Buffer Overflows

Many of the following slides are based on those from

Complete Powerpoint Lecture Notes for Computer Systems: A Programmer's Perspective (CS:APP)

Randal E. Bryant and David R. O'Hallaron

http://csapp.cs.cmu.edu/public/lectures.html

The book is used explicitly in CS 2505 and CS 3214 and as a reference in CS 2506.
Buffer Overflows

What is a buffer overflow?
How can it be exploited?
How can it be avoided?
  - Through programmer measures
  - Through system measures (and how effective are they?)
String Library Code

Implementation of Unix function `gets`

No way to specify limit on number of characters to read

```c
/* Get string from stdin */
char *gets(char *dest)
{
    int c = getc();
    char *p = dest;
    while (c != EOF && c != '\n') {
        *p++ = c;
        c = getc();
    }
    *p = '\0';
    return dest;
}
```
Vulnerable Buffer Code

int main() {
    printf("Type a string:");
    echo();
    return 0;
}

/* Echo Line */
void echo() {
    char buf[4]; /* Way too small! */
    gets(buf);
    puts(buf);
}
Buffer Overflow Executions

unix> ./bufdemo
Type a string: 123
123

unix> ./bufdemo
Type a string: 12345
Segmentation Fault

unix> ./bufdemo
Type a string: 12345678
Segmentation Fault
Buffer Overflow Stack

/* Echo Line */
void echo()
{
    char buf[4]; /* Way too small! */
    gets(buf);
    puts(buf);
}

echo:
    pushl %ebp    # Save %ebp on stack
    movl %esp,%ebp
    subl $20,%esp  # Allocate space on stack
    pushl %ebx    # Save %ebx
    addl $-12,%esp  # Allocate space on stack
    leal -4(%ebp),%ebx # Compute buf as %ebp-4
    pushl %ebx    # Push buf on stack
call gets      # Call gets
    ...
Buffer Overflow Stack Example

Before call to `gets`

```
unix> gdb bufdemo
(gdb) break echo
Breakpoint 1 at 0x8048583
(gdb) run
Breakpoint 1, 0x8048583 in echo ()
(gdb) print /x *(unsigned *)$ebp
$1 = 0xbfffff8f8
(gdb) print /x *((unsigned *)$ebp + 1)
$3 = 0x804864d
```

Stack Frame for `main`

```
08 04 86 4d 0xbfffff8d8
```

Stack Frame for `echo`

```
buf ff ff f8 f8 0xbfffff8d8
```

```
8048648: call 804857c <echo>
804864d: mov 0xfffffffff8(%ebp),%ebx # Return Point
```
Buffer Overflow Example #1

Before Call to `gets`

- **Stack Frame for main**
  - Return Address
  - Saved `%ebp`:
    - [3] [2] [1] [0]
  - Stack Frame for `echo`

- **Stack Frame for echo**
  - `buf`

Input = “123”

- **Stack Frame for main**
  - Return Address
  - Saved `%ebp`:
    - [3] [2] [1] [0]
  - Stack Frame for `echo`

- **Stack Frame for echo**
  - `buf`
  - `0xbfffff8d8`

No Problem
Input = “12345”

Saved value of %ebp set to 0xbfff0035

Bad news when later attempt to restore %ebp

```
8048592: push %ebx
8048593: call 80483e4 <_init+0x50>  # gets
8048598: mov 0xfffffffffe8(%ebp),%ebx
804859b: mov %ebp,%esp
804859d: pop %ebp  # %ebp gets set to invalid value
804859e: ret
```
Buffer Overflow Stack Example #3

Input = “12345678”

No longer pointing to desired return

Stack Frame for main

Stack Frame for echo

%ebp and return address corrupted

Invalid address

8048648: call 804857c <echo>
804864d: mov 0xfffffffffe8(%ebp),%ebx # Return Point
Malicious Use of Buffer Overflow

```c
void foo() {  
    bar();  
    ...  
}

void bar() {  
    char buf[64];  
    ...  
}
```

Stack right after call to `bar()`

- `foo` stack frame
- `bar` stack frame

B = &buf[0]
Malicious Use of Buffer Overflow

void foo()
{
  bar();
  ...
}

return address

toB

void bar()
{
  char buf[64];
  gets(buf);
  ...
}

gets() can now write whatever it wants, beginning at buf[0]

Stack right after call to gets()
Malicious Use of Buffer Overflow

```c
void foo()
{
    bar();
    ...
}
```

```c
void bar()
{
    char buf[64];
    gets(buf);
    ...
}
```

Stack right before return from `gets()`

- Input string to `gets()` contains byte representation of executable code
- And new pointer that overwrites return address with address of buffer
- And padding (which overwrites rest of `bar()`’s frame)
- Return address to `bar` overwritten with address of buffer
- New pointer that overwrites return address with address of buffer
- Stack frame of `bar` overwritten with buffer data

Diagram:
- Stack frame after `gets()` call
- Return address overwritten
- Buffer data overwritten
Malicious Use of Buffer Overflow

void foo()
{
    bar();
    ...
}

void bar()
{
    char buf[64];
    gets(buf);
    ...
}

Stack right after return to \texttt{bar()} from \texttt{gets()}

When \texttt{bar()} executes \texttt{ret}, will jump to exploit code
Use Library Routines that check/limit string lengths

- `fgets` instead of `gets`
- `strncpy/strlcpy` instead of `strcpy`
- `snprintf` instead of `sprintf`
- Don’t use `scanf` with `%s` conversion specification
  - Use `fgets` to read the string

/* Echo Line */
void echo()
{
    char buf[4]; /* Way too small! */
    fgets(buf, 4, stdin);
    puts(buf);
}

http://lwn.net/Articles/507319/