Stack Buffer Overflows

ACKNOWLEDGEMENT:
This lecture uses slides that were prepared by Prof. Dave O'Halloran (CMU)

Internet Worm and IM War

November, 1988
- Internet Worm attacks thousands of Internet hosts.
- How did it happen?

July, 1999
- Microsoft launches MSN Messenger (instant messaging system).
- Messenger clients can access popular AOL Instant Messaging Service (AIM) servers
Internet Worm and IM War (cont.)

August 1999

- Mysteriously, Messenger clients can no longer access AIM servers.
- Microsoft and AOL begin the IM war:
  - AOL changes server to disallow Messenger clients
  - Microsoft makes changes to clients to defeat AOL changes.
  - At least 13 such skirmishes.
- How did it happen?

The Internet Worm and AOL/Microsoft War were both based on stack buffer overflow exploits!
- many Unix functions do not check argument sizes.
- allows target buffers to overflow.

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;
}
```

- Similar problems with other Unix functions
  - `strcpy`: Copies string of arbitrary length
  - `scanf`, `fscanf`, `sscanf`, when given `%s` conversion specification
Vulnerable Buffer Code

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

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

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
```
IA32/Linux Stack Frame

Current Stack Frame (“Top” to Bottom)
- Parameters for function about to call
  - “Argument build”
- Local variables
  - If can’t keep in registers
- Saved register context
- Old frame pointer

Caller Stack Frame
- Return address
  - Pushed by call instruction
- Arguments for this call

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
    addl $-12,%esp # Allocate space on stack
    leal -4(%ebp),%ebx # Compute buf as %ebp-4
    pushl %ebx
    call gets       # Push buf on stack
    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`  
$1 = 0xbffff8f8  
(gdb) `print /x *((unsigned *)$ebp + 1)`  
$3 = 0x804864d

8048648: call 804857c <echo>  
804864d: mov 0xffffffe8(%ebp),%ebx  
# Return Point

Before Call to `gets`

Input = “123”  

No Problem
Buffer Overflow Stack Example #2

Input = “12345”

```
8048592: push %ebx
8048593: call 80483e4 <_init+0x50>  # gets
8048598: mov 0xffffffe8(%ebp),%ebx
804859b: mov %ebp,%esp
804859d: pop %ebp
# %ebp gets set to invalid value
804859e: ret
```

Saved value of %ebp set to 0xbfff0035
Bad news when later attempt to restore %ebp

```
0xbffff8d8
Return Address
Saved %ebp
[3][2][1][0]
buf
Stack Frame for echo
```

```
Stack Frame for main
```

```
echo code:
804859c: mov 0xffffffe8(%ebp),%ebx
# Return Point
0xbffff8d8
Return Address
Saved %ebp
[3][2][1][0]
buf
Stack Frame for echo
```

```
Stack Frame for main
```

```
Saved value of %ebp set to 0xbfff0035
Bad news when later attempt to restore %ebp
```

Buffer Overflow Stack Example #3

Input = “12345678”

```
8048648: call 804857c <echo>
804864d: mov 0xffffffe8(%ebp),%ebx  # $%ebp gets set to invalid value
8048652: mov %ebp,%esp
8048654: pop %ebp
8048656: ret
```

```
0xbffff8d8
Return Address
Saved %ebp
[3][2][1][0]
buf
Stack Frame for echo
```

Invalid address
No longer pointing to desired return point

```
Stack Frame for main
```

```
804864c: mov 0xffffffe8(%ebp),%ebx  # %ebp and return address corrupted
```

```
Stack Frame for echo
Invalid address
```

```
Stack Frame for main
```

```
 echo code:
804859c: mov 0xffffffe8(%ebp),%ebx  # $%ebp gets set to invalid value
804859e: ret
```

```
Stack Frame for main
```

```
Saved value of %ebp set to 0xbfff0035
Bad news when later attempt to restore %ebp
```

```
Stack Frame for echo
```

```
Invalid address
```

```
Stack Frame for main
```

```
echo code:
804859c: mov 0xffffffe8(%ebp),%ebx  # %ebp and return address corrupted
804859e: ret
```

```
Stack Frame for main
```

```
Saved value of %ebp set to 0xbfff0035
Bad news when later attempt to restore %ebp
```

```
Stack Frame for echo
```

```
Invalid address
```

```
Stack Frame for main
```

```
echo code:
804859c: mov 0xffffffe8(%ebp),%ebx  # %ebp and return address corrupted
804859e: ret
```

```
Stack Frame for main
```

```
Saved value of %ebp set to 0xbfff0035
Bad news when later attempt to restore %ebp
```

```
Stack Frame for echo
```

```
Invalid address
```
Malicious Use of Buffer Overflow

- Input string contains byte representation of executable code
- Overwrite return address with address of buffer
- When bar() executes ret, will jump to exploit code

LC3 Activation Record

- Dynamic link
- Return address
- Return value

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Offset</th>
<th>Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>int</td>
<td>4</td>
<td>NoName</td>
</tr>
<tr>
<td>b</td>
<td>int</td>
<td>5</td>
<td>NoName</td>
</tr>
<tr>
<td>w</td>
<td>char</td>
<td>-2</td>
<td>NoName</td>
</tr>
</tbody>
</table>
Exploits Based on Buffer Overflows

Buffer overflow bugs allow remote machines to execute arbitrary code on victim machines.

Internet worm
- Early versions of the finger server (fingerd) used `gets()` to read the argument sent by the client:
  - `finger droh@cs.cmu.edu`
- Worm attacked fingerd server by sending phony argument:
  - `finger "exploit-code padding new-return-address"`
  - exploit code: executed a root shell on the victim machine with a direct TCP connection to the attacker.

Exploits Based on Buffer Overflows

Buffer overflow bugs allow remote machines to execute arbitrary code on victim machines.

IM War
- AOL exploited existing buffer overflow bug in AIM clients
- exploit code: returned 4-byte signature (the bytes at some location in the AIM client) to server.
- When Microsoft changed code to match signature, AOL changed signature location.
Avoiding Overflow Vulnerability

Use Library Routines that Limit String Lengths

- `fgets` instead of `gets`
- `strncpy` instead of `strcpy`
- Don't use `scanf` with `%s` conversion specification
  - Use `fgets` to read the string

Extra Credit Assignment

Construct a buffer overflow attack on the LC-3

Assume that there is a procedure `test()` as shown below
Assume that the activation record for a procedure follows the format discussed in previous lectures
Write LC3 assembly code for `test()`, and construct an attack where your exploit code is executed

```
void test()
{
    char buf[4];  /* Way too small! */
    gets(buf);
    return(1);
}
```

Assume that the input string typed in by the user is stored in a packed array, i.e. assume that each character takes 1 byte instead of 1 word as is the default in the LC3

You can make your exploit code print out a message to the screen. Can you construct an attack where your exploit code is executed, the damage to the stack is repaired, and then `test()` appears to return "normally"?