Java Security Model

Steven M. Bellovin

`smb@research.att.com`

908-582-5886

AT&T 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

The Language Model of Security

- Security features are based on the Java language model.
- The ability of a program to execute a particular native method depends on its ability to utter the method's name.
- But the features of the Java source language—type safety, lack of pointers, etc.—are all but irrelevant, except as they allow for a simpler virtual machine.

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

Virtual Machine Operations

- 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.

Poorly-Defined Semantics

- What is "the originating host"?
- DNS interactions, including signalling via the DNS channel.
- Conflict between desired abilities, such as file and network I/O. The safety of any of these operations depends on the program's history and context, and is not easily checked statically.

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.

Adminstrative Issues

- Difficult to set up a site-wide security policy. This is partly inherent in the nature of the technology—anyone can install any Java-capable browser they wish—and partly in the nature of the configuration files and/or firewall abilities. That is, why should it read `~/.hotjava/properties` rather than `/usr/lib/hotjava/properties` for security stuff?
- `CLASSPATH` should not have the current directory always first.

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?)