SELinux Policy Language
Research

James Carter
jwcart2@tycho.nsa.gov
National Security Agency
National Information Assurance Research Laboratory (NIARL)
Motivation

- Desire to experiment with new language features.
  - Type inheritance
  - Cloning types
  - Removing rules
  - Deny rules

- Problem – Adding features to the current policy toolchain is neither easy nor enjoyable
  - Complex (M4, awk, Python, flex, bison, ???)
  - Slow (150 seconds or more to compile)
A Slippery Slope

- Started just to create a simple policy language
  - Just create a valid policy.conf
  - Use checkpolicy to make a binary policy
- But needed a somewhat working policy
  - Try to translate parts of Refpolicy
- Ended with just parsing Refpolicy
  - Not as bad as one would think
  - 1200 lines of Lua
Parsing Problems

- Can't parse misc_macros.spt
  - Too much M4isms
  - Must create own versions of:
    - can_exec, gen_user, gen_context, gen_bool
- Auto-generated macros need to be added
  - all_dbus_perms, all_nscd_perms, all_passwd_perms
- Interface calls can be a challenge
  - Problems when arguments not treated as just strings
    - ($1,$2-shadow_t) - Is this a set or an MLS label?
    - $2 is null => {{}}-shadow_t} => {}
Not a replacement for the current toolchain

- Can produce a policy.conf from Refpolicy
  - Duplicate rules are removed
    - Results in a smaller binary policy
- Does not do MLS yet
  - Just need to handle user rules and verify the mls label
- Does not do binary modules
- None of the semanage functionality
- Still verifying that it produces the same policy as the regular toolchain.
Some Experimentation

- Auto generation of attributes
  - 4,000 attributes
    - Policy not smaller, checkpolicy dies
  - 200 attributes
    - ~20% smaller binary policy
- clone_type rule
  - Just started working on it
- remove rule
Thoughts

- Policy tree is useless
  - Only need it to evaluate optional blocks
    - Optional blocks are painful
- We need to make it easier to build higher level policies.
- We have a policy assembly language already.
- Separate the tool chain
  - High level language compiler
  - Low level language compiler
One Possible Way

Policy -> High-level Compiler -> managed files -> Low-level Compiler -> Binary Policy

Policy Management

Customizations
Low-level compiler

- Could be checkpolicy
- Could be much simpler
  - No set operations
  - No aliases
  - No optional blocks
  - Avoid ambiguous statements
    - allow, class
  - End all statements with a “;”
High-level compiler

• Could be
  – Refpolicy
  – Some crazy XAMCL-based thing

• One high-level computer language does not meet the needs of every programmer.

• Likewise, one high-level policy language is not going to meet the needs of everyone writing security policy.
Questions?