Buffer Overflows

Joseph Hallett

bristol.ac.uk

What is it?

What happens when you declare array?

- You get a region of memory

Pointers are used to address arrays

- Very easy to fall off the end of the region!

Have been known about since the dawn of computers, but earliest tutorial on how to exploit them in *Phrack magazine*

http://phrack.org/issues/49/14.html

How do functions work?

```
example1.c:
------------------------------------------------------------------------------
void function(int a, int b, int c) {
   char buffer1[5];
   char buffer2[10];
}

void main() {
  function(1,2,3);
}
------------------------------------------------------------------------------
bottom of                                                            top of
memory                                                               memory
           buffer2       buffer1   sfp   ret   a     b     c
<------   [            ][        ][    ][    ][    ][    ][    ]

top of                                                            bottom of
stack                                                                 stack
```

What about something like…

```
example2.c
------------------------------------------------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
------------------------------------------------------------------------
```

```
example2.c
------------------------------------------------------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
------------------------------------------------------------------------------
bottom of                                                             top of
memory                                                                memory
                  buffer                sfp   ret   *str
<------          [                ][    ][    ][    ]

top of                                                             bottom of
stack                                                                  stack
```

```
example2.c
------------------------------------------------------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
------------------------------------------------------------------------------
bottom of                                                                top of
memory                                                                   memory
                  buffer            sfp    ret    *str
<------          [AAAAAAAAAAAAAAAA][    ][    ][    ]

top of                                                                bottom of
stack                                                                     stack
```

bristol.ac.uk

```
example2.c
-----------------------------------------------------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
-----------------------------------------------------------------------------
bottom of                                                             top of
memory                                                                memory
                  buffer             sfp    ret    *str
<------          [AAAAAAAAAAAAAAAAAA][AAAA][AAAA][AAAA]AAAAAAAAAAAAAAAAAAAAAAAA

top of                                                             bottom of
stack                                                                  stack
```

bristol.ac.uk

```
example2.c
------------------------------------------------------------------------------
void function(char *str) {
   char buffer[16];

   strcpy(buffer,str);
}

void main() {
  char large_string[256];
  int i;

  for( i = 0; i < 255; i++)
    large_string[i] = 'A';

  function(large_string);
}
------------------------------------------------------------------------------
bottom of                                                                  top of
memory                                                                     memory
                  buffer             sfp    ret    *str
<------          [AAAAAAAAAAAAAAAA][AAAA][AAAA][AAAA]AAAAAAAAAAAAAAAAAAAAA

top of                                                                  bottom of
stack                                                                       stack
```

bristol.ac.uk

Where shall we return?

Being able to overwrite stack data is bad…

But overwriting return addresses gives us arbitrary code execution…

Normally will just cause an access validation (non-executable memory)

…Or a bad instruction (something isn't valid machine code/aligned)

But *sometimes* you can take over the program…

bristol.ac.uk

Shellcode

Classic way of doing this is with buffer shellcode

- This rarely works now… (W^X memory breaks it)
- …but in the labs we'll let you turn off these protections

Modern way is with *Return Oriented Programming* (ROP)

- We'll cover this later!

Some tricks to make it a bit easier…

- Alphabetic shellcode
- NOP-sleds

Shellcode

Classic way of doing this is with buffer shellcode

- This rarely works now… (W^X memory breaks it)
- …but in the labs we'll let you turn off these protections

Modern way is with *Return Oriented Programming* (ROP)

- We'll cover this later!

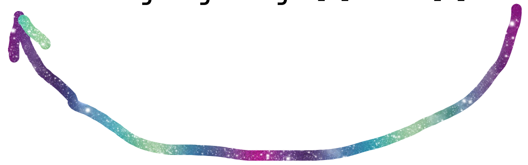
Some tricks to make it a bit easier…

- Alphabetic shellcode
- NOP-sleds

Shellcode

```
bottom of                                                              top of
memory                                                                 memory
                   buffer            sfp   ret   *str
<------           [?????NKsjaljvJKjv][AAAA][1234][AAAA]AAAAAAAAAAAAAAAAAAAAAA

top of                                                              bottom of
stack                                                                   stack
```



bristol.ac.uk

How do we stop this?

- Stack canaries spot if buffers have been overrun!
- W^X makes shellcode harder (but not impossible)
- Use bounded data structures not the old C ones
- Use the bounded memory functions (`strncpy`)
- Use a modern compiler toolchain and turn on the security features
  - `-D_FORTIFY_SOURCE=2 -fstack-protector-all -fsanitize=address…`

- Avoid C?
- (or at least take care…)

bristol.ac.uk