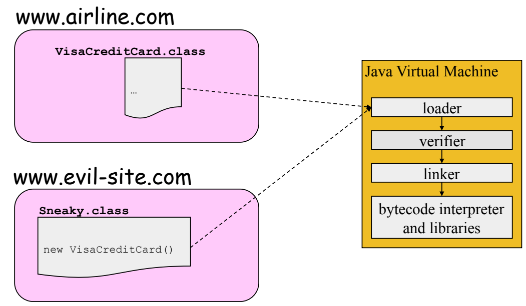
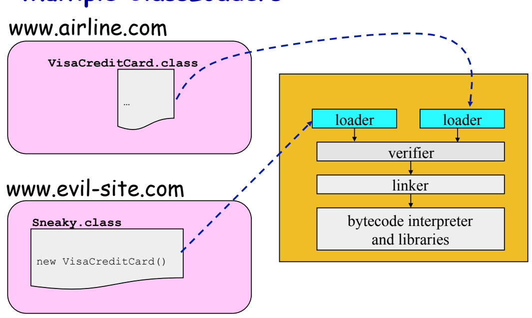
Security (2)

CSCI 334
Stephen Freund

NameSpace Management



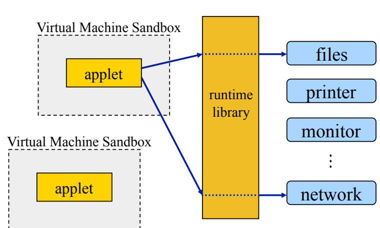
Multiple ClassLoaders



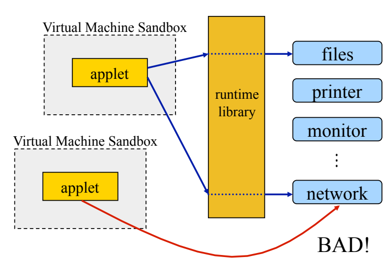
Class Loader

```
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;
  }
}
```

Sandbox Security Model (Incomplete)



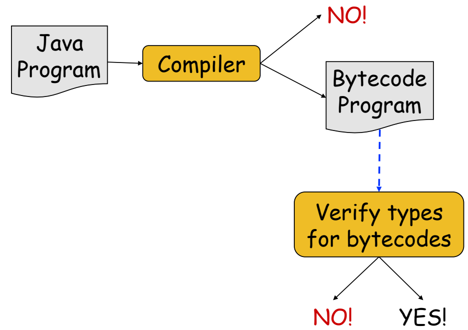
Sandbox Security Model (Incomplete)



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
```

Does stack top have two integers?

Does field i exist for object on stack?

Java Bytecodes

- Java:

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

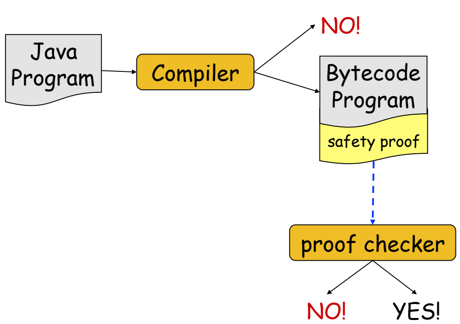
- Bytecode:

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

S_3=int::int::α
S_4=int::α

S_4=int::A::β
A has field "int i"
S_5=β

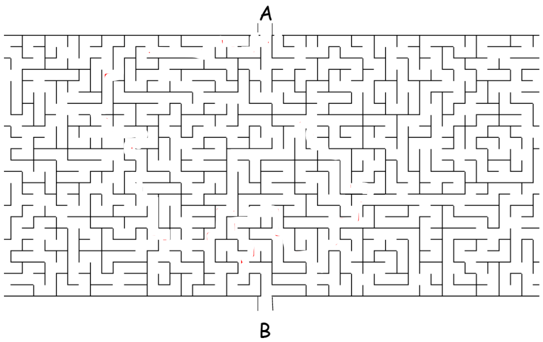
Proof-Carrying Code

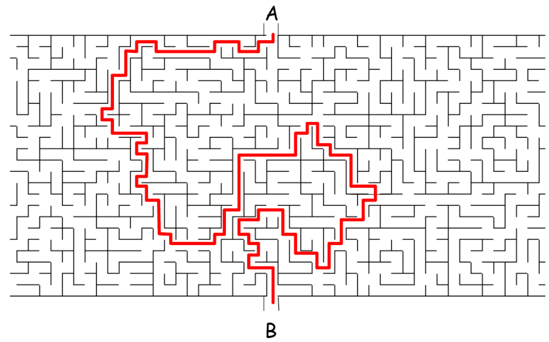


Java Bytecodes

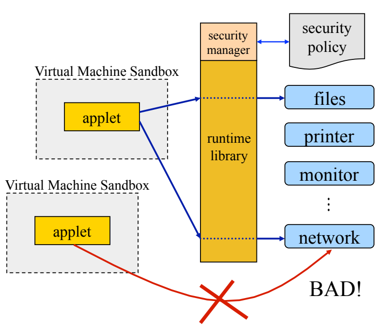
"Proof:"

Method void f(int)			
0 aload 0	S=[]	R_0=A	R_1=int
1 iload 1	S=A::[]	R_0=A	R_1=int
2 iconst 1	S=int::A::[]	R_0=A	R_1=int
3 iadd	S=int::int::A::[]	R_0=A	R_1=int
4 putfield #4 <Field int i>	S=int::A::[]	R_0=A	R_1=int
5 return	S=[]	R_0=A	R_1=int





Sandbox Security Model



Granting Privileges to Principals

- Local security policy file: **java.policy**

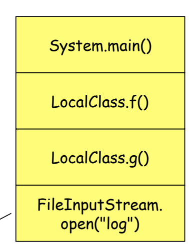
```
grant CodeBase "www.cs.williams.edu" {
    permission java.io.FilePermission
               "/home/data"
               "read", "write"
}

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




```
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

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

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

System.main()
SomeApplet.run()
SomeApplet.sneaky()
FileInputStream. open("/etc/passwd")

Stack Inspection (Example 2)

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

Stack Inspection (Example 2)

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

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

Stack Inspection (Example 2)

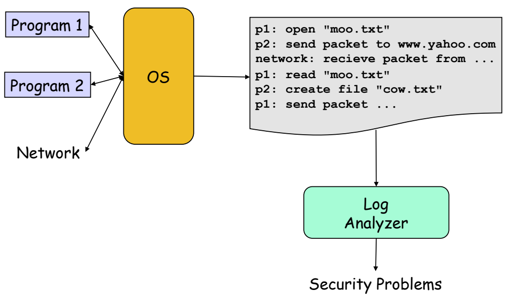
0	System.main()
0	Applet.run() { showDialog("Times") }
0	System. showDialog("Times")
1	FontManager. loadFont("Times")
0	FileInputStream. open("Times")

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

Stack Inspection (Example 2)

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

Audit Logs



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***