Buffer Overflows

Address space layout
the stack discipline
+ C's lack of bounds-checking
HUGE PROBLEM

getaddrinfo ()
Feb. 2016

https://cs.wellesley.edu/~cs240/s20/
x86-64 Linux memory layout

not drawn to scale
String library code

C standard library function `gets()`

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

What could go wrong in this code?

Same problem in many functions:

- `strcpy`: Copies string of arbitrary length
- `scanf`, `fscanf`, `sscanf`, when given `%s` conversion specification
Vulnerable buffer code: C

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

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

```bash
$ ./bufdemo
Type a string: 123
123
```

```bash
$ ./bufdemo
Type a string: 0123456789012345678901234
Segmentation Fault
```

```bash
$ ./bufdemo
Type a string: 012345678901234567890123
012345678901234567890123
```
Vulnerable buffer code: disassembled x86

00000000004006cf <echo>:

00000000004006cf: 48 83 ec 18
sub $24,%rsp
00000000004006d3: 48 89 e7
mov %rsp,%rdi
00000000004006d6: e8 a5 ff ff ff
callq 400680 <gets>
00000000004006db: 48 89 e7
mov %rsp,%rdi
00000000004006de: e8 3d fe ff ff
callq 400520 <puts@plt>
00000000004006e3: 48 83 c4 18
add $24,%rsp
00000000004006e7: c3
retq

4006e8: 48 83 ec 08
sub $0x8,%rsp
4006ec: b8 00 00 00 00 00
mov $0x0,%eax
4006f1: e8 d9 ff ff ff
callq 4006cf <echo>
4006f6: 48 83 c4 08
add $0x8,%rsp
4006fa: c3
retq

Buffer Overflows
Buffer overflow example: before input

Before call to gets

Stack frame for call_echo

<table>
<thead>
<tr>
<th>Return Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00 00</td>
</tr>
<tr>
<td>00 40 06 f6</td>
</tr>
</tbody>
</table>

20 bytes unused

[3] [2] [1] [0]

buf ← %rsp

void echo() {
    char buf[4];
    gets(buf);
    ...
}

echo:
    subq $24, %rsp
    movq %rsp, %rdi
    call gets
    ...

call_echo:

    ...
    4006f1: callq 4006cf <echo>
    4006f6: add $0x8,%rsp
    ...

Buffer Overflows 6
### Buffer overflow example: input #1

**After call to gets**

<table>
<thead>
<tr>
<th>Stack frame for call_echo</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00 00</td>
</tr>
<tr>
<td>00 40 06 f6</td>
</tr>
<tr>
<td>00 32 31 30</td>
</tr>
<tr>
<td>39 38 37 36</td>
</tr>
<tr>
<td>35 34 33 32</td>
</tr>
<tr>
<td>31 30 39 38</td>
</tr>
<tr>
<td>37 36 35 34</td>
</tr>
<tr>
<td>33 32 31 30</td>
</tr>
</tbody>
</table>

**Return Address**

**Call frame for call_echo**

- `$ ./bufdemo`
- Type a string: `01234567890123456789012` `01234567890123456789012`

---

**Buffer Overflows**

```c
void echo()
{
    char buf[4];
    gets(buf);
    ...
}
```

**Stack frame for call echo**

- Return Address
- `buf ← %rsp`

**call echo**:

```
. . .
4006f1: callq 4006cf <echo>
4006f6: add $0x8,%rsp
. . .
```

---

Overflowed buffer, but did not corrupt state.
Buffer overflow example: input #2

After call to gets

void echo()
{
    char buf[4];
    gets(buf);
    ...
}

echo:
    subq $24, %rsp
    movq %rsp, %rdi
    call gets
    ...

call_echo:
    ...
    4006f1: callq 4006cf <echo>
    4006f6: add $0x8,%rsp
    ...

buf ← %rsp

unix> ./bufdemo
Type a string: 0123456789012345678901234
Segmentation Fault

Overflowed buffer and corrupted return pointer
Buffer overflow example: input #3

After call to gets

<table>
<thead>
<tr>
<th>Stack frame for call_echo</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00 00 00 00</td>
</tr>
<tr>
<td>00 40 06 00</td>
</tr>
<tr>
<td>33 32 31 30</td>
</tr>
<tr>
<td>39 38 37 36</td>
</tr>
<tr>
<td>35 34 33 32</td>
</tr>
<tr>
<td>31 30 39 38</td>
</tr>
<tr>
<td>37 36 35 34</td>
</tr>
<tr>
<td>33 32 31 30</td>
</tr>
</tbody>
</table>

Return Address

call_echo:

void echo()
{
    char buf[4];
    gets(buf);
    ...
}

echo:
    subq $24, %rsp
    movq %rsp, %rdi
    call gets
    ...

buf ← %rsp

unix> ./bufdemo-nsp
Type a string: 012345678901234567890123
012345678901234567890123

 Overflowed buffer, corrupted return pointer, but program seems to work!
Buffer overflow example: input #3

After call to gets

Stack frame for call_echo

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>00</td>
<td>40</td>
<td>06</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>38</td>
<td>37</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>30</td>
<td>39</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>36</td>
<td>35</td>
<td>34</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>32</td>
<td>31</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

Return Address

buf ← %rsp

Some other place in .text

```
 mov  %rsp,%rbp
 mov  %rax,%rdx
 shr  $0x3f,%rdx
 add  %rdx,%rax
 sar  %rax
 jne  400614
 pop  %rbp
 retq
```

“Returns” to unrelated code
Lots of things happen, without modifying critical state
Eventually executes retq back to main
Exploiting buffer overflows

Input string contains byte representation of executable code
Overwrite return address A with address of buffer (need to know B)
When \texttt{bar()} executes \texttt{ret}, will jump to exploit code (instead of A)
Exploits in the wild

Buffer overflow bugs allow remote attackers to execute arbitrary code on machines running vulnerable software.

1988: Internet worm

Early versions of the finger server daemon (fingerd) used `gets()` to read the argument sent by the client:

```
finger somebody@cs.wellesley.edu
```

Attack by sending phony argument:

```
finger "exploit-code padding new-return-address"
```

... Still happening

"Ghost:" 2015

gethostbyname()
Heartbleed (2014)

Buffer over-read in OpenSSL
Widely used encryption library (https)
“Heartbeat” packet
Specifies length of message
Server echoes that much back
Library just “trusted” this length
Allowed attackers to read contents of memory anywhere they wanted
~17% of Internet affected
“Catastrophic”
Github, Yahoo, Stack Overflow, Amazon AWS, ...

Heartbeat – Normal usage
Server, send me this 4 letter word if you are there: "bird"

Client

Heartbeat – Malicious usage
Server, send me this 500 letter word if you are there: "bird"

Client

By FenixFeather - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=32276981
Avoiding overrun vulnerabilities

1. Use a memory-safe language (not C)!

2. If you have to use C, use library functions 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
     - Or use `%ns` where `n` is a suitable integer

Other ideas?
System-level protections

Available in modern OSs/compilers/hardware
(We disabled these for buffer assignment.)

1. Randomize stack base, maybe frame padding

2. Detect stack corruption
   save and check stack "canary" values

3. Non-executable memory segments
   stack, heap, data, ... everything except text
   hardware support

Helpful, not foolproof!
Return-oriented programming, over-reads, etc.