Java — Threat or Menace

Steven M. Bellovin

smb@research.att.com

908-582-5886

AT&T Labs Research

Murray Hill, NJ 07974

I drank half a cup, burned my mouth, and spat out grounds. Coffee comes in five descending stages: Coffee, Java, Jamoke, Joe, and Carbon Remover. This stuff was no better than grade four.

Glory Road,
Robert A. Heinlein

What Java Does

- An outsider supplies code, to be executed by you, on your machine.
- A variety of mechanisms attempt to contain the execution.
- Do these mechanisms actually contain Java?
- Is the containment sufficient in theory? Can it be?

What We Want

- “Absolute” security.
- “Do what we mean” level of protection.
- Dancing dinosaurs on our screens. . .

Can We Have it All?

- Most features of Java are necessary for some legitimate functions.
- Individually, these may each be safe; collectively, they present dangers.
- The functionality we need for legitimate uses is precisely what can be exploited to cause mischief.

Examples

- File I/O necessary to keep state locally—but can be abused if the applet can read sensitive files.
- Network I/O necessary to let the applet act as the user's agent—but can be used to attack other machines.
- Screen I/O necessary for graphics—but can be used for denial-of-service attacks.

Java Security Model

- Some network I/O is permitted, but only to the originating site.
- Some file I/O is permitted, but only to certain files or directories.
- Protection is orthogonal to the operating system's mechanisms.

Implementation of the Security Model

- Access to primitives defined in terms of Java source language restrictions (inheritance, name space, etc.)
- But browsers see “byte code”, not Java source.
- Complex software (the byte code verifier and the class loader) attempts to verify that the byte code represents a legal Java program.
- I claim that this model is too complex to ever be trustable.

☞ The AppletSecurity system is about 500 lines of code; the byte-code verifier is seven times larger.

Virtual Machine Operations

- Virtual machine not designed for verification.
- Theorem-prover used for type safety and the like.
- Access is via ordinary `invoke` instruction.
- Restriction is based on Java language semantics.
- Contrast this with Ghostscript's SAFER mode, where the dangerous operations are *deleted* from the language.

Protection Model for Real Hardware

- R/W/X bits per page.
- Virtual memory makes some areas invisible.
- Supervisor state allows for privileged operations.
- Transition to supervisor state only via special operation

☞The basic checking operations are much simpler.

Network I/O

- Java relies on the DNS for enforcement.
  - DNS games have been used to compromise the security model.
  - DNS queries leak information by going *around* the security model.
- Java can be used to attack firewalls from the inside. . .

Attacking Firewalls

- Call the FTP server on the originating host (a legal Java operation).
- Issue a `PORT` command specifying the `telnet` port (or worse).
- (Many) dynamic packet filters will obligingly open up a channel to that port.
- Variants on this attack (ab)use `socks`, upload encoding, etc.

☞ Security mechanisms don't always compose cleanly — each was built on its own input/output model, and the assumptions sometimes clash.

Denial of Service Attacks

- Not covered by the Java security model.
- Applets can use up resources—CPU, memory, screen I/O (which upsets your window manager), sound (which upsets you), etc.
- Defending against any denial of service attack is hard; defending against these may be harder.

Digitally Signed Applets

- In theory, signed applets are no worse than store-bought programs.
- In the environment of the Web, it doesn't scale; there are too many requests for signatures.
- You don't know what you're "buying" — a search engine, an animated ad, or a Trojan horse.
- Web servers are more susceptible to attack than are a vendor's development machines — did the site observe proper security on its digital signature key?
- The attacker knows the buyer, and can tailor attacks on the Web.
- Did the user validate the certificate hierarchy properly? What about `microsoft.com` versus `MICROSOFT.COM`? Should it be `nasa.gov` or `nasa.com`?

Structural Security

- Security has to be based on small, simple, easy-to-understand primitives.
- The security kernel should irrevocably shed permissions and abilities. Suppose, for example, that every applet listed, up front, the name of each file it wanted to read or write. Internal methods never do real `open` calls; they just use file descriptors.
- Minimize the amout of security-sensitive code.
- The AppletSecurity system is about 500 lines of code; the byte-code verifier is seven times larger.

Orange Book — C1/C2 Definition

> The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system. **The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.**

The text in boldface was added for C2.

In an Ideal World. . .

- We'd have an operating system with the right primitives for secure execution of any untrusted application.
- Failing that, we should tighten up the Java execution environment.
  - + Add local administrative controls on Java's permissions.
  - + Eliminate reliance on the byte-code verifier.
  - + Put the security checking into the native methods.

Rationale

The intended state is a security model that is a reasonable approximation to what a conventional operating system and machine would provide to a user process. To the extent that Java has unique needs, either more or less restrictive, these should be modeled as different user permission sets or capabilities. I say this not because I think that “conventional” systems are perfect; rather, I'm looking for strong assurance of security. (How many security holes are the fault of the kernel?)