Buffer Overflows

COMP 435
Fall 2017
Prof. Cynthia Sturton

Buffers

char sample[10];
int i;

for (i=0; i<=9; i++)
sample[i] = ‘A’;

Exam Mon., Nov. 6
○ Covers material since last exam, including today’s lecture
○ Review in OH Fri., Nov. 3, 10-12 FB 354

Poster group info due tomorrow (or lose points!)
○ Group sign-up using google form
○ OR Email instr-435-cs@cs.unc.edu to be grouped

Assignment 3 due Wed., Nov. 8
○ Can work on it after the exam
○ New this time: Sakai will stay open for resubmissions
Buffers

```c
char sample[10];
int i;
for (i=0; i<=9; i++)
sample[i] = 'A';
sample[10] = 'B';
```

Buffers in Memory

Terminology

- Buffer Overflow
- Buffer Overrun
- Buffer Overflow Attack
- Stack Smashing
Buffer Overflow Attack

- 1988 Morris Worm
- 2014 Heartbleed

Memory Layout

<table>
<thead>
<tr>
<th>Code &amp; Data</th>
<th>User A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code &amp; Data</td>
<td>User B</td>
</tr>
<tr>
<td>Code &amp; Data</td>
<td>Operating System</td>
</tr>
</tbody>
</table>

Proc 1
- Code
- Data
Proc 2
- Code
- Data

Memory Layout: one process

Stack
- Heap
  - Local Data
  - Local Code
  - System Data
  - System Code
```c
int p2(char *input){
    char buffer[100];
    strcpy(buffer, input);
    return 0;
}
```
An Attack

```c
int p2(char *input){
    char buffer[100];
    strcpy(buffer, input);
    return 0;
}
```

Defense: Canary

- Add a secret value to the stack
- Check value before returning

Compromised Stack
Attack

Guess Canary

Defense: $W \oplus X$

Regions of memory can be either writable or executable

Attack: Return to LibC

Point to existing code in the system

Compromised Stack

Procedure $p_1$

```
{...
call $p_2$;
...}
```

Points to elsewhere in memory

Stack Frame $p_1$

Stack Frame $p_2$
**Attack: Return oriented programming**

Point to “gadgets” within the existing code

**Current State**

We have trained the attackers very well!

**Heartbleed**

**TLS Heartbeat**

heartbeat request: 3 bytes, “cat”

heartbeat reply: “cat”
TLS Heartbleed Attack

- Heartbeat request: 64 Kbytes, "cat"
- Heartbeat reply: "cat0xCAFEPasswordKeys"

History of Heartbleed
- Bug introduced to code repository Dec 2011
- Published for comment Feb 2012
- Included and enabled by default in OpenSSL March 2012
- First discovered March 2014

Current State
Buffer overflow and overruns are the major source of vulnerabilities in SW

Integer Overflows

```c
int *vulnerable(int *array, int len) {
    int *myarray;
    myarray = malloc(len * sizeof(int));
    if (myarray == NULL) {
        return -1;
    }
    for (i = 0; i < len; i++) {
        myarray[i] = array[i];
    }
    return myarray;
}
```
Overflow Countermeasures

- Bounds Checking
- Input validation & sanitization
- Use safe utilities
- Least privilege
- Sandboxing

Bounds Checking

- Manual code review
- Static analysis
- Dynamic analysis

Input Validation & Sanitization

- Use a template
  (919) 555-1234

Input Validation & Sanitization

- Use a template
  (919) 555-1234
- Templates prescribe good behavior
Templates are Difficult

(919) 555-1234
1-919-555-1234
+1 919 555 1234
919 555 1234
919.555.1234

Input Validation & Sanitization

The length of buffer writes should be dictated by buffer size, not by user-provided input.

Input Validation & Sanitization

Never trust user-supplied input.

Use Safe Utilities

strcpy vs strncpy

```c
char str1[] = "hello";
char str2[6];
/* copy to terminating null */
strcpy (str2, str1);
/* copy only 6 chars */
strncpy (str2, str1, 6);
```
Use Safe Utilities

- strcpy vs strncpy
- strcmp vs strncmp
- sprintf vs snprintf

Least Privilege

Subject should have access to fewest number of objects necessary to do its work

Least Privilege

- Code should have only the permissions needed
- Limits the damage done by compromise

Sandboxing

Contain sections of code within a specified address range
Programming Language

The choice of language plays a role as well
Type Safety
Memory Safety

In-class Exercise