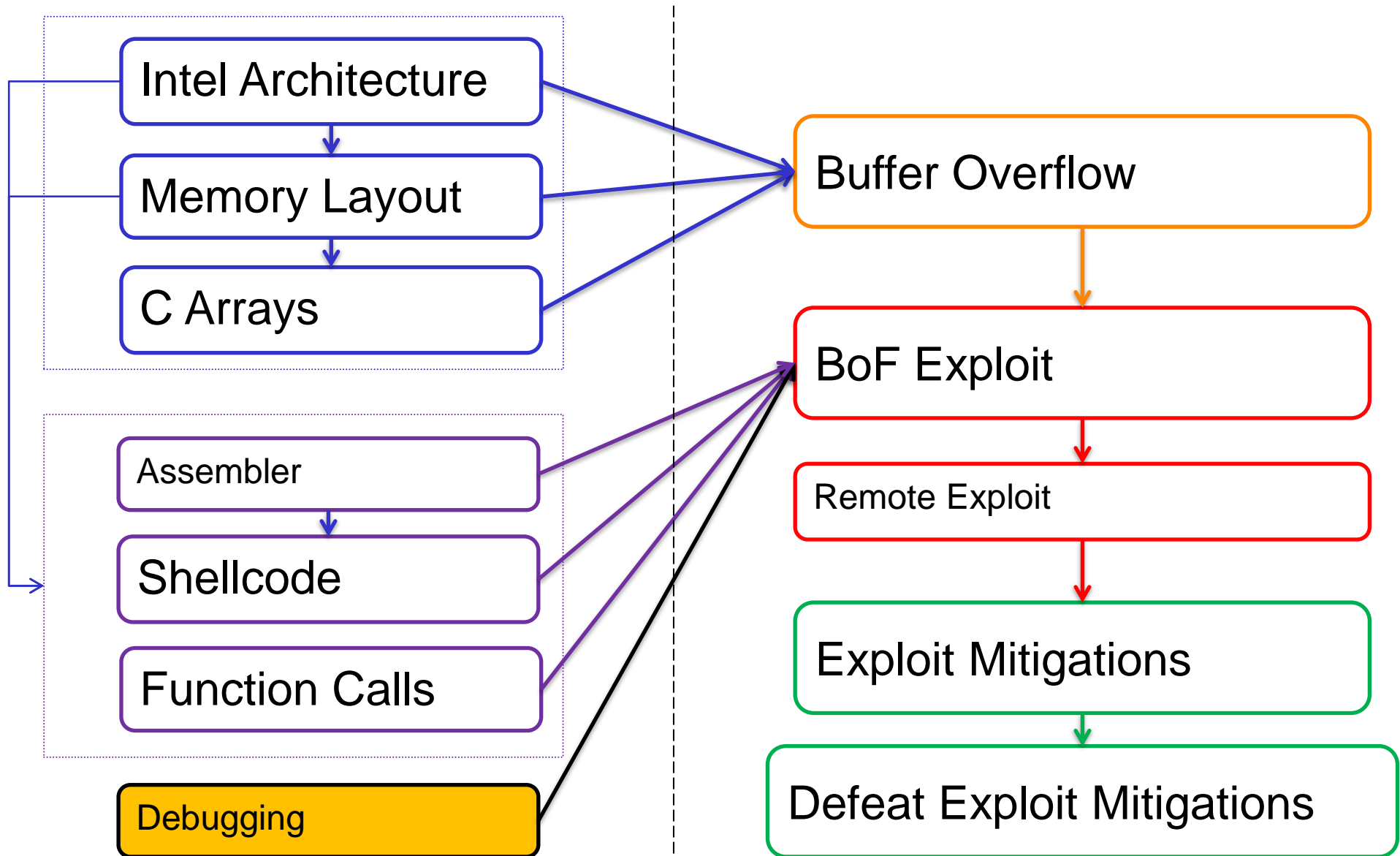

Debugging

Content



Debugging

Inspect state of a program while its running

Debugging

Start GDB:

```
$ gdb <filename>
```

Load a file while being in gdb:

```
(gdb) file <filename>
```

Start the program:

```
(gdb) run
```

Debugging

Inspecting code:

Where am i?

```
(gdb) where
```

Disassemble a function:

```
(gdb) disas main
```

```
Dump of assembler code for function main:
```

```
0x0000000000400b64 <+0>:    push    %rbp
0x0000000000400b65 <+1>:    mov     %rsp,%rbp
0x0000000000400b68 <+4>:    sub     $0x150,%rsp
0x0000000000400b6f <+11>:   mov     %edi,-0x144(%rbp)
0x0000000000400b75 <+17>:   mov     %rsi,-0x150(%rbp)
```

Debugging

Setting a breakpoint:

```
(gdb) break *0x00000000000400be3  
Breakpoint 1 at 0x400be3
```

Info about set breakpoints:

```
(gdb) info breakpoints
```

Num	Type	Disp	Enb	Address	What
1	breakpoint	keep	y	0x00000000000400be3	<main+127>

Delete a breakpoint:

```
(gdb) delete 1
```

Debugging

Continue execution:

(gdb) **continue**

Single step:

(gdb) **step**

Debugging

Reaching a breakpoint:

```
(gdb) run test test
```

```
Starting program: /home/hacker/bfh/challenge1 test test
```

```
Breakpoint 1, 0x0000000000400834 in main (argc=3,  
argv=0x7fffffffefa28) at challenge1.c:47
```

```
47      handleData(argv[1], argv[2]);
```

Backtrace:

```
(gdb) backtrace
```

```
#0  0x0000000000400834 in main (argc=3, argv=0x7fffffffefa28) at challenge1.c:47
```


Debugging

Inspecting registers:

```
(gdb) info register
```

rax	0x7fffffffecae	140737488350382
rbx	0x0 0	
rcx	0x0 0	
rdx	0x7fffffffecb3	140737488350387
rsi	0x7fffffffecb3	140737488350387
...		

Debugging

Inspecting memory:

```
(gdb) x/32x 0x7fffffff940
```

```
0x7fffffff940:  0x00000000    0x00000000    0xf781bb45    0x00007fff
0x7fffffff950:  0x00000000    0x00000000    0xffffea28    0x00007fff
0x7fffffff960:  0x00000000    0x00000003    0x004007e0    0x00000000
0x7fffffff970:  0x00000000    0x00000000    0xa2dfa5c8    0x1175d69a
```

```
(gdb) x/8b 0x7fffffff940
```

```
0x7fffffff940:  0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
```

```
(gdb) x/8g $rsp-8
```

```
0x7fffffff928:  0x0000000000400640    0x00007fffffff9ea28
0x7fffffff938:  0x0000000300000000    0x0000000000000000
0x7fffffff948:  0x00007ffff781bb45    0x0000000000000000
0x7fffffff958:  0x00007ffff9ea28     0x0000000300000000
```

x/<count><format><unit>

Debugging

x/<count><format><unit>

Format:

- ✦ x: Hexadecimal
- ✦ d: Decimal
- ✦ i: instructions
- ✦ s: string
- ✦ c: character

Unit:

- ✦ b: bytes
- ✦ w: Words (4 bytes, 32 bit)
- ✦ g: Giant words (8 bytes, 64 bit)

Debugging

If compiled with debugging symbols (-ggdb)

```
(gdb) list
```

Local variables

```
(gdb) info locals
```

Debugging

```
(gdb) info file
```

```
Symbols from "/home/hacker/bfh/day2/challenge3".
```

```
Local exec file:
```

```
  `/home/hacker/bfh/day2/challenge3', file type elf64-x86-64.
```

```
Entry point: 0x400640
```

```
0x0000000000400200 - 0x000000000040021c is .interp
```

```
0x000000000040021c - 0x000000000040023c is .note.ABI-tag
```

```
0x000000000040023c - 0x0000000000400260 is .note.gnu.build-id
```

```
0x0000000000400260 - 0x000000000040027c is .gnu.hash
```

```
0x0000000000400280 - 0x00000000004003a0 is .dynsym
```

```
0x00000000004003a0 - 0x0000000000400455 is .dynstr
```

```
0x0000000000400456 - 0x000000000040046e is .gnu.version
```

```
0x0000000000400470 - 0x00000000004004b0 is .gnu.version_r
```

```
0x00000000004004b0 - 0x00000000004004c8 is .rela.dyn
```

```
0x00000000004004c8 - 0x0000000000400588 is .rela.plt
```

```
0x0000000000400588 - 0x00000000004005a2 is .init
```

```
...
```

Debugging

Important settings:

Attach to a running process, and follow forks:

```
(gdb) set follow-fork-mode child
```

This will be important for the remote exploit challenge

Debugging

Attach to already existing processes:

```
(gdb) attach <pid>
```

Debugging

Allow creation of core files:

```
$ ulimit -c unlimited
```

Use a core file:

```
$ gdb <binary> <corefile>
```


Debugging

More gui:

```
$ gdb -tui
```

```
(gdb) layout asm
```

```
(gdb) layout regs
```

Debugging

Helpful GDB Plugins:

PEDA

- ✦ PEDA - Python Exploit Development Assistance for GDB
- ✦ <https://github.com/longld/peda>

GEF

- ✦ GDB Enhanced Features
- ✦ <https://github.com/hugsy/gef>

Lisa.py

- ✦ LLDB
- ✦ Lisa.py: An Exploit Dev Swiss Army Knife.
- ✦ <https://github.com/ant4g0nist/lisa.py>

Voltron

- ✦ Voltron is an extensible debugger UI toolkit written in Python.
- ✦ <https://github.com/snare/voltron>