b;
}

int subtract(int a, int b) {
    return a - b;
}

int do_math(int a, int b) {
    int n1 = add(a, b);
    int n2 = subtract(a, b);
    return n1*n2;
}

int main() {
    do_math(8, 2);
    return 0;
}
```



Simple buffer overflow On dummy program (DoS Attack)

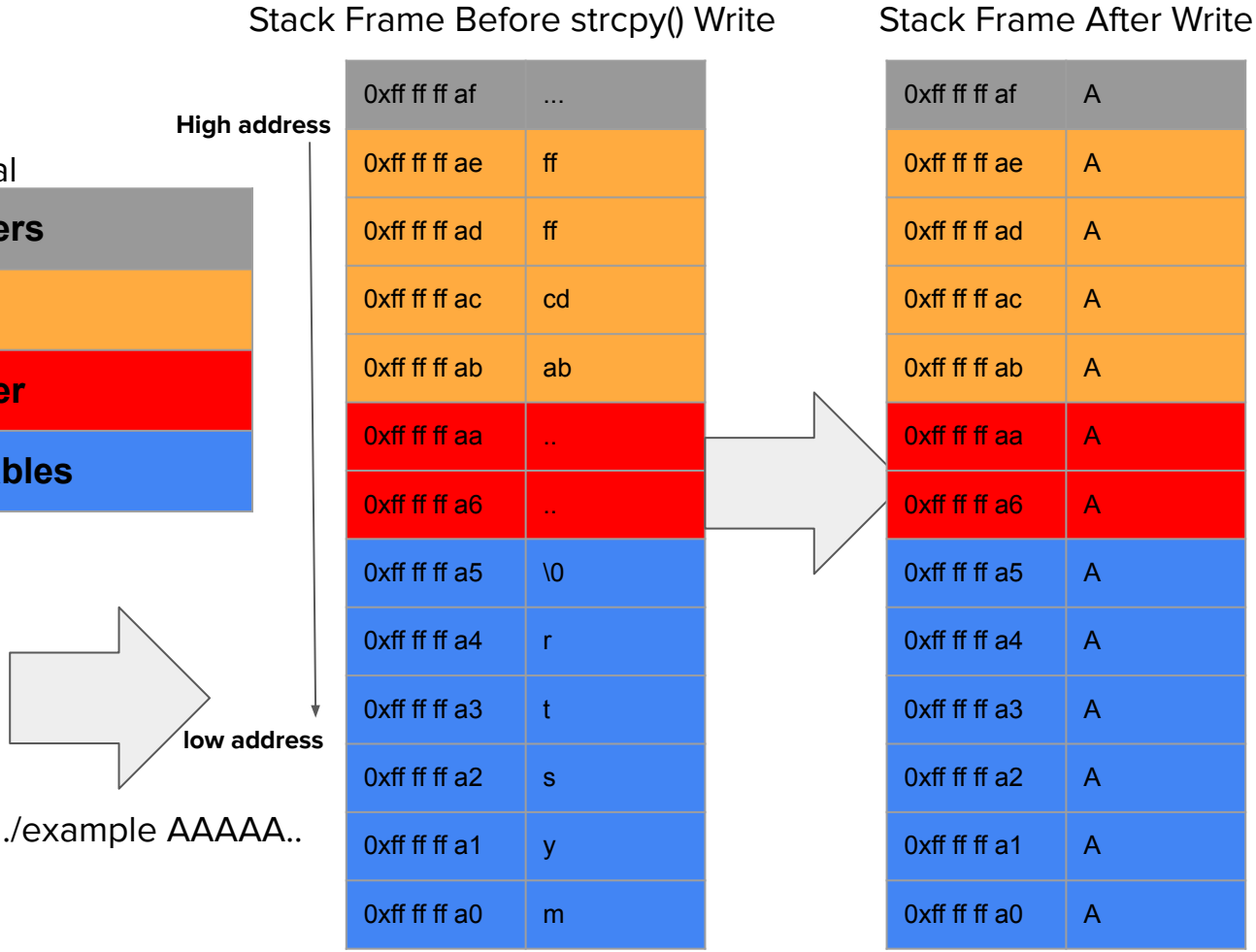
1. Stack Frame General

2.

```c
#include <stdio.h>
#include <string.h>

void main(int argc, char *argv[]) {


    char str[] = "mystr";


    strcpy(str, argv[1]);
}
```

./example AAAAA..

3.



Connman

- Network Management software (think DNS queries and responses)
  - Handles all of this
- Lightweight
- Common in Internet of Thing (IoT) devices

‘dig’ command

- **Domain information groper (dig)** - command line tool for Querying DNS servers
  - Receives a **DNS response**
    - Many fields, including a **name** field

Connman vulnerability (i.e. our vulnerability for today!)

- Stack-based buffer overflow vulnerability **CVE-2017-12865**
  - Connman handles dig requests
    - Stores **name** field of DNS response in `name` buffer of size **1024 bytes**
    - Uses memcpy for data write, so no bounds-checking!
    - DNS response with **name** field greater than 1024 bytes will overwrite adjacent memory!
- Attacker-controlled DNS Server can send malicious DNS responses
  - For this event, we will be making minor edits to `simple_dns.py`, our mock malicious DNS server
    - **It is already on your AMI !!**

Download `simple_dns.py` to have it locally!

Exploit 1: Simple Buffer Overflow (DoS) on Connman

- `dns_dos_payload(self, data)` (line 194 of `simple_dns.py`) function returns DNS response for queries to “dos.com”
  - Lines 195 through 201 represent appropriate DNS response fields
  - Line 203 is beginning of **name** field that will be stored in `name` buffer in Connman
    - Note its length!!!
- Let’s perform this exploit!

Exploit 2: Advanced Stack Smashing/Code Injection Attack on Connman

- Malicious code at memory address address 0x 12 34 56 78
  - Overwrite return address with this
  - Our computers reads **high to low**! (*little-endian* - least significant byte first)

Exploit 2: Advanced Stack Smashing/Code Injection Attack on Connman cont.

- Computers are dumb… they do whatever they're told
  - "Told" things with **assembly language instructions**, each with a corresponding Opcode (1 byte)
  - NOP = "No Operation", Opcode = "\x90"
    - Computer slides to next instruction
  - **Shellcode** - sequence of assembly language instructions to perform some task
    - I.e. spawning a **shell**
- Strategy:
  - Send: NOPs + shellcode + NOPs + (address of a NOP)

Exploit 2: Advanced Stack Smashing/Code I
Attack on Connman cont.

- Head over to the Instructions (Part IV)
  - and follow them!

Thanks for coming…