

CIS/TCOM 551

Computer and Network Security

Spring 2005

Lecture 3

Announcements

- Project 2 NEW Deadline: Monday, March 14

- Another relevant CIS colloquium
 - Michael Swift (of U. Washington)
 - "Improving the Reliability of Commodity Operating Systems"
 - today at 3:00 in Levine 101

Implementing Secure Software

- Picking a programming language
- Programming idioms that lead to bad/good code
- Tool support
- Beyond C/C++
 - Perl scripting / taint checking
 - Java and C#
- Stack Inspection
 - Example of a security feature build in to Java and C#

Programming Languages

- What reasons are there for a project to choose one programming language over another?
- Class answers:
 - Portability
 - Development time
 - Tool set / maturity of development environment
 - Familiarity of developers (may need training)
 - Language features (e.g. objects or templates)
 - Efficiency
 - Type safety
 - Compatibility (e.g. with legacy code)
 - Maintenance (inheritance / oo-features)
 - Scalability
 - Budget

What about C/C++ ?

- Arguments in their favor:
 - Efficiency
 - Legacy code
 - Low-level data representation (i.e. can interface with hardware)
- Arguments against them:
 - Efficiency is not *that* important.
 - If it is, then it is likely that only 10% of code affects performance significantly (so optimize that).

What's wrong with this code? (1)

- ```
void main(int argc, char **argv) {
 FILE *f = /* create file handle */
 char c = getc(f);
 while (c == '\t' || c = ' ' || c == '\n')
 c = getc(f);
 f(c)
}
```

Need '==' instead of '=' -- this loop doesn't terminate.

# What's wrong with this code? (2)

---

- ```
void main(int argc, char **argv) {  
    char program_name[256];  
    strcpy(program_name, argv[0]);  
    f(program_name);  
}
```

Classic buffer overflow: argv[0] may be too long.

What's wrong with this code? (3)

- ```
void main(int argc, char **argv) {
 char program_name[256];
 strncpy(program_name, argv[0], 256);
 f(program_name);
}
```

String program\_name may not be null terminated.

# What's wrong with this code? (4)

---

- ```
char *copy(char *s) {  
    char buffer[BUF_SIZE];  
    strncpy(buffer, s, BUF_SIZE-1);  
    buffer[BUF_SIZE-1] = '\0';  
    return buffer;  
}
```

This program returns a pointer to *local* memory.

What's wrong with this code? (4)

- ```
char *copy(char *s) {
 char *buffer = (char *)malloc(BUF_SIZE);
 strncpy(buffer, s, BUF_SIZE-1);
 buffer[BUF_SIZE-1] = '\\0';
 return buffer;
}
```

This program is OK... unless buffer is never freed.

# If you must use C/C++

---

- Avoid the (long list of) broken library routines:
  - strcpy, strcat, sprintf, scanf, sscanf, *gets*, read, ...
- Use (but be careful with) the "safer" versions:
  - e.g. strncpy, snprintf, fgets, ...
- *Always* do bounds checks
  - One thing to look for when reviewing/auditing code
- Be careful to manage memory properly
  - Dangling pointers often crash program
  - Deallocate storage (otherwise program will have a memory leak)
  
- Be aware that doing all of this is difficult.

# Tool support for C/C++

---

- Extensions to gcc that do array bounds checking
- Link against "safe" versions of libc (e.g. libsafe)
- Test programs with tools such as Purify or Splint
- Compile programs using tools such as:
  - Stackguard and Pointguard (Cowan et al., [immunix.org](http://immunix.org))
- Research compilers:
  - Ccured (Necula et al.)
  - Cyclone (Morrisett et al.)
- Binary rewriting techniques
  - Software fault isolation (Wahbe et al.)

# Perl

---

- Here is a sample vulnerability:

```
$arg = shift; # get first arg.
system ("echo $arg); # echo to console
```

- What is the problem?

[DEMO of Perl "taint checking" ...]

# Perl Taint Checking

---

- Idea: associate a "tainted" bit with every piece of data.
  - Propagate the bit through computations (e.g. string concatenation)
- Check when any data is passed to a "security-relevant" operation (such as a system call)
  - If tainted bit is set, then give an error.
  - If not, then proceed as usual.
- Programmers can remove the "taint" bit by doing pattern matching to check for validity
  - Not fool proof... Why?