

CSE451 – Section 7

Project 2 hints
VM (process memory, buffer
overrun attacks)

1

Webserver w/user threads

- Problems with synchronous I/O and user threads:
 - Accept() blocks the main thread
 - Hint: yield() in main thread after handing off the socket id
- Use pthreads for Web server-related parts (parts 4 and 6)
- We *won't* test sioux with user threads

2

Relevant Issues from Project 1

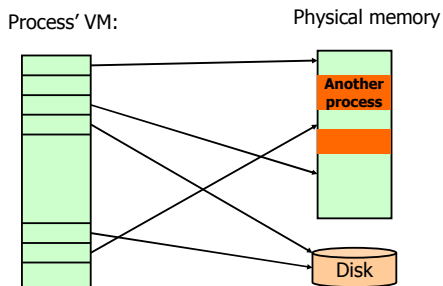
- Methodology for the simple benchmark:
 - Give details about your experimental setting:
 - E.g., used P4 3GHz, averaged over 1M trials of running a system call

3

Project 2 – questions?

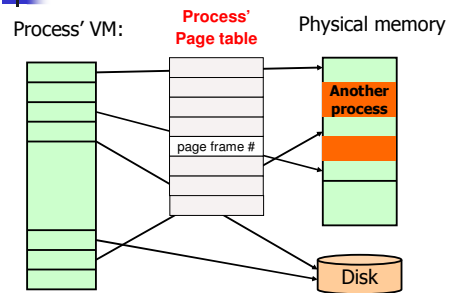
4

Virtual Memory (quick recap)



(Inspired from wikipedia) 5

Virtual Memory (quick recap)



6

Virtual Memory (quick recap)

Process' VM: **Process' page table** Physical memory

How can we use paging to set up sharing of memory between two processes?

7

Process Memory Organization

TEXT Code 0x00000000

DATA Initialized/uninitialized global/static variables static int a = 3;

DATA Heap char* buff = (char*)malloc(...)

STACK Program stack What goes here? 0xffffffff

- In figure, stack grows downwards, but it's not necessary
- How can you determine the direction in which stack grows?

8

Example

- What does the process' memory look like for the following code:

```
void foo(int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
}
void main() {
    foo(1, 2, 3);
}
```

We're here w/ execution

Example taken from "Smashing The Stack For Fun And Profit "

9

Contents of the stack

How many words?

buffer2

buffer1

SFP Saved return address (Where does it point to?)

RET

a 1 word

b 1 word

c 1 word

Bottom of stack (0xffffffff)

Word size (4B)

10

How would you smash the stack?

3 words buffer2

2 words buffer1

1 word SFP

1 word **RET**

1 word a

1 word b

1 word c

Bottom of stack (0xffffffff)

Word size (4B)

11

Buffer overflows

- What's the bug?

```
void foo(char* str) {
    char buffer[5];
    strcpy(buffer, str);
}
void main() {
    char large_string[256];
    ...
    foo(large_string);
}
```

Example taken from "Smashing The Stack For Fun And Profit "

12

The stack contents

```

void foo(char* str) {
    char buffer[5];
    strcpy(buffer, str);
}
void main(int argc,
          char* argv[])
{
    ...
    foo(argv[1]);
}

```

Small addr

Large addr

13

The stack contents

```

void foo(char* str) {
    char buffer[5];
    strcpy(buffer, str);
}
void main(int argc,
          char* argv[]) {
    foo(argv[1]);
}

```

- What cmd-line argument should attacker provide?
 - Length?
 - Contents?

Small addr

Large addr

14

Shell code

- Program to bring up a shell:


```

#include <stdio.h>
void main() {
    char *name[2];
    name[0] = "/bin/sh";
    name[1] = NULL;
    execve(name[0], name, NULL);
}

```
- If you de-assemble, you get:


```

"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

```

15

More information

- Smashing The Stack For Fun And Profit, Aleph One:

<http://www.cs.washington.edu/education/courses/cse599g/CurrentQtr/stack.txt>

16