

# 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!

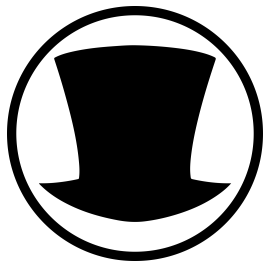
# Event Outline

- 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

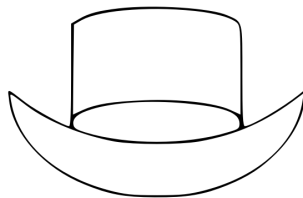
# What is hacking?

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

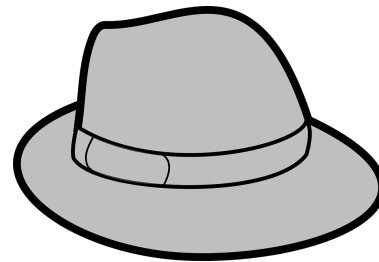
# Hacking motivations (Hats)



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



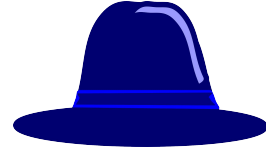
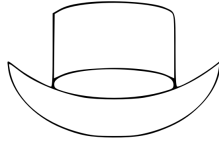
- Good guys
- Penetration tests
- Vulnerability assessments
- "Ethical Hacking"



- 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)

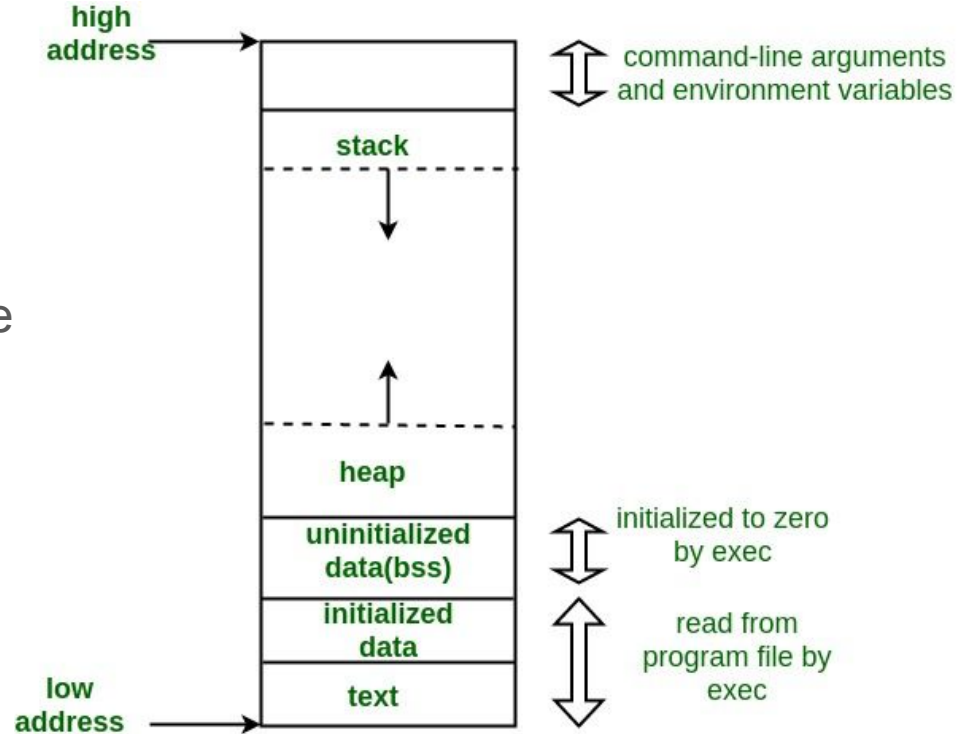


# Setting up your sandbox

- Follow “Getting Set Up” instructions [here](#) to get your environment set up

# What is a process?

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

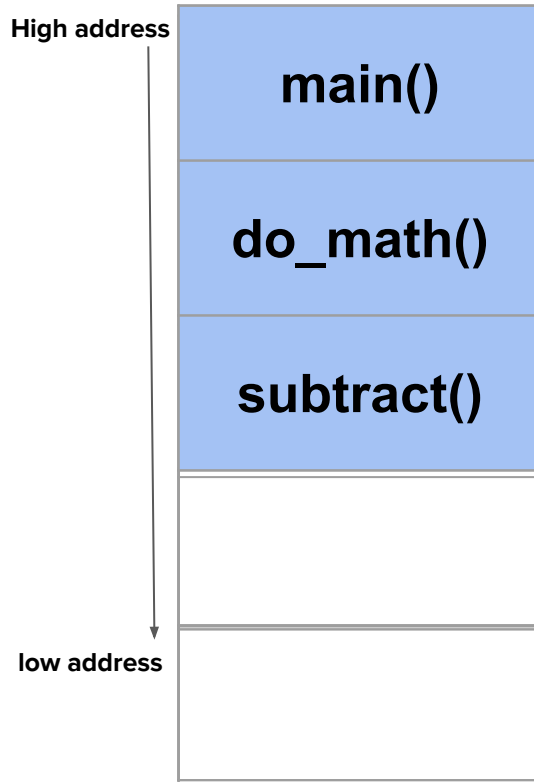


# Stack Frame

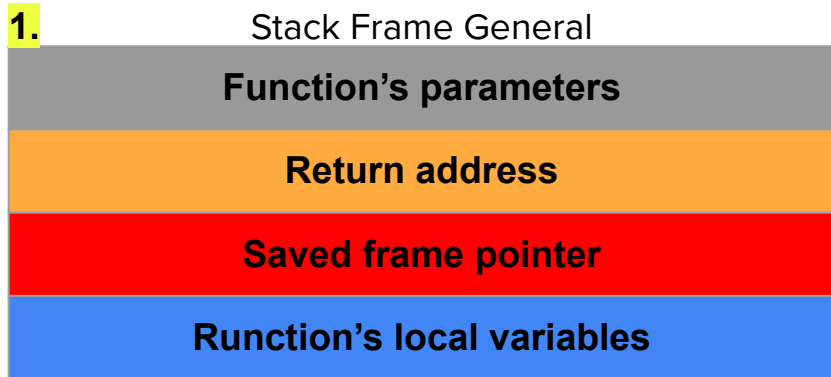
- Memory allocated on stack for function when it is *called*
  - Popped off stack when function *returns*

# Stack frame

```
1  #include <stdio.h>
2
3  int add(int a, int b) {
4  →   return a + b;
5  }
6
7  int subtract(int a, int b) {
8  →   return a - b;
9  }
10
11 int do_math(int a, int b) {
12 →   int n1 = add(a, b);
13 →   int n2 = subtract(a, b);
14 →   return n1*n2;
15 }
16
17 int main() { ←—————
18     do_math(8, 2); ←————
19 →   return 0;
20 }
```



# Simple buffer overflow On dummy program (DoS Attack)



2.

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void main(int argc, char *argv[]) {
5
6
7      char str[] = "mystr";
8
9
10     strcpy(str, argv[1]);
11 }
```

low address

./example AAAAAA..

3.

Stack Frame Before strcpy() Write

0xff ff ff af	...
0xff ff ff ae	ff
0xff ff ff ad	ff
0xff ff ff ac	cd
0xff ff ff ab	ab
0xff ff ff aa	..
0xff ff ff a6	..
0xff ff ff a5	\0
0xff ff ff a4	r
0xff ff ff a3	t
0xff ff ff a2	s
0xff ff ff a1	y
0xff ff ff a0	m

Stack Frame After Write

0xff ff ff af	A
0xff ff ff ae	A
0xff ff ff ad	A
0xff ff ff ac	A
0xff ff ff ab	A
0xff ff ff aa	A
0xff ff ff a6	A
0xff ff ff a5	A
0xff ff ff a4	A
0xff ff ff a3	A
0xff ff ff a2	A
0xff ff ff a1	A
0xff ff ff a0	A

# 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 !!**

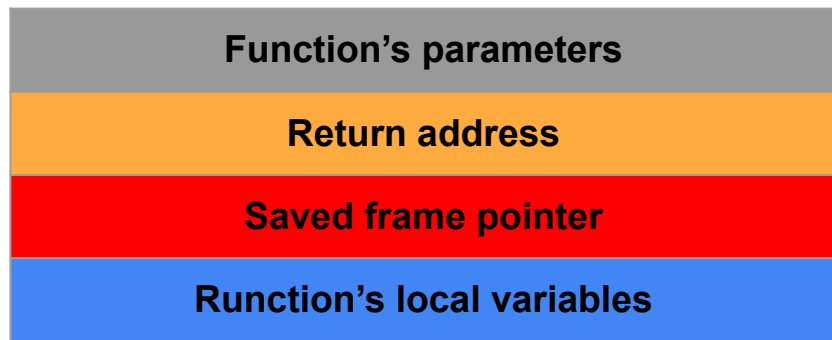
Download [simple\\_dns.py](#)  
to have it locally!



## 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)

A diagram illustrating memory addresses in little-endian format. A vertical arrow on the left points downwards, labeled "High address" at the top and "low address" at the bottom. To the right of the arrow is a table with two columns: the first column contains hexadecimal addresses, and the second column contains the corresponding values.

0xff ff ff af	A
0xff ff ff ae	\x12
0xff ff ff ad	\x34
0xff ff ff ac	\x56
0xff ff ff ab	\x78
0xff ff ff aa	A
0xff ff ff a6	A
0xff ff ff a5	A
0xff ff ff a4	A
0xff ff ff a3	A
0xff ff ff a2	A
0xff ff ff a1	A
0xff ff ff a0	A

## 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 Injection Attack on Connman cont.

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

High address

low address

0xff ff ff bc	...
0xff ff ff bb	\xa0
0xff ff ff ba	\xff
0xff ff ff b9	\xff
0xff ff ff b8	\xff
0xff ff ff b7	A
0xff ff ff b6	A
0xff ff ff b5	shellcode
0xff ff ff a4	...
0xff ff ff a3	shellcode
0xff ff ff a2	\x90
0xff ff ff a1	\x90
0xff ff ff a0	\x90

Thanks for coming...