Smashing Fun with Smashing Stacks

Jed Liu

Department of Computer Science
Cornell University

CS 316 Section
23-26 October 2007
InFamous Examples

- Morris worm
  - Robert Morris, 2 Nov 1988
  - Spread partly by exploiting finger

- Witty worm
  - 19 Mar 2004
  - Exploited BlackICE firewall
  - In $\frac{1}{2}$ hour:
    - Infected 12,000 machines
    - Generated 90 Gb/s UDP traffic

- Slammer worm
  - 25 Jan 2003
  - Exploited Microsoft SQL Server
  - Infected 75,000 machines
    - Most infected within 10 minutes
Program Memory Layout

Stack
↓
Heap
↑
BSS\(^1\)
(static vars)
Code
Reserved

\(^1\)Block Started by Symbol
Stack Frame Layout

- return address
- saved fp
- saved registers
- local variables
- arguments to callee
A Buffer Overrun Vulnerability

return address
saved fp
saved registers
local variables
arguments to callee

void f(char* str) {
    int i = 0;
    char[8] buf;
    while (str[i] != '\0') {
        buf[i] = str[i];
        i++;
    }
    ...
}
A Buffer Overrun Vulnerability

```c
void f(char* str) {
    int i = 0;
    char[8] buf;
    while (str[i] != '\0') {
        buf[i] = str[i];
        i++;
    }
    ...
}
```
A Buffer Overrun Vulnerability

```c
void f(char* str) {
    int i = 0;
    char[8] buf;
    while (str[i] != '\0') {
        buf[i] = str[i];
        i++;
    }
    ...
}

void main() {
    f("Hello!");
}
```
A Buffer Overrun Vulnerability

```c
void f(char* str) {
    int i = 0;
    char[8] buf;
    while (str[i] != '\0') {
        buf[i] = str[i];
        i++;
    }
    ...
}

void main() {
    f("Hello, world!");
}
```
A Buffer Overrun Exploit

![Diagram of buffer overrun exploit]

- **0xdeadbf03**: return address
- **0xdeadbeff**: buf
- **0xdeadbefb**: i
- **0xdeadbef7**: low
- **0xdeadbef3**: high

Jed Liu
Smashing Fun with Smashing Stacks
A Buffer Overrun Exploit

0xdeadbf03 0xdeadbeef
0xdeadbeff nop
0xdeadbefb nop
0xdeadbef7 jr $1
0xdeadbef3 ori $1,$1,0xbeef
0xdeadbeef lui $1, 0xdead

i
Countermeasures

Exist at several different levels:

- **Programming language**
  - Type safety: e.g., Java, Cyclone, Python, ML, ...

- **Language library**
  - e.g., `strcpy` vs. `strncpy`, `strcpy_s`

- **Compiler**
  - Static analysis
  - Instrument code w/ runtime checks
  - Insert stack canaries

- **OS/Hardware**
  - No-execute bit (OS or CPU)
  - Address space layout randomization (OS)
  - Deep packet inspection
    - Check incoming network traffic for signatures