**Security**

CSCI 334  
Stephen Freund

**Principles of Security Design**
- Ensure: secrecy, integrity, availability
- Least Privilege [Saltzer & Schroeder 75]
- Small Trusted Computing Base (TCB)

**Trojan Horse [Ken Thompson]**
- Create compiler binary that:
  A. inserts Trojan horse code to steal passwords when source code for login program is compiled
  B. inserts code to do (A) and (B) whenever source code for compiler is compiled
- Source code for login and compiler have no signs of attack
- Recompiling doesn’t help...

**Morris Worm [Robert Morris, Jr]**
- Program that spreads across Internet and "lives forever"
  - Find new computers on Internet
  - Break into them
  - Copy source code, compile it, start running it, and hide tracks.
- Breaking in:
  - password guessing
  - buffer overrun in fingerd
  - bug in sendmail

**Stack Smashing**

```c
void readName(FILE *socket) {
    char str[512];
    fgets(socket, str);
}
```
- Send name "asd32423098c0sedh1..."
- Overwrite past end of str
- Replace RA with address of code to perform malicious operation

```c
void readName(FILE *socket) {
    char str[512];
    fgets(socket, str);
}
```
- Send name "asd32423098c0sedh1..."
- Overwrite past end of str
- Replace RA with address of code to perform malicious operation
**Context Has Changed**
- TCB Small
- Connections Isolated
- Sys Admins Skilled
- Vendors Few
- Delivery Physical
- Update freq. Seldom
- Update size Whole
- Executables Large Apps
- Sensitive Data Local HD

**Stakes are High**
- Personal:
  - privacy, identity theft, ransom-ware
- Companies/Governments...
  - Sony
  - Anonymous: RIA/MPAA, ISIS, ...
  - PRISM: NSA breaches Google, Facebook, Apple, ...
  - Stuxnet
  - North Korea missile program
  - 2016 US Election, 2018 US Election, 2020 ...
  - Bitcoin, Etherium, ...
- **Williams/CERT attacks?**

**How to Prevent Attacks From Active Content**
- Attach Social Stigma
- Educate Users
- Disable active content, disconnect from Web
- Virus Scanning
- Code Signing

**Virus Scanning**
- Virus
- Known Virus Definition Patterns
- Virus Scanner
- match found
- no match

**Public Key Cryptography for Communication**
- Used in certificates too (ex: X.509)
**Code Signing**

Springer’s private key

Program \( M \) → Encrypt → \( E(M, \text{priv}) \)

(certified program)

\( E(M, \text{priv}) \) → Decrypt → Program \( M' \)

Steve’s Inbox

**Signature Checking**

Springer’s public key

Program \( M \) → Decrypt → \( M' = D(#, \text{pub}) \)

- ✔: Springer signed the code
- ✘: Springer did not sign the code

---

**How to Prevent Attacks**

- Attach Social Stigma
- Educate Users
- Disable active content, disconnect from Web
- Virus Scanning
- Code Signing
- Language-Based Least Privilege

---

**Sandbox Security Model**

Virtual Machine Sandbox

Files → Printer → Monitor → Network

Scoping → Type system

---

**Granting Privileges to Principals**

- Local security policy file: java.policy

```java
grant CodeBase "www.cs.williams.edu" {
  permission java.io.FilePermission "/home/data" "read", "write"
}

grant CodeBase "www.sneaky.com" {
  permission java.io.FilePermission "/tmp" "read", "write"
}
```

---

**Security Manager**

- Methods
  - checkRead
  - checkWrite
  - checkListen
  - checkConnect
  - ...

- Run-time system calls these methods prior to every resource access.
Stack Inspection

Permission depends on:
- calling method (based on principals)
- all methods above it on stack

Two Basic principals:
- SYSTEM
- UNTRUSTED

Stack Inspection

void open(String s) {
  SecurityManager.checkRead();
 ...
}

Stack Inspection (Example 2)

public Font loadFont(String s) {
  setPrivileged();
  ... input.open(s);
  }

Stack Inspection (Example 2)

FileInputStream.open("log")

System.main()
LocalClass.f()
LocalClass.g()
FileInputStream.open("log")

FileInputStream.open("log")

System.main()
LocalClass.f()
LocalClass.g()
FileInputStream.open("log")

FileInputStream.open("log")

System.main()
LocalClass.f()
LocalClass.g()
FileInputStream.open("log")

FileInputStream.open("log")

System.main()
LocalClass.f()
LocalClass.g()
FileInputStream.open("log")

FileInputStream.open("log")

System.main()
LocalClass.f()
LocalClass.g()
FileInputStream.open("log")

FileInputStream.open("log")
Stack Inspection (Example 2)

```
0  System.main()
0  Applet.run()
   showDialog("passwd")
0  System.
   showDialog("passwd")
1  FontManager.
   loadFont("passwd")
0  FileInputStream.
   open("passwd")
```

Type Safety: Verifier

```
Java Program  Compiler

Verify types for bytecodes

NO!  YES!
```

Java Bytecodes

- **Java:**
  ```
  class A extends Object {
   int i;
   void f(int val) { i = val + 1; }
  }
  ```

- **Bytecode:**
  ```java
  Method void f(int)
  0  aload 0
  1  iload 1
  2  iconst 1
  3  iadd
  4  putfield #4 <Field int i>
  5  return
  ```

Java Bytecodes

- **Java:**
  ```
  class A extends Object {
   int i;
   void f(int val) { i = val + 1; }
  }
  ```

- **Bytecode:**
  ```java
  Method void f(int)
  0  aload 0
  1  iload 1
  2  iconst 1
  3  iadd
  4  putfield #4 <Field int i>
  5  return
  ```

Proof-Carrying Code

```
Java Program  Compiler

Bytecode Program

safety proof

proof checker

NO!  YES!
```

Java Bytecodes

```
Method void f(int)  "Proof:"
  0  aload 0  S=[]  R=A  R=int
  1  iload 1  S=A:[]  R=A  R=int
  2  iconst 1  S=A:[]  R=A  R=int
  3  iadd  S=A:[]  R=A  R=int
  4  putfield #4 <Field int i>  S=A:[]  R=A  R=int
  5  return  S=[]  R=A  R=int
```
**Post-Mortem**

- Audit logs
- Look for anomalous behavior
- Reconstruct failures, prevent in future

- **OpenSSL attack (2014?)**
  - buffer overr
  - divulged information from server
  - no audit log --- can’t tell who was actually hacked...