Smashing the Stack for Fun and Profit
General Overview

- “Smashing the Stack” is a type of buffer overflow attack – overwriting the return address to redirect control to attack code

- Most common buffer overflow error since it is the easiest to make and take advantage of
Buffer Overflows

- No one would do something like this, right?
Slammer Worm Info

- First example of a high speed worm (previously only existed in theory)
- Infected a total of 75,000 hosts in about 30 minutes
- Infected 90% of vulnerable hosts in 10 min
- Exploited a vulnerability in MS SQL Server Resolution Service, for which a patch had been available for 6 months
Code randomly generated an IP address and sent out a copy of itself

Used UDP – limited by bandwidth, not network latency (TCP handshake).

Packet was just 376 bytes long...

Spread doubled every 8.5 seconds

Max scanning rate (55 million scans/second) reached in 3 minutes
Slammer Worm – Eye Candy
Slammer Worm – Eye Candy

Map Source: www.caida.org

http://www.caida.org

Copyright (C) 2003 UC Regents

Set Jan 25 06:00:00 2003 (UTC)
Number of hosts infected with Sapphire: 74855
Anatomy of Memory

Assumptions

• Stack grows down

• Stack pointer points to the last address on the stack
Let us consider how the stack of this program would look:

```c
void function(int a, int b, int c){
    char buf[16];
}

int main(){
    function(1,2,3);
}
```
Stack Frame

**function prolog**

- `sw $ra, -4(sp)`
- `sw $s0, -8(sp)`

- `addi $sp, $sp, -24`

Allocates space for stack frame

- **Return Address**
- **Saved registers**
- **Local Variables**
  - (char buf[4])

Higher Memory Addresses
Linear View Of Frame/Stack

Bottom of memory
Top of stack

buf 16
s0 4
ra 4

Bottom of memory
Top of stack
Example Program 2

```c
void function(char *str){
    char buf [16];
    strcpy(buf, str);
}

int main(){
    char large_string[32];
    int i;
    for (i = 0; i < 31; i++){
        large_string[i] = 'A';
    }
    function(large_string);
}
```
When this program is run, it results in an exception

The return address is overwritten with ‘AAAA’ (0x41414141)

Function exits and goes to execute instruction at 0x41414141.....
Example Program 3

Can we take advantage of this to execute code, instead of crashing?

```c
void function(int a, int b, int c){
    char buf[4];
    int *r;
    r = buf + 20;
    (*r) += 8;
}

int main(){
    int x = 0;
    function(1,2,3);
    x = 1;
    printf("%d\n", x);
}
```
Example Program 3

This causes it to skip the assignment of 1 to x, and prints out 0 for the value of x
We have seen how we can overwrite the return address of our own program to crash it or skip a few instructions – basically just writing a buggy program

How can these principles be used by an attacker to hijack the execution of a program?

Attacker can use some kind of user/network input to inject attack code into such a buffer