CSE 351
buffer overflows and lab 3
Buffer overflows

• C performs no bounds-checking on array accesses
  • This makes it fast but also unsafe

• ex) int arr[10]; arr[15] = 3;
  • No compiler warning, just memory corruption

• What symptoms are there when programs write past the end of arrays?
  • Hint: we saw an example of this in lab 0
x86-64 Linux Memory Layout

- **Stack**
  - Runtime stack (8MB limit)
  - E.g., local variables

- **Heap**
  - Dynamically allocated as needed
  - When call `malloc()`, `calloc()`, `new()`

- **Data**
  - Statically allocated data
    - Read-only: string literals
    - Read/write: global arrays and variables

- **Text / Shared Libraries**
  - Executable machine instructions
  - Read-only

Hex Address: 0x7FFFFFFF

not drawn to scale
Stack layout

- To which byte does buf[17] refer to in this example?
- In buffer overflow attacks, malicious users pass values to attempt to overwrite important parts of the stack or heap
- For example, an attacker could overwrite the return instruction pointer with the address of a malicious block of code
Protecting against overflows

- `fgets(char* s, int size, FILE* stream)`
  - Takes a size parameter and will only read that many bytes from the given input stream
- `strncpy(char* dest, const char* src, size_t n)`
  - Will copy at most `n` bytes from `src` to `dest`
- Stack canaries
  - Use a random integer before return instruction pointer
  - Check if tampered
- Data execution prevention
  - Mark some parts of the memory (notably the stack) as non-executable.
Lab 3: Buffer overflow exploits

• The exploitable function in lab 3 is called Gets (capital ‘G’)
  • It is called from the getbuf function

• getbuf allocates a small array and reads user input into it via Gets.

• If the user input is too long, then certain values on the stack within the getbuf function will be overwritten...
Lab 3: Buffer Overflow

This has a buffer overflow

```c
int getbuf() {
    char buf[36];
    Gets(buf);
    return 1;
}
```

Why?
- `Gets()` doesn’t check the length of the buffer

The Stack in `getbuf()`
- `return addr`
- `saved regs (if any)`
- `local vars`
Lab 3: Buffer Overflow

This has a buffer overflow

```c
int getbuf() {
    char buf[36];
    Gets(buf);
    return 1;
}
```

Why?
- `Gets()` doesn’t check the length of the buffer

The Stack in `getbuf()`

- return addr
- saved regs (if any)
- local vars
Lab 3: Buffer Overflow

This has a buffer overflow

```c
int getbuf() {
    char buf[36];
    Gets(buf);
    return 1;
}
```

Why?
- `Gets()` doesn’t check the length of the buffer

The Stack in `getbuf()`

- `return addr`
- `saved regs (if any)`
- `buf[35]`
- `buf[0]`
- `36 bytes`
Level 0: Call smoke()

Goal: call the smoke() function from getbuf()

```c
int getbuf() {
    char buf[36];
    Gets(buf);
    return 1;
}
```

How?
- overwrite the return address so we “return” to smoke()
Lab 3: Understand the tools

• **sendstring** – Use to generate your malicious strings
  - Takes a list of space-separated hex values and formats them in raw bytes suited for exploits

• **gdb** – You will use this a lot to inspect your code
  - `set args -u <username>`
    - Set the argument to the program
  - `x/40wx ($rsp - 40)`
    - Show the 40 bytes above rsp
    - Change `w` to `g` to check the value in 8 byte chunks.
  - `b *(&getbuf + 12)`
    - Create a breakpoint at 12 bytes away after the start of getbuf

• **bufbomb** – u [UW_NetID] – Everyone’s lab is different
  - Your username alters the lab slightly
Level 0 walkthrough

• **Goal:** Make `getbuf()` jump to a function called `smoke()`

• **How?** Overwrite the return address with your own
  • Write past the end of the buffer to do this
Passing in the 7\textsuperscript{th} argument

https://courses.cs.washington.edu/courses/cse351/16sp/lectures/06-procedures_16sp.pdf#page=72