Security

"One of the most significant aspects of Java programming is that it creates applications that have extraordinary relevance to computer security. Few UNIX administrators would be prepared to allow millions of users to execute programs as root (the administrative superuser) on their system, yet this level of potentially total power is what every user cedes when they point their browser at a URL containing some form of Java executable."

Security

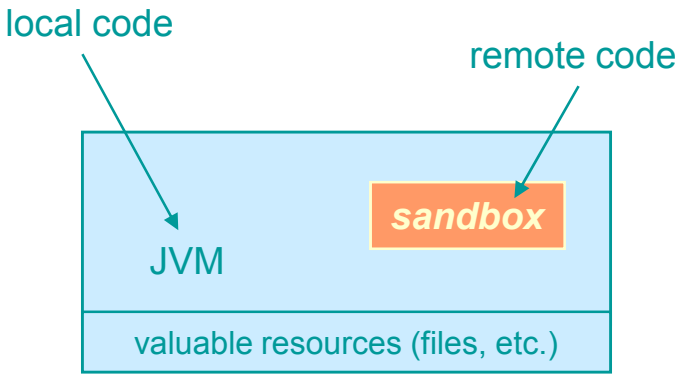
- Much ballyhoo
  - important issue
    - *“Security measures are an integral part of Java’s design.”*
  - handled at several levels
    - language
      - no pointers, bounded arrays, GC, etc.
    - implementation
      - available for ‘public’ scrutiny
      - bugs & issues acknowledged quickly and responsibly
        - so far!?
    - policy
      - applet security restrictions

Security

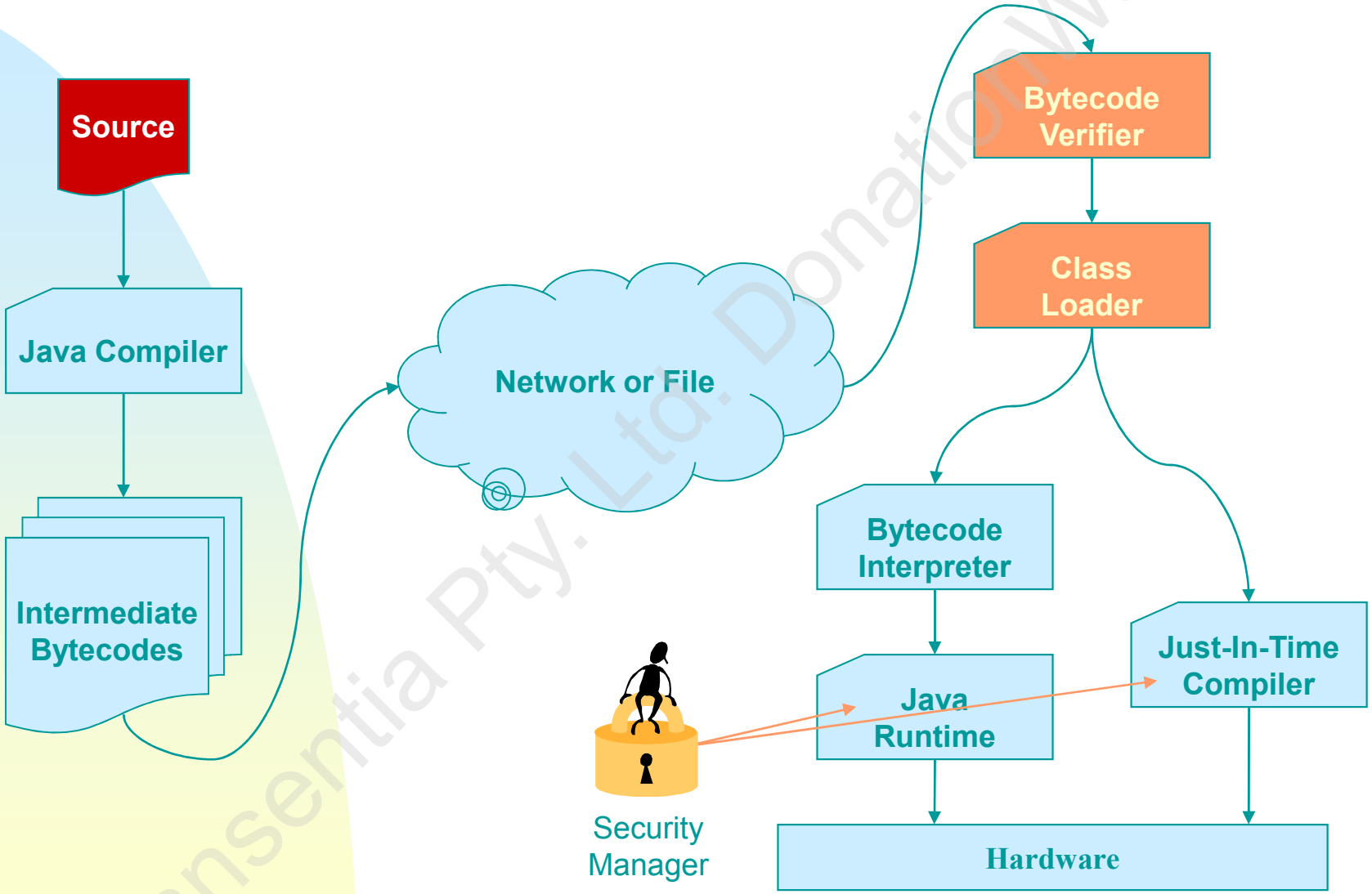
- Delicate issue
  - *"The trouble with applets is that security restrictions are getting tighter and tighter, while what you can do is becoming less and less significant. Every time…another hole, the noose tightens."*
  - *"…stuck in a sandbox."*
  - *"Java is becoming restricted to creating executable window dressing, not (useful) executable content."*

Security

- The sandbox
  - allows untrusted code to execute in a trusted environment
    - untrusted code is code not on the CLASSPATH
  - 3 aspects:
    - bytecode verifier
    - applet class loader
    - SecurityManager



Security

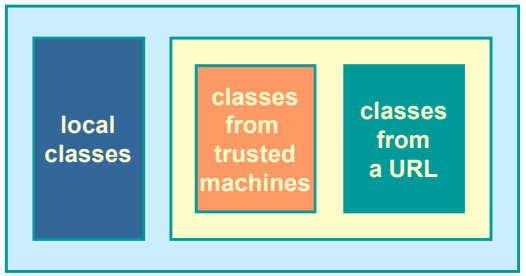


Security

- Bytecode verifier
  - verification takes place before execution
    - ☞ *all* untrusted code is verified
      - valid JVM class
        - may not come from javac!
      - simple analysis:
        - all operations leave stack in a correct state
        - registers are used correctly
        - data types are not subject to illegal conversions
        - all bytecodes are legal
        - all class member accesses obey the access specs
    - ☞ *"establishes a base level of security guarantee"*
      - possible since Java is inherently 'quite' secure

Security

- class loader
  - *"…the first line of defence in the Java security model."*
  - determines when & how code can add classes to the runtime environment
    - ensure that core parts are not overwritten
  - may be multiple class-loaders running concurrently
    - ensures strong separation of namespaces
      - each namespace may have its own class loader



Security

- SecurityManager class
  - abstract class that collects all the policy decisions that the run-time system must make
    - access to filesystem, network connectivity, thread integrity, operating system resources, etc.
  - complete control over a well-defined set of ‘dangerous’ activities
  - decision may depend on origin of code:
    - {local, downloaded} applet
    - application
    - code on CLASSPATH

Security

- methods, etc.
  - SecurityException
  - checkPackage{Access, Definition} ()
  - check{Read, Write} ()
  - check{Listen, Connect} ()
  - checkExit ()
  - getSecurityContext ()
  - checkExec ()
  - checkTopLevelWindow ()
  - etc.

Security

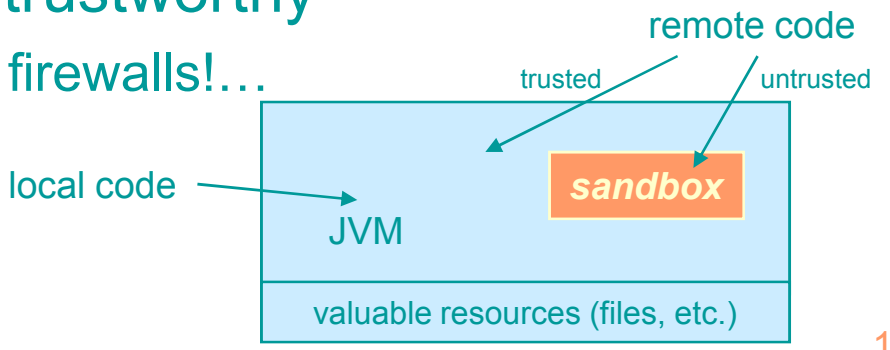
```
// from FileoutputStream.java--actual source
public FileOutputStream (String name) throws IOException
  {
  SecurityManager security = System.getSecurityManager ();
  if (security != null)
    {
    security.checkWrite (name);
    }
  try
    {
    fd = new FileDescriptor ();
    open (name);
    }
  catch (IOException e)
    {
    throw new FileNotFoundException (name);
    }
  }
```

Security

- Hostile applets still possible
  - resource wastage
    - CPU, memory
  - grim reaper
    - kills any other applets that may run
  - forgery
    - mail, port 25; telnet, port 23
  - denial of service
    - tsunami

Security

- Code signing
  - premise: shouldn't run untrusted code, so establish trustworthiness of code
    - Microsoft's strategy for ActiveX
  - reduced limitations on behavior
    - applet allowed to step outside the sandbox
  - binaries digitally signed as 'guarantee' of quality
    - signature irrevocable and unforgeable
    - signatory must be trustworthy
      - may be OK within firewalls!…



Security

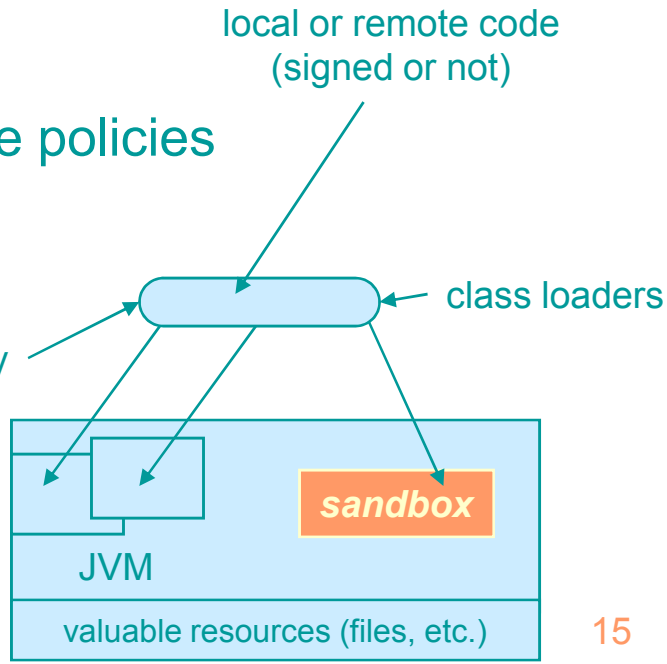
- Digital signatures
  - a sequence of bytes embedded in the code
  - placed by the originator
  - unforgeable—allows users/contexts to:
    - identify object signers
    - detect tampering
    - examined to determine what the Java code wants to do so that a decision can be made about the resources that can be allocated to it
  - recall that much corporate crime is perpetrated by long-time, trusted employees…
    - so do we trust 'trust'…?

Security

- *"Security is an evolutionary process"*
  - probably a few flaws in design
  - may (will?) be many in implementation
    - Sun professes *"zero tolerance for security bugs"*
  - relies on trust
    - sensationalist media could destroy that trust
  - *"A surprisingly large portion of the entire infrastructure must be trustworthy, including pieces you might not have realized were critical."*

Security

- Security features still evolving
  - becoming *lots* more sophisticated
    - 1.0: simple sandbox model
      - namespaces via class loaders/code verification
    - 1.1: code signing and digital signatures
      - verify origin and establish trust
    - 1.2: policy-driven security
      - access control lists & definable policies
        - load-time mechanisms
      - no built-in ideas of trust
      - for *all* Java code
        - applets, servlets, apps
        - etc.



Security

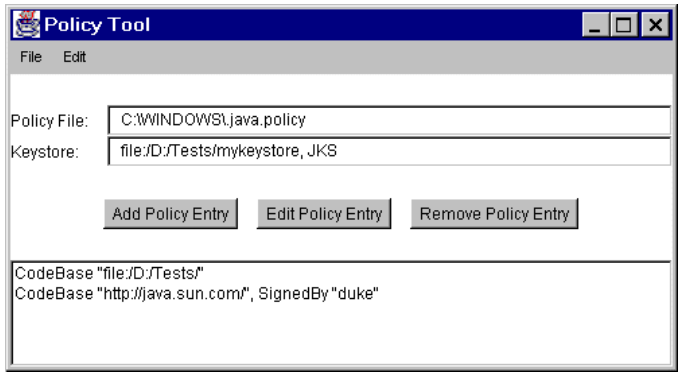
- Policies
  - specified at runtime

```
> java -Djava.security.policy=file://.my.default.policy MyJavaApp hello world
```

  - policy files

```
// this file: .my.default.policy
// if the code is signed by "Duke", grant it read/write access to all
// files in /tmp:
grant signedBy "Duke"
  {
  permission java.io.FilePermission "/tmp/*", "read,write";
  };
// Grant everyone the standard permissions:
grant
  {
  permission java.util.PropertyPermission "java.vendor";
  };
```

  - policytool
    - for editing policies



Security

- Privileged blocks
  - *“enabling a piece of trusted code to temporarily enable access to more resources than are available directly to the code that called it”*
    - e.g. access to installed fonts: normal code must become (controllably) privileged while obtaining the current platform’s installed fonts

```
void doSomethingPrivileged (final String param)
  {
  Object result = AccessController.doPrivileged
                    (
                    new PrivilegedAction ( )  // an interface
                      {
                      public Object run ( )
                        {
                        // privileged code goes here, for example:
                        return doWhatever (param);
                        }
                      }
                    );

  }
```