CSci 4271W
Development of Secure Software Systems
Day 5: Memory Corruption 1
(or, why to avoid C and C++)

Stephen McCamant (he/him)
University of Minnesota, Computer Science & Engineering

Based in large part on slides originally by Prof. Nick Hopper
Licensed under Creative Commons Attribution-ShareAlike 4.0

To follow along

From a Linux terminal:

```
git clone https://github.umn.edu/badlycoded/memcorr.git
```

(various example code from the next 3 lectures)

Memory corruption

- Memory corruption bugs happen when a program writes data to an area of memory that it shouldn't.
- Type-safe languages such as Java, OCaml, Rust, Swift, and Go can prevent most such bugs.
- In C or C++ it is easy to write a program that corrupts memory:

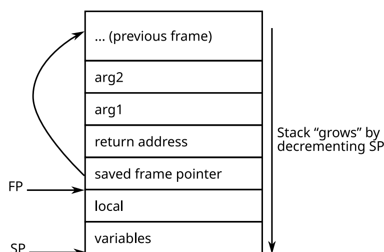
```
int x = 0x0011;
char buff[4];
buff[4] = 'a'
```

So what?

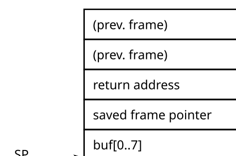
Recall, each function call has a stack frame that stores:

- Function arguments
- Local variables
- Any "callee-saved" register values
- The memory address of the next instruction from the calling function, i.e. the return address.

Typical stack frame



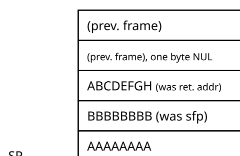
Smashing the stack




```
void func(char *str) {
  char buf[8];
  strcpy(buf, str);
  do_something(buf);
}
```

Calling `func("AAAAAAAABBBBBBBBABCDEFGH")`

Smashing the stack, cont'd

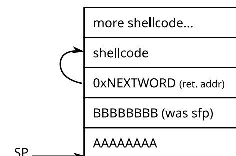



```
void func(char *str) {
  char buf[8];
  strcpy(buf, str);
  do_something(buf);
}
```

Calling `func("AAAAAAAABBBBBBBBABCDEFGH")`
After `strcpy`

Func will try to return to instruction at address ABCDEFGH

For fun and profit




```
void func(char *str) {
  char buf[8];
  strcpy(buf, str);
  do_something(buf);
}
```

Classic attack: replace ABCDEFGHI with address inside overwrite on stack, insert compiled shell code that calls `exec("bin/sh")`

`https://archives.phrack.org/issues/49/14.txt`

Memory corruption: big picture

- Modern OSes and compilers block such “easy” attacks, but more clever variants can still work.
- Today: different ways memory can become corrupted.
- Next lecture: some attacks based on memory corruption.

Outline

- Memory corruption intro
- Announcements break
- Memory corruption vectors
- Low-level code example

Homework 1

- Now open for submission on Gradescope (linked from Canvas)
- Due tonight by 11:59pm
- May do in groups of up to 3 students
- Be careful of the following on Gradescpe:
  - Include the names of your other group members
  - Provide the right range of pages for each answer

Outline

- Memory corruption intro
- Announcements break
- Memory corruption vectors
- Low-level code example

Memory corruption 1: overflow

Writing past the end of a stack buffer:

```
char buf[8];
strcpy(buf,"xxxxxxxxxxx");
```

Writing past the end of a heap buffer:

```
char *p = malloc(8);
char *q = "xxxxxxxxxxxxx";
while (*p++ = *q++) ;
```

Lots of functions will do this “for” you: `strcpy`, `gets`, `strcat`, `memcpy`, `scanf`, `sprintf`...

Memory corruption 2: temporal

Manipulating memory allocation functions, e.g. use-after-free:

```
char *p = malloc(sizeof(long));
strcpy(p, "hello");
free(p);
long *x = malloc(sizeof(long));
*x = 17;
printf("%8s\n", *p);
```

Memory corruption 3: integers

Pointer arithmetic and integer overflows:

```
uintptr_t p1 = UINTPTR_MAX;
char *p2 = malloc(32);
*(p2 + p1) = 'A';
```

Often caused by string-to-integer conversions:

```
int x = strtol(untrusted_input_string, NULL, 10);
return a[x];
```

Memory corruption 4: format strings

```
printf("May the %dth be %s\n", 4, "with you");
```

Format String

- Format strings are little programs.
- `printf` and friends step through the format string, writing output to a buffer.
- When the interpreter finds a % directive, it looks at the next word on the stack for the argument.
- So what happens if we call `printf("%p");` ?

Memory corruption 4: format strings (2)

```
printf("May the %dth be %s\n", 4, "with you");
```

Format String

- Format strings are little programs.
- An interesting option is positional arguments: `%i$w.pC` uses argument at position i instead of the next argument. (w is "width", p is "precision", both optional)

```
printf("%2$x\n",0x17,0x42);
```

Memory corruption 4: format strings (3)

```
printf("May the %dth be %s\n", 4, "with you");
```

Format String

- Format strings are little programs.
- `%n` is an interesting conversion: it treats its argument as a pointer to `int`, and stores the number of bytes output to the buffer so far:

```
int c = 42;
char s[128];
sprintf(s,"%127x%n",0,&c);
printf("%x\n",c);
```

Memory corruption 4: format strings (4)

Putting things together:

- The attacker knows there is a pointer on the stack to variable v, wants to set v to value z
- Find the right position (in the stack) i
- Make a format string that prints z characters
- Store (via `%n`) the number of characters to "argument" i

```
printf("%zx%i$n");
```

Outline

Memory corruption intro

Announcements break

Memory corruption vectors

Low-level code example