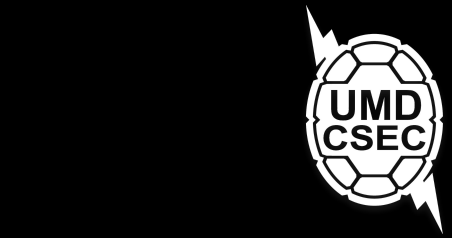


CMSC388U

Binary Exploitation

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

Announcements

- How was break?
- HW 7 out soon :)
  - Due by next lecture
- How was HW6 & the midterm?
- Office hour appointments & piazza!
- Mid-semester check-in?

Any HW/Midterm Q's?

- If you disassemble a C binary, the result is…
- Which of these are valid bash syntax?
- Out of the following CVE IDs, which has the highest impact by CVSS?
  - Everyone who got this wrong said the same answer → Piazza?

- What is the IP address for http://supersecure.store? (Mention how you found it)
- What is the importance of a hash when dealing with forensics?
- Write an nmap scan to do OS detection on http://fake.domain where it does OS detection, scans ports 80, 22, and 443, and saves the output to a file.
- How is file un-deletion possible?
- What metadata does this image have? Is there perhaps a flag somewhere?

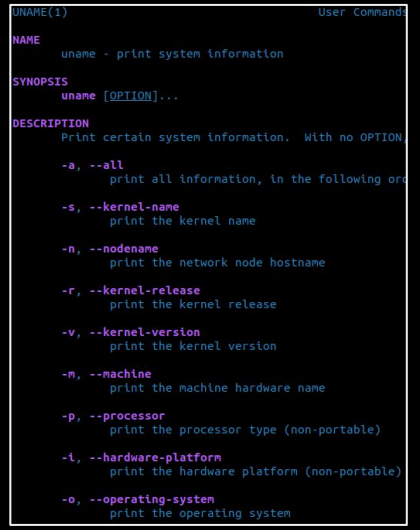
*Fun*damentals

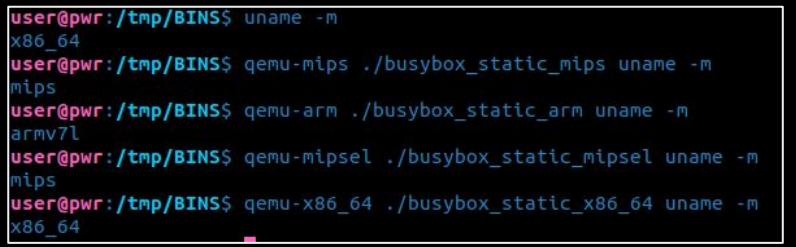
Basic x86 Assembly Overview	Static and Dynamic Linking
Stack & Heap	Binary Structure

ASMbly

- You (hopefully) saw MIPS assembly in CMSC216
  - x86 and ARM are also very interesting

- x86
  - Primarily larger devices
    - laptops, desktops, most personal computers
  - 1500+ *documented* instructions
- ARM
  - Primarily smaller devices
    - Phones, tablets, cameras, routers, etc.
- MIPS
  - Mainly embedded devices
    - security cameras, routers, etc.





x86 Overview

General Purpose Registers:

- **rax**: “accumulator” - for math operations
- **rbx:** “base” address for memory calculations
- **rcx:** “counter” register for repeated operations
- **rdx**: auxiliary “data” register for math ops
- **rsi:** “source index” for load/copy instructions
- **rdi**: “destination index” for store/copy instructions
- **rbp:** “base pointer” for stack frames
- **rsp:** “stack pointer” for operations on the stack
- **r8-r15:** new 64 bit registers only in x86-64

```
user@pwr:~$ cat helloworld.c
#include <stdio.h>

int main() {
  char user_input[50];

  printf("Hello, world!\nWhat is your name? ");
  scanf("%s", user_input);

  printf("Hi, %s!\n", user_input);
}

user@pwr:~$ gcc helloworld.c -o helloworld
user@pwr:~$ ./helloworld
Hello, world!
What is your name? Vanya
Hi, Vanya!
user@pwr:~$ ./helloworld
Hello, world!
What is your name? Alden
Hi, Alden!
```

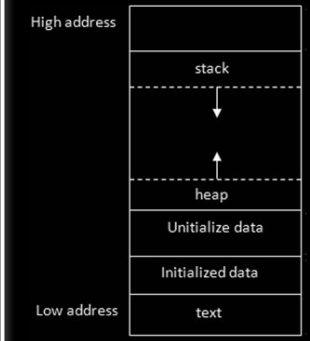
```
user@pwr:~$ gdb ./helloworld
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04) 9.2
Copyright (C) 2020 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./helloworld...
(No debugging symbols found in ./helloworld)
(gdb) set disassembly-flavor intel
(gdb) disass main
Dump of assembler code for function main:
   0x0000000000001189 <+0>:     endbr64
   0x000000000000118d <+4>:     push   rbp
   0x000000000000118e <+5>:     mov    rbp,rsp
   0x0000000000001191 <+8>:     sub    rsp,0x40
   0x0000000000001195 <+12>:    mov    rax,QWORD PTR fs:0x28
   0x000000000000119e <+21>:    mov    QWORD PTR [rbp-0x8],rax
   0x00000000000011a2 <+25>:    xor    eax,eax
   0x00000000000011a4 <+27>:    lea    rdi,[rip+0xe5d]        # 0x2008
   0x00000000000011ab <+34>:    mov    eax,0x0
   0x00000000000011b0 <+39>:    call   0x1080 <printf@plt>
   0x00000000000011b5 <+44>:    lea    rax,[rbp-0x40]
   0x00000000000011b9 <+48>:    mov    rsi,rax
   0x00000000000011bc <+51>:    lea    rdi,[rip+0xe67]        # 0x202a
   0x00000000000011c3 <+58>:    mov    eax,0x0
   0x00000000000011c8 <+63>:    call   0x1090 <__isoc99_scanf@plt>
   0x00000000000011cd <+68>:    lea    rax,[rbp-0x40]
   0x00000000000011d1 <+72>:    mov    rsi,rax
   0x00000000000011d4 <+75>:    lea    rdi,[rip+0xe52]        # 0x202d
   0x00000000000011db <+82>:    mov    eax,0x0
   0x00000000000011e0 <+87>:    call   0x1080 <printf@plt>
   0x00000000000011e5 <+92>:    mov    eax,0x0
   0x00000000000011ea <+97>:    mov    rdx,QWORD PTR [rbp-0x8]
   0x00000000000011ee <+101>:   xor    rdx,QWORD PTR fs:0x28
   0x00000000000011f7 <+110>:   je     0x11fe <main+117>
   0x00000000000011f9 <+112>:   call   0x1070 <__stack_chk_fail@plt>
   0x00000000000011fe <+117>:   leave
   0x00000000000011ff <+118>:   ret
End of assembler dump.
```

Stack & Heap

Stack (system defined memory)

TLDR: when you run a program on a linux/*NIX machine, the system allocates memory for the program to run inside of - the stack

- **The stack grows down**
- **Typically 8MB in size (which is *usually* fine)**
- **Stack pointer (rsp) keeps track of top element of the stack**



Heap (prog. defined memory)

TLDR: think calling malloc, free, calloc etc.

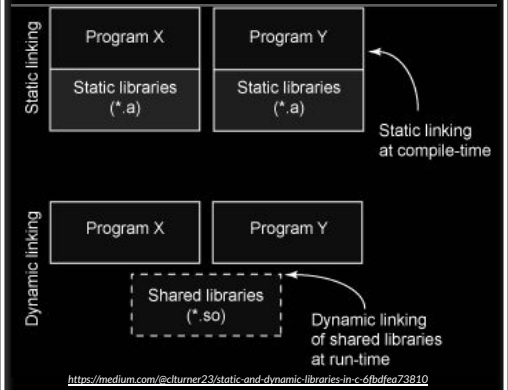
The heap grows up

No predefined size

Linking

Static

- During the linking phase of compilation, if binary is statically linked, all library modules are copied into the executable
- You know the addresses of all those library functions **before** execution



https://medium.com/@clturner23/static-and-dynamic-libraries-in-c-6fbdfea73810

Dynamic

- The names of the external libraries are placed in the final executable - but they are referred to by multiple programs (why they're sometimes called *shared libraries)*
- Addresses of libraries is unknown before runtime

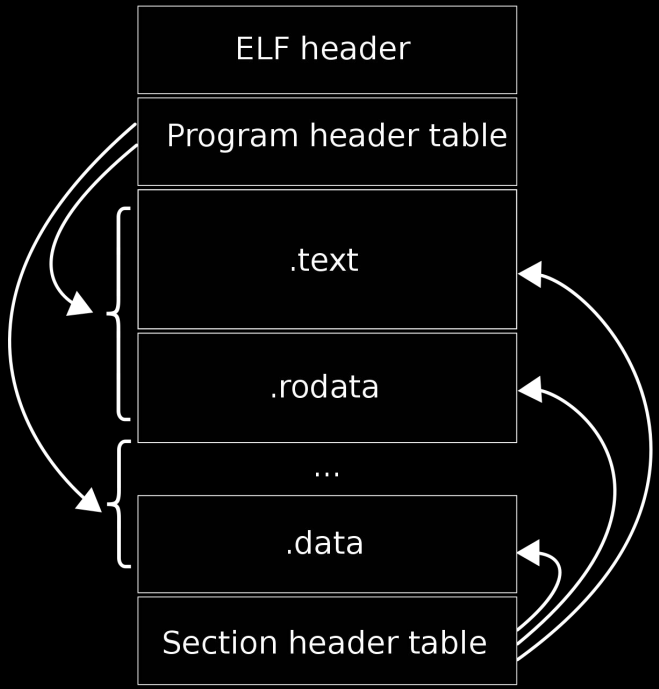
Binary Structure

Executable and Linkable Format (ELF)

- **TL;DR:** Basic *nix executables
- **ELF header:** 32 bytes, identifies the format of the file, determines **endianness**
- Table info
- Sections (data, functions, etc)
- Table 2!

Portable Executables (PE)

- **TL;DR:** Basic Windows executables
- **PE header:** DOS-stub, PE signature, COFF file header (COFF file header + optional header)
- More complicated than ELF (don't worry about it for this class)

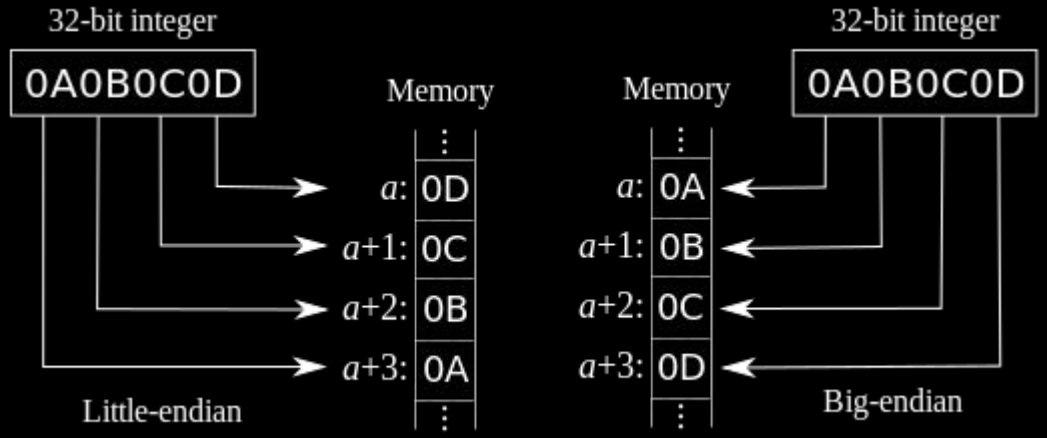


64 bit

	0	1	2	3	4	5	6	7	
	Signature 0x5A4D	DOS Header							
		(0x3C) Pointer to PE Header							
	DOS STUB								
0x0000	Signature 0x50450000				Machine		#NumberOfSections		COFF Header
0x0008	TimeDateStamp				PointerToSymbolTable (deprecated)				
0x0010	# NumberOfSymbolTable (deprecated)				SizeOfOptionalHeader		Characteristics		
0x0018	Magic		MajorLinker Version	MinorLinker Version	SizeOfCode (sum of all sections)				Standad COFF Fields
0x0020	SizeOfInitializedData				SizeOfUninitializedData				
0x0028	AddressOfEntryPoint (RVA)				BaseOfCode (RVA)				
0x0030	BaseOfData (RVA)				ImageBase				
0x0038	SectionAlignment				FileAlignment				
0x0040	MajorOperating SystemVersion		MinorOperating SystemVersion		MajorImage Version		MinorImage Version		Windows Specific Fields
0x0048	MajorSubsystem Version		MinorSubsystem Version		Win32VersionValue (zeros filled)				
0x0050	SizeOfImage				SizeOfHeaders				
0x0058	CheckSum (images doesn't checked)				Subsystem		DllCharacteristics		
0x0060	SizeOfStackReserve				SizeOfStackCommit				
0x0068	SizeOfHeapReserve				SizeOfHeapCommit				
0x0070	LoaderFlags (zeros filled)				# NumberOfRvaAndSizes				Optional Header
	ExportTable (RVA)				SizeOfExportTable				
	ImportTable (RVA)				SizeOfImportTable				
	ResourceTable (RVA)				SizeOfResourceTable				
	ExceptionTable (RVA)				SizeOfExceptionTable				
	CertificateTable (RVA)				SizeOfCertificateTable				
	BaseRelocationTable (RVA)				SizeOfBaseRelocationTable				
	Debug (RVA)				SizeOfDebug				
	GlobalPtr (RVA)				00	00	00	00	Data Directories
	TLSTable (RVA)				SizeOfTLSTable				
	LoadConfigTable (RVA)				SizeOfLoadConfigTable				
	BoundImport (RVA)				SizeOfBoundImport				
	ImportAddressTable (RVA)				SizeOfImportAddressTable				
	DelayImportDescriptor (RVA)				SizeOfDelayImportDescriptor				
	CLRRuntimeHeader (RVA)				SizeOfCLRRuntimeHeader				
	00	00	00	00	00	00	00	00	
	Name								Section Table
	VirtualSize				VirtualAddress (RVA)				
	SizeOfRawData				PointerToRawData				
	PointerToRelocations				PointerToLinenumbers				
	NumberOfRelocations		NumberOfLinenumbers		Characteristics				

Endiness

- **Big-endian:** stores the most significant byte of a word (small collection of bytes) at the smallest memory address and the least significant byte at the largest
- **Little-endian:** Stores the least-significant byte at the smallest address
- **Bi-endianness:** Some crazy architectures (e.g ARM) use a setup that allows for switching between BE and LE
- https://gchq.github.io/CyberChef/



What is Binary Exploitation?

- Binary Exploitation
  - Taking advantage of errors/vulnerabilities in compiled code (binaries) with the purpose of realizing unintended behavior (RCE, memory leaks, etc)

- Goals:
  - Code execution
  - Leaking sensitive info
  - Privilege escalation
  - Keygen

Command Injection, The Sequel

- Unsanitized user input
  - A.k.a. untrusted/unchecked/etc.
- We talked about exec()/subprocess/etc. in other languages
  - With C, we care about `system()` !
- Recap: C programs can execute other programs with `system()`
  - You did this in CMSC216
- `system(arg_with_user_input);` // Can lead to code execution

```
user@pwr:~$ ./inject
What do you want to ls? /tmp/example
Your input: ls -t /tmp/example
a  b  c  d  e  f  file1  file2  g
user@pwr:~$ ./inject
What do you want to ls? /tmp/example;whoami
Your input: ls -t /tmp/example;whoami
a  b  c  d  e  f  file1  file2  g
user
user@pwr:~$ cat inject.c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int custom_ls(char *);

int main() {
  char cmd[50] ="ls -t ";
  char user_input[50];

  printf("What do you want to ls? ");
  scanf("%s", user_input);

  strcat(cmd, user_input);
  printf("Your input: %s\n", cmd);

  system(cmd); /* 🔥 This is fine 🔥 */
}
```

Format Strings

- Remember `printf()` and `scanf()` syntax?
  - Generally can format like:
    - `printf("My var is: %s!\n", my_variable);`
  - But also…
    - `printf("This is a string!\n");`
  - What if…
    - `printf(my_variable); //?`

- What if `my_variable` contains format strings?
  - %s for strings
  - %x for hexadecimal
  - %d for integers

```
user@pwr:~$ ./format
What do you want me to say? test
test
user@pwr:~$ ./format
What do you want me to say? %x%x%x
a00
user@pwr:~$
```

```
ser@pwr:~$ cat format.c
nclude <stdio.h>

t main() {
 volatile char user_input[50];
 volatile char secret[25] = "You can't print me!";

 printf("What do you want me to say? ");
 scanf("%s", user_input);

 printf(user_input); /* 🔥 This is fine 🔥 */
 printf("\n");


ser@pwr:~$ gcc ./format.c -o format -w -fno-stack-protector
ser@pwr:~$ ./format
at do you want me to say? TEST
EST
ser@pwr:~$ ./format
at do you want me to say? %8$llx%7$llx%6$llx%5$llx%4$llx%3$llx
1656d20746e6972702074276e616320756f597ca0
ser@pwr:~$ echo "21656d20746e6972702074276e616320756f597ca0" | xxd -r -p # hex to ascii
m tnirp t'nac uoY|◆user@pwr:~$
ser@pwr:~$ # wrong endianess but :)
```

Buffer Overflow (BoF)

- Stack-based
  - Statically allocated buffers (e.g. `char buf[15];`) are limited by the allocation
    - What happens when we go past this limit?
      - Depending on the function, you can read/write into “random” data
        - Or… other values on the stack >:)

- Heap based
  - Heap BoF Guide

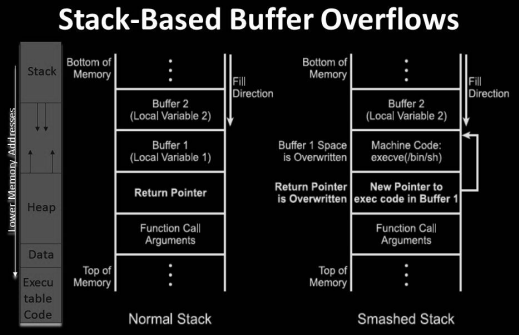


EXHIBIT 10.2 A normal stack and a stack with a buffer overflow.

https://www.mkdynamics.net/current_projects/computer_security/Basic_Linux_exploits.html

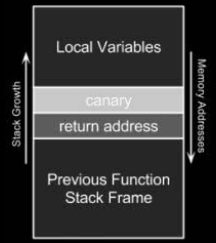
Anti-RE

- Stripped binaries
  - Smaller, but no debug symbols
- Code obfuscation
  - Making code harder to read and understand
- VM detection
- Debugger Detection

https://hackaday.com/2020/07/10/mmm-obfuscated-shell-donuts/

```
const detectVM = () => {
  // TODOfuture for now no VM detection on Mac OS X
  if (process.platform === 'darwin') {
    return;
  }
```

Binary Protections



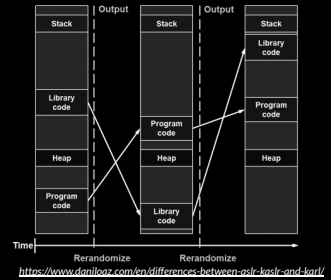
https://security.cs.pub.ro/summer-school/wiki/session/11

- Stack canaries
  - Within the stack, trigger a failsafe if overwritten (e.g. by BoF)

- Stack memory protections
  - NX (No eXecute)
  - RELRO (RELocation Read Only)

- ASLR (Address Space Layout Randomization)
  - Randomly positions the base address of an executable and the positions of binary components…

```
user@pwr:~$ gcc ./format.c -o format
./format.c: In function 'main':
./format.c:13:10: warning: passing argument 1 of 'printf' discards 'volatile' qualifier from pointe
   13 |   printf(user_input); /* 🔥 This is fine 🔥 */
      |          ^~~~~~~~~~
In file included from ./format.c:1:
/usr/include/stdio.h:332:43: note: expected 'const char * restrict' but argument is of type 'volati
  332 | extern int printf (const char *__restrict __format, ...);
      |                    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~~~~~~~
./format.c:13:10: warning: format not a string literal and no format arguments [-Wformat-security]
   13 |   printf(user_input); /* 🔥 This is fine 🔥 */
      |          ^~~~~~~~~~
user@pwr:~$ gcc ./format.c -o format -w
user@pwr:~$ ./format
What do you want me to say? %x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x%x
*** stack smashing detected ***: terminated
Aborted (core dumped)
user@pwr:~$ python3 -c "print('%x'*29)" | ./format
*** stack smashing detected ***: terminated
Aborted (core dumped)
```



https://www.daniloqz.com/en/differences-between-aslr-kaslr-and-kaiser/

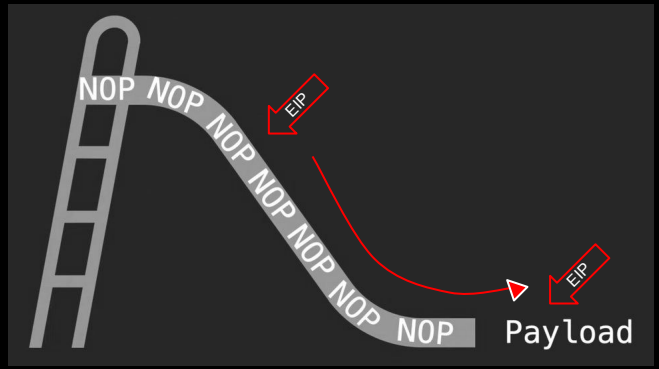
checksec

- Is a program that allows you to check which binary protections are enabled on a particular binary
- https://github.com/slimm609/checksec.sh
- `checksec --file={binary files}`

```
alden@ubuntu:~/programming/ctfs/picoctf2021/binary-gauntlet-1$ checksec --file=gauntlet1
RELRO           STACK CANARY      NX            PIE           RPATH      RUNPATH      Symbols       FORTIFY Fortified      Fortifiable  FILE
Partial RELRO   No canary found   NX disabled   No PIE        No RPATH   No RUNPATH   67 Symbols    No      0              3            gauntlet1
```

More Advanced Bugs

- NOP slide
  - If you don't know where program execution (the instruction pointer) will go
  - Sliiiiide down to the payload with "No-OPeration"s
- GOT
  - "Global Offset Table" in ELFs
  - Stores function locations
  - Can overwrite entries to point to attacker code
- Ret2libc & ROP
  - ROP guide
  - ret2libc guide



Resources & Tools

- Tools
  - GDB (& pwngdb)
  - Pwntools
  - Ghidra

- Resources
  - ghidraninja/stacksmashing on YT
  - Liveoverflow on YT
  - ctf101.org
  - exploit.education

For next time...

- Homework 7 will be released asap :)
  - We recommend starting a little early

- If you are confused, don't be afraid to:
  - Schedule office hours (over Piazza)
  - Make public or private Piazza posts