Announcements

- Project 2 has been graded
  - Average: 88

- Project 3 is due Friday
Program Security

• Firewalls are not enough!
  – Some data & code is intentionally permitted through the firewall (mobile code)
  – e.g. Postscript documents are programs
  – e.g. Word documents contain macros
  – e.g. Java applets embedded in web pages.

• Programs have vulnerabilities
  – Exploitable bugs in “good” programs
  – Too many “features” (i.e. automatically execute macros)

• Malicious programs
  – Viruses
  – Worms
  – Trojan Horses
Buffer Overrun in the News

• From Slashdot
  – “There is an unchecked buffer in Microsoft Data Access Components (MDAC) prior to version 2.7, the company said. MDAC is a "ubiquitous" technology used in Internet Explorer and the IIS web server. The buffer can be overrun with a malformed HTTP request, allowing arbitrary code to be executed on the target machine.”
  – http://www.theregister.co.uk/content/55/28215.html
The Consequences

• From Microsoft
  – “An attacker who successfully exploited it could gain complete control over an affected system, thereby gaining the ability to take any action that the legitimate user could take.”
Buffer Overflow Attacks

- > 50% of security incidents reported at CERT are related to buffer overflow attacks

- Problem is access control (i.e. complete mediation) but at a very fine level of granularity

- C and C++ programming languages don’t do array bounds checks
C’s Control Stack

```c
f() {
    g(parameter);
}

g(char *args) {
    int x;
    // more local
    // variables
    ...
}
```

![Diagram of C's control stack showing stack frames and memory allocation.](image-url)
C’s Control Stack

```c
f() {
    g(parameter);
}

g(char *args) {
    int x;
    // more local
    // variables
    ...
}
```

- **f**'s stack frame
- Larger Addresses
- Input parameter
- f’s stack frame
C’s Control Stack

```c
f() {
    g(parameters);
}

// Larger Addresses
int x;  // local variables
// more local variables
...
}
```

Diagram:
- int x;  // local variables
- return address
- base pointer
- Input parameter
- f’s stack frame
Buffer Overflow Example

```c
void g(char *text) {
    char buffer[128];
    strcpy(buffer, text);
}
```
Buffer Overflow Example

g(char *text) {
    char buffer[128];
    strcpy(buffer, text);
}

ADDR:

Attack code
128 bytes

ADDR

base pointer

Attack code
128 bytes

ADDR

f's stack frame

ADDR:

text

Attack code
128 bytes

ADDR

f's stack frame

ADDR:

Attack code
128 bytes

ADDR

base pointer

Attack code
128 bytes

ADDR

f's stack frame

ADDR:

text
The Problem

• C’s `strcpy`, `gets`, `strcat` functions don’t check array bounds

• These functions are used extensively
  – Feb. 25 2002: Internet Explorer HTML tags
    `<EMBED SRC="ATTACK CODE">`
  – telnetd Unix Telnet Daemon
  – Microsoft Outlook
  – Many, many more examples…
Solutions

• Don’t write code in C
  – Use a safe language instead (Java, C#, …)
  – Not always possible (low level programming)
  – Doesn’t solve legacy code problem

• Link C code against safe version of libc
  – May degrade performance unacceptably

• Software fault isolation
  – Instrument executable code to insert checks
  – Pay a performance penalty

• Program analysis techniques
  – Examine program to see whether “tainted” data is used as argument to strcpy.
  – Perl provides these kinds of checks to prevent “format string” vulnerabilities.
  – Possible to eliminate dynamic checks
Software Deployment Architecture

- **Trusted Computing Base**
  - Becomes huge when software is run on many, many hosts

- **Minimunize TCB:**
  - Ensure the quality of the software

- **Must be cheap, easy to deploy**
  - Otherwise won’t be adopted
Existing Approach: Virus Scanners

- Virus Scanners?
  - e.g., McAfee, Norton, etc.
  - perhaps the most commercially effective tool.
  - only works for previously seen bad code.
  - virus kits make it easy to disguise a virus.
  - not clear that it scales over time.

- Not a complete solution
Existing Approach: Signatures

- Digital Signatures of Code?
  - e.g., Verisign, Authenticode, MS device drivers
  - bad assumption: signature implies “good”
    - keys may be stolen
    - “good” for what context?
    - even well-intentioned people make “bad” code
  - bad assumption: you can sue the signer
- Not a complete solution
- Can we do better?
Language-based Security

• Use compiler & programming language technology to improve security.
  • Before the program runs
    – Static type systems (Java/C#)
  • During the program execution
    – Array bounds checks
    – Stack inspection
  • After the program runs
    – Bug report features
    – Auditing

Helps the code producer

Protects the code consumer.
Java Bytecode

• Verify the bytecode at the consumer
• Pro: Simple, cost effective
• Con: Large TCB:
  – commercial, optimizing JIT: 200,000-500,000 LOC
  – when is the last time your favorite software company wrote a bug-free 200,000 line program?
• Con: Java specific policy
Proof Carrying Code

- Verify a *provided* proof of program security
  - Meaning of the proof connected to meaning of program (unlike signatures)
  - Up to code producer to generate proof
  - Consumer only has to *check* the proof
- Verifier is *small*
  - 3000 LOC
PCC: An Analogy

Legend: code
proof
PCC Advantages

• Reduces the TCB
  – Verification is simpler/faster than proof generation.
  – Consumer is independent of how the proof is generated ⇒ compiler not trusted.

• Tamperproof
  – Changing the proof or program is either (1) detected or (2) proven to be OK.

• No cryptography, no trusted 3rd party

• No run-time overhead
  – Static checking