Java Policies

One−size−fits−all policies are fundamentally limited

- ‘‘Intranet’’ apps don’t need restrictions
- Interesting classes of applications can’t be written

To break the system, you’ll try to change your security label

- Higher assurance necessary *before* adding new features

Get A Good Policy

***Java Security FAQ* isn't a specification**

***Step 1:* Discuss Java with normal security terminology (subjects, objects, channels)**

***Step 2:* Formally specify access rules [Lampson et al. 92, Wobber et al. 94]**

Build the theorem prover into object method invocation

Applet digital signatures fit naturally

Policy Specification

Trust groups (internal, external−friend, ...)

From digital signature or ‘‘code source’’

Application profiles / privilege subsets

Authorized by user or trusted digital signature

Problem:

User input + signatures + policy rules == complex relationships

UI Design

Guiding principle: *don't bother the user*

Limited filesystem access [Karger 87]

`openLogFile()` − unique file for applet to save preference data
`openUserFile()` − user gets dialog box, applet gets open file handle

Other places for controlled access

clipboard
microphone
speaker
video camera
root−level windows

Preventing denial of service

Thread lister / killer
CPU meter

User endorsement

applets − one version or any version?
vendors − blanket trust or per−product?
CA's − local, national, arbitrary?

Company−wide

Assume a given user machine *will* be compromised

WWW proxy server == security server

- Centralize trust − one machine [Kerberos]
- Save all ‘‘.class’’ files
  - Reconstruct an attack afterward
- Network logging facility
- Centralized policy decisions
  - Server can say who to trust, who not to trust

Guiding principle: *don’t bother the user*

Putting It Together

Java implementation

higher assurance before flexible security

Java policies

formal specifications
run−time support

WWW proxy server

add logging / authorization services

UI design

no security without usability