Alternatives to Comprehensive Least-Privilege

Karl MacMillan <kmacmillan@tresys.com>
Background and Motivation

• All large SELinux policies are least-privilege
  – Fine-grained types, attributes, object classes, and perms
  – Minimal use of equivalence classes
  – Tend to evolve towards more least-privilege over time

• Several reasons why:
  – Offers excellent security
  – Conceptually clear – requires less risk analysis

• Several drawbacks around size and complexity:
  – Examining some security properties requires analysis
  – Large policy customization require engineering
  – Policy is sometimes brittle in the face of system change
  – Disk and memory footprint large for some systems
Some Observations

• Reference policy has improved situation . . .
  – But end of improvements from engineering may be near
  – Fundamental simplifications are desirable
• Some applications are difficult to constrain
  – Is it really possible to effectively constrain HAL, udev, etc.?
• Limiting how domains interact can be uninteresting
  – Is the mechanism of IPC between domains important?
  – Perhaps we just care about read / write between domains?
• Fine-grained types often just for later customization
  – Work-around for 'type splitting' problem
• Users often request other security goals
  – Example: just remove network access from user shells
  – Implementation difficult because of policy size
Suggestion 1: Exploit Equivalence

• Current policy mirrors application / file structure
  – Similar applications are given separate types
  – Many policies are largely similar

• Collapse similar types
  – Into a fewer, more generic types
  – Examples: small trusted base, package managers, etc.

• Fewer types results in fewer interactions
  – Reduces allow rules, interfaces, templates, etc.
  – Simplified testing

• Potential problems:
  – Hampers future customization - “type splitting problem”
  – Care required to avoid overly broad equivalence
Suggestion 2: Reduce Objects / Perm

• Reduce the number of object classes and perms
  – Remove unneeded granularity in object classes
    • e.g., have a single IPC object class
  – Make permissions more consistent across classes
    • read, write, open, create, delete, append, execute
  – May need to retain 'inline assembler' for raw access

• Rely more on types to differentiate access

• Potential drawbacks:
  – Inconsistent objects / perms in kernel denials
  – Tool changes (audit2allow) can help
  – Policies for different use cases may diverge at object level
Other Suggestions

• Experiment with focus on other security goals
  – E.g., application integrity, separation, confidentiality
  – Allow broader access by default according to goals
  – Ideally provide several alternatives for a single application

• Analyze security threats and policy effectiveness
  – May lead to alternative approaches
  – Enables balancing of complexity and security benefit

• Explore language features to ease customization
  – Much policy complexity is to enable later customization
  – Current policy aims to be all things to all users
    • Often to work around language shortcomings
  – Other talks today on this subject
Approach

• Emerging policy tools will allow experimentation
  – Language features for easier customization
  – Object / perm reduction can be done by policy tools
  – Some tools exist today: e.g., CDSFramework

• New SELinux-enabled platforms offer opportunities
  – Embedded devices in particular
  – Appliances (virtual or real) offer narrowly focused goals
  – Also OpenSolaris and Ubuntu

• Ideally successful experiments will be upstreamed
  – Both userland tools and Reference Policy
  – However, short term divergence is healthy
Questions / Discussion

Karl MacMillan <kmacmillan@tresys.com>