Security (2)
CSCI 334
Stephen Freund

Namespace Management

www.airline.com
VisaCreditCard.class
new VisaCreditCard()

www.evil-site.com
Sneaky.class
new Sneaky()

Multiple ClassLoaders

www.airline.com
VisaCreditCard.class
new VisaCreditCard()

www.evil-site.com
Sneaky.class
new Sneaky()

ClassLoader
class NetworkClassLoader extends ClassLoader {
    private String host;
    private Hashtable<String, Class> loaded;
    public Class loadClass(String name) {
        Class c = loaded.get(name);
        if (c == null) {
            byte b[] = readFileFromNetwork(host + name);
            c = defineClass(b, 0, b.length);
            loaded.put(name, c);
        }
        return c;
    }
}

Sandbox Security Model (Incomplete)

Class Loader
class NetworkClassLoader extends ClassLoader {
    private String host;
    private Hashtable<String, Class> loaded;
    public Class loadClass(String name) {
        Class c = loaded.get(name);
        if (c == null) {
            byte b[] = readFileFromNetwork(host + name);
            c = defineClass(b, 0, b.length);
            loaded.put(name, c);
        }
        return c;
    }
}
Enforcing Sandbox Boundaries

• **ClassLoader** prevents interference between applets

• **Bytecode verifier** prevents direct access to resources
  - classify "unsafe" operations as type errors
  - don’t run programs with type errors
  - example:
    * `String s = "hello";`
    * `s = s - 3;`  ⇐ BAD
  - another example:
    * `byte b[] = { 0x12, 0xa3, 0x05, ... };`
    * `((Function)b)();`  ⇐ REALLY BAD

Verifier

Java Bytecodes

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

• Bytecode:
  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:
  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 Bytecodes

"Proof:"

Method void f(int)
0 aload 0 S=1:[] R0=A R1=int
1 iload 1 S=A:[] R0=A R1=int
2 iconst 1 S=int:A:[] R0=A R1=int
3 iadd S=int:int:A:[] R0=A R1=int
4 putfield #4 <Field int i> S=int:A:[] R0=A R1=int
5 return S=1[] R0=A R1=int
**Sandbox Security Model**

- Virtual Machine Sandbox
- Security Manager
- Security Policy
- Files
- Printer
- Monitor
- Runtime Library
- Network

**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
  - checkCreateClassLoader
  - checkExec
  - ...

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

**Stack Inspection**

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

```java
void open(String s) {
    SecurityManager.checkRead();
    ...
}
```
Stack Inspection

- Permission depends on:
  - permission of calling method (based on principals)
  - permission of all methods above it on stack
- Two Basic principals:
  - SYSTEM
  - UNTRUSTED

Stack Inspection

- Permission depends on:
  - permission of calling method (based on principals)
  - permission of all methods above it on stack
- Two Basic principals:
  - SYSTEM
  - UNTRUSTED

Stack Inspection (Example 2)

System.main()
Applet.run()
  showDialog("Times")
FontManager.loadFont("Times")
FileInputStream.open("Times")

Stack Inspection (Example 2)

System.main()
Applet.run()
  showDialog("Times")
FontManager.loadFont("Times")
FileInputStream.open("Times")

Stack Inspection (Example 2)

0  System.main()
1  Applet.run()
  showDialog("Times")
2  System.showDialog("Times")
3  FontManager.loadFont("Times")
4  FileInputStream.open("Times")

Stack Inspection (Example 2)

0  System.main()
1  Applet.run()
  showDialog("Times")
2  System.showDialog("Times")
3  FontManager.loadFont("Times")
4  FileInputStream.open("Times")
Audit Logs

Program 1
Program 2
OS
Network

Log Analyzer

Security Problems

Programming Languages

• PL defines programming model
  - discuss concepts, formulate algorithms, and describe computation in model
  - every language has a different model

• Different PLs support different ways of thinking
  - functional, imperative, OOP, concurrent, distributed, ...

• PLs change constantly
  - new technology
  - new domains

PL Tool Box

• Use languages effectively

• Learn new languages

• Critically compare different languages
  - PL design is full of trade-offs
  - The right language can make a problem easy.
  - The wrong language can make a problem hard.

• Create new domain-specific languages, systems, APIs