Defending Computer Networks

Lecture 2: Vulnerabilities

Stuart Staniford
Adjunct Professor of Computer Science
Logistics

• Still space in class
• Restriction to CS M.Eng will be lifted shortly
• HW1 probably given out next time
• Website not up yet (my bad).
Main Goals for Today

• Understand system() function vulnerabilities
• Outline understanding of buffer overflow vulnerabilities
Interesting News This Week

**JPMorgan and Other Banks Struck by Hackers**

A number of United States banks, including JPMorgan Chase and at least four others, were struck by hackers in a series of coordinated attacks this month, according to four people briefed on a continuing investigation into the crimes.

The hackers infiltrated the networks of the banks, siphoning off gigabytes of data, including checking and savings account information, in what security experts described as a sophisticated cyberattack.

The motivation and origin of the attacks are not yet clear, according to investigators. The F.B.I. is involved in the investigation, and in the past few weeks a number of security firms have been brought in to conduct studies of the penetrated computer networks.

Russian hackers attacked the U.S. financial system in mid-August, infiltrating and stealing data from **JPMorgan Chase & Co. (JPM)** and at least one other bank, an incident the FBI is investigating as a possible retaliation for government-sponsored sanctions, according to two people familiar with the probe.

System() Function Vulnerabilities

• Very basic class of C/Unix vulnerability
• “man 3 system”
• Been known for decades
• Still occurs, however.
• Let’s work through an example
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <assert.h>
#include <strings.h>
#include <unistd.h>
#include <stdlib.h>

// This code is a very short hack to illustrate server vulnerabilities!!
// Do not write production code like this!!!
int sockFd;
int connFd;
unsigned short port = 3333;
struct sockaddr_in serverAddress;
struct sockaddr_in clientAddress;
void setupSocket(void)
{
    unsigned clientLen;
    assert( (sockFd = socket(AF_INET, SOCK_STREAM, 0)) >= 0);
    bzero(&serverAddress, sizeof(struct sockaddr_in));
    serverAddress.sin_family = AF_INET;
    serverAddress.sin_addr.s_addr = INADDR_ANY;
    serverAddress.sin_port = htons(port);
    assert(bind(sockFd, (struct sockaddr *)&serverAddress, sizeof(struct sockaddr_in)) >= 0);
    assert(listen(sockFd, 5)>=0);
    clientLen = sizeof(struct sockaddr_in);
    assert( (connFd = accept(sockFd, (struct sockaddr *)&clientAddress, &clientLen)) > 1);
}
int getLineFromSocket(char* buffer, int len)
{
    int n;
    assert(write(connFd, "Type Symbol>", 12) >= 0);
    n = read(connFd, buffer, len);
    buffer[n-2] = '\0';
    return n;
}
void extractCountFromFile(char* fileName, char* answer)
{
    char buf[256];
    char* start;

    FILE* file = fopen(fileName, "r");
    assert(file);
    fgets(buf, 256, file);
    for(start = buf; *start; start++)
    {
        if(*start == ' ' || *start == '\t')
            continue;
        else
            break;
    }
    if(*start)
    {
        char* end = index(start, ' ');
        if(end)
        {
            *end = '\n';
            *(++end) = '\0';
            strncpy(answer, start);
        }
    }
}
void processLine(char* buffer, int len)
{
    // line format is "username\n"
    char answer[256];
    char command[256];
    sprintf(command, "ps aux |grep %s |wc > tmp.txt", buffer);
    fprintf(stdout, "%s\n", command);
    system(command);
    extractCountFromFile("tmp.txt", answer);
    assert(write(connFd, answer, strlen(answer)) >= 0);
}
int main(int argc, char* argv[]) 
{
    char buf[256];
    int n;
    if(argc == 2)
    {
        port = atoi(argv[1]);
    }
    setupSocket();
    while(getLineFromSocket(buf, 256))
    {
        processLine(buf, n);
    }
}
Live Demonstration of Exploitation
General Point

• When writing a server
  – Task is to mediate access to server’s resources
  – Not grant arbitrary access
  – Have to be very careful in channeling
    • Constrained client-server protocol
    • To general-purpose OS/computer

• Attackers are evil/bad/smart/patient
• They are out to get you!
Side Note

• SQL Injection Vulnerabilities are closely related
  – Eg ‘;’ passed through to SQL server is a statement separator there too.

• The general issue is failure to properly sanitize input before passing it to general execution engines.
Interlude

Vulnerability Discovery Rate for Adobe Reader (SecurityFocus)
Buffer Overflow Vulnerabilities

• Most important early class of vulnerabilities
  – Still important
• Will start today, finish in subsequent lecture(s)
• Today, will introduce a “fictionalized” account
  – How things used to be 10-20 years ago
  – Simpler to understand
  – Will not match what happens if you look at output of a modern compiler
    • Modern OS/compilers have numerous defenses
    • Still vulnerable though, just more complex to exploit
  – We will expand into more realistic detail next time
• Loosely based on Aleph1 *Smashing Stack for Fun and Profit.*
Example 1

```c
void myFunc(int a, int b, int c) {
    char buffer1[5];
    char buffer2[10];
}

int main(int argc, char* argv[]) {
    myFunc(1,2,3);
}
```
Assembler

- Function Call:
  - pushl $3
  - pushl $2
  - pushl $1
  - call myFunc

- Function Prologue:
  - pushl %ebp
  - movl %esp,%ebp
  - subl $20,%esp
Stack in Example 1

Buffer2  Buffer1  SFP  Ret  1  2  3

Top of Memory

Stack Pointer %esp

Top of Stack

Frame Pointer %ebp

Saved Frame Pointer
Example 2

```c
void myFunc(char *str)
{
    char buffer[64];
    strcpy(buffer, str);
}

int main(int argc, char* argv[])
{
    char large_string[256];
    int i;
    for( i = 0; i < 255; i++)
    {
        large_string[i] = 'A';
        myFunc (large_string);
    }
}
Stack in Example 2 right before strcpy()
Stack in Example 2 right after `strcpy()`

Top of Memory

Top of Stack

Saved Frame Pointer

Stack Pointer %esp

Frame Pointer %ebp

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
More Useful Stack for Attacker

Buffer[64] contains shellcode

Stack Pointer %esp
Frame Pointer %ebp
Saved Frame Pointer
Top of Stack
Top of Memory
Note similarity to System()

• Both cases it’s channel mixing
  – “;” mixed with commands in shell language
  – Instruction pointers mixed with data
• Mixing control and data is frequently useful
  – But usually dangerous
Additional Readings

• Cowan et al: *StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks*

• Shacham et al *On the Effectiveness of Address-Space Randomization*

• Hovav Shacham *The Geometry of Innocent Flesh on the Bone*
  – http://cseweb.ucsd.edu/~hovav/dist/geometry.pdf