Program Security

COMS W4115

Prof. Stephen A. Edwards
Fall 2005
Columbia University
Department of Computer Science
Original by Prof. Angelos Keromytis

Why Security?

Many, many more computers and users than ten years ago.

Internet changes the whole game: easy to be attacked from the other side of the planet.

Military-style security of permission levels, groups, etc. worked for closed mainframes, imperfect for networked computers.

Trojan Horses

Leave a program running that looks like the login prompt:

Exploits

At least 40/week on BUGTRAQ mailing list

phrack.com is a how-to guide

Construct-your-own exploit wizards? Point-and-click to breach security?

Code Vulnerabilities

Protocols and algorithms often good; implementations rarely are

Source is bad code with subtle bugs:

Buffer overflows

Race conditions

SQL injection

What's Wrong With This Code?

```
int main(int argc, char **argv) {
   char fname[]= "/tmp/testfile";
   char buffer[16];
   u_long distance;
   distance= (u_long)fname - (u_long)buffer;
   printf("fname = %p\nbuffer = %p\n
           distance = 0x%x bytes\n",
           fname, buffer, distance);
   printf("fname = %s\n",fname);
   strcpy(buffer, argv[1]);
   printf(fname = %s\n",fname);
   return 0;
}
```

Stack Smashing Scenario

PC	foo()'s
buf[16]	activation
	record
PC	strcpy()'s
	activation
	record

```
void foo(char *mystring) {
  char buf[16];
  strcpy( buf, mystring );
}
```

Heap Smashing

Similar trick works when buffer is on heap and something on the heap is treated as a function pointer.

e.g., a C++ virtual table pointer

```
class C {
  virtual void foo();
}

void foo(char * mybuffer) {
   C c = new C();
   char *buf = new char[10];
   strcpy(buf, mybuffer);
   c->foo();
}
```

SQL code injection

SQL queries typically built as a string and passed to the database

```
String query =
  "select * from mysql.user where username='" +
   uid + "' and password=password('" + pwd + "');";
mydb.submit(query);
```

But what if the uid the user supplied is malicious? Could get the query

```
select * from mysql.user where username='' or 1=1; /*
    ' and password=password('whatever');
```

Nice way to get, say, all the credit card numbers from an e-commerce site.

Race Conditions

The Unix filesystem is a shared resource. Things can change from when you check something to when you use it. Example from the "passwd" program:

Pick a "random" filename

If the file already exists in /tmp, try again

Open the file

Copy the contents of /etc/passwd to the /tmp file

Remove existing entry; add the new one

Copy the /tmp file to /etc/passwd

Solutions?

Things to ask about any solution:

How much does it slow things down?

How effective is it against existing and future attacks?

How much of a pain is it to use?

Solutions: Code Signing

Somebody you trust "signs" the code cryptographically

You check signature before you install and run it

Works so-so: is it practical to check everything?

What if it doesn't come from Microsoft?

Do you really trust Bill Gates?

Widely-used: an obvious first line of defense

Sandboxing: Unix's chroot()

In Unix, everything interesting is part of the filesystem.

So limit what a process can do by changing what filesystem it sees.

```
chroot("/usr/ftp");
/* ... public FTP server ... */
```

FreeBSD's `jail()` system call even stronger

Sandboxing: System Call Monitoring

Monitor pattern of system calls in the OS kernel

If a process deviates from what it's allowed to do, terminate it

Works OK for daemons, but what to do for arbitrary downloaded code?

Who writes these policies?

Java's Security Manager does something like this

Solutions: Static Analysis

Run a complex compiler-like program on source code that identifies all vulnerabilities

Really tricky to do quickly and accurately

Fundamentally undecidable; interesting work is on safe abstractions

An open field of research

Solutions: Dynamic Analysis

Augment buffers with size information

Add checks before all reads and writes

Invasive; difficult to get right

Another approach: Perl's "taint" model: Track which information has come from a potentially malicious user and don't allow it to be used in certain ways.

Solutions: Software Fault Isolation

Augmenting binary

Insert checking code around each load, store, or jump that checks it before it is done

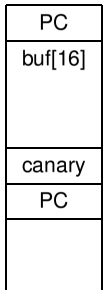
Can be done at compile, link, or run time

Slows things down, bloats code

Really a mess for CISC architectures, which always access memory

Compiler Tricks: StackGuard

Idea: Put a canary on the stack and check if it is still alive before returning.




```
void foo(char *mystring) {
  char buf[16];
  strcpy( buf, mystring );
}
```

Nice defense against stack-smashing, but doesn't do anything for heap smashing, SQL injection, etc.

Solutions: Better APIs

C's string manipulation routines are some of the worst offenders.

```
char *strcpy(char *dest,
             const char *src);

char *strncpy(char *dest,
              const char *src,
              size_t n);
```

Solutions: Better APIs

NAME

tmpnam — create a name for a temporary file

SYNOPSIS

```
char *tmpnam(char *s);
```

DESCRIPTION

The `tmpnam()` function returns a pointer to a string that is a valid filename, and such that a file with this name did not exist at some point in time, so that naïve programmers may think it a suitable name for a temporary file.

BUGS

Never use this function. Use `mkstemp(3)` instead.

Solutions: Better Languages

We're language people, so we should be able to solve everything.

Java, ML, Erlang, etc. aren't subject to buffer overflow attacks (we hope)

Cyclone is a "safe subset" of C (we hope)

CCured: Static analysis of C plus runtime checks for what cannot be determined statically.