CDSFramework: Experiences implementing a high-level policy language for SELinux

2006 SELinux Symposium

Chad Sellers, James Athey, Spencer Shimko,
Art Wilson, Frank Mayer, Karl MacMillan

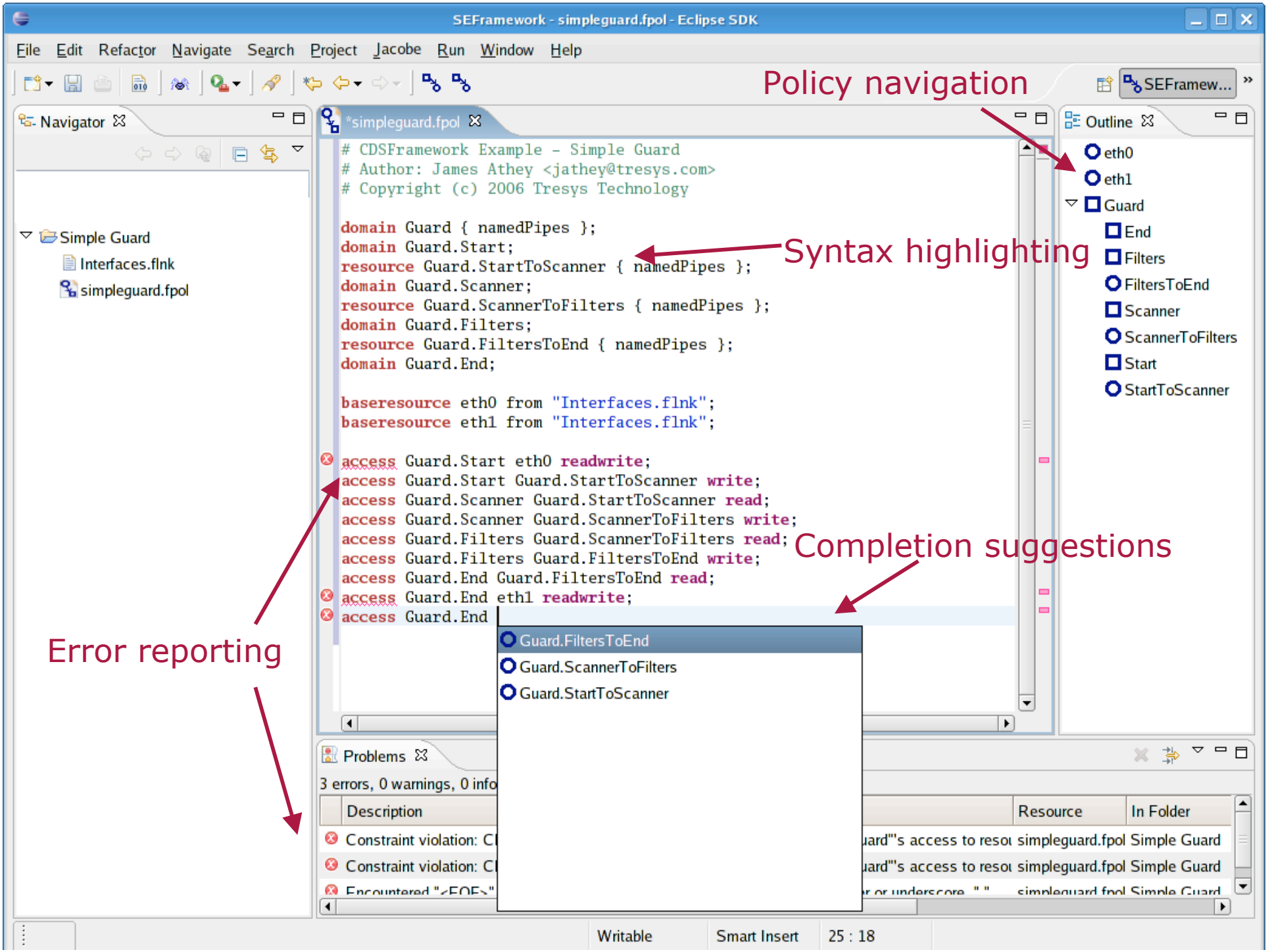
Tresys Technology, USA

Writing Policy in a High-Level Language

- SELinux provides MAC foundation
  - Flexible
  - Steep learning curve
  - Policy language directly describes resulting policy
- A High-Level Language opens up new possibilities
  - Easily represent different security paradigms
  - New users
    - Uses abstractions to cover SELinux details where possible
    - Uses terms accessible and familiar to other audiences
  - New language features
    - enable smarter compilers, policy analysis, toolkits

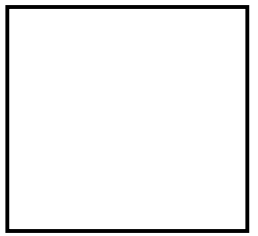
The CDSFramework Language and Tools

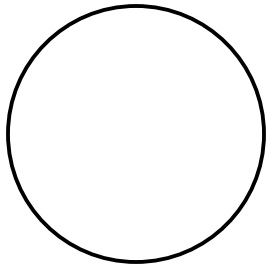
- High-level policy language
  - Policy described in terms of information flow
  - Developed with Cross-Domain Solutions in mind
  - Meant to be accessible to application developers
- Compiler and IDE
  - Provide developer-friendly features
    - e.g, Completion suggestions, informative errors
- Introduced as the SEFramework at 2005 SELinux Symposium

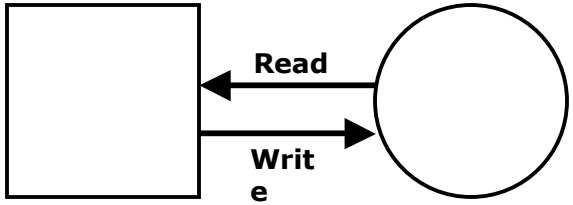

SEFramework - simpleguard.fpol - Eclipse SDK
File Edit Refactor Navigate Search Project Jacobe Run Window Help
Policy navigation
Navigator
Simple Guard
Interfaces.flnk
simpleguard.fpol
*simpleguard.fpol
CDSFramework Example - Simple Guard
Author: James Athey <jathey@tresys.com>
Copyright (c) 2006 Tresys Technology
domain Guard { namedPipes };
domain Guard.Start;
resource Guard.StartToScanner { namedPipes };
domain Guard.Scanner;
resource Guard.ScannerToFilters { namedPipes };
domain Guard.Filters;
resource Guard.FiltersToEnd { namedPipes };
domain Guard.End;
baseresource eth0 from "Interfaces.flnk";
baseresource eth1 from "Interfaces.flnk";
access Guard.Start eth0 readwrite;
access Guard.Start Guard.StartToScanner write;
access Guard.Scanner Guard.StartToScanner read;
access Guard.Scanner Guard.ScannerToFilters write;
access Guard.Filters Guard.ScannerToFilters read;
access Guard.Filters Guard.FiltersToEnd write;
access Guard.End Guard.FiltersToEnd read;
access Guard.End eth1 readwrite;
access Guard.End
Guard.FiltersToEnd
Guard.ScannerToFilters
Guard.StartToScanner
Syntax highlighting
Completion suggestions
Outline
eth0
eth1
Guard
End
Filters
FiltersToEnd
Scanner
ScannerToFilters
Start
StartToScanner
Error reporting
Problems
3 errors, 0 warnings, 0 info
Description
Resource
In Folder
Constraint violation: C
Constraint violation: C
Encountered "<EOF>"
simpleguard.fpol Simple Guard
Writable
Smart Insert
25 : 18

CDSFramework Concepts

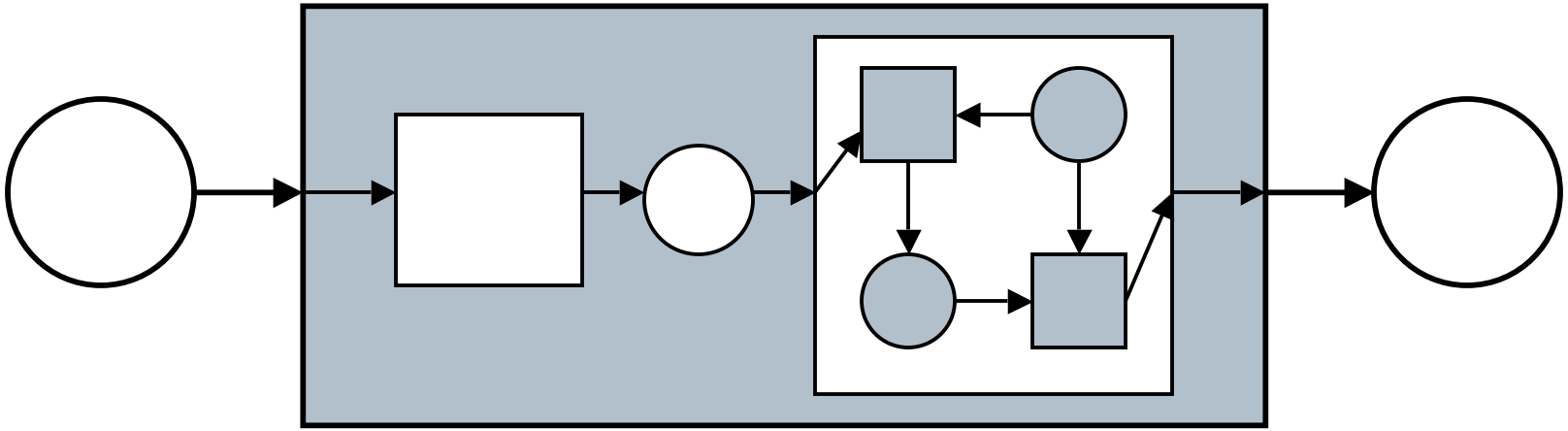
- Domains
  - Box represents security perimeter
    - Defines a separate security space for processes and private resources
- Shared Resources
  - Passive entities that allow domains to interact
    - Domains can share information only through shared resources
- Access
  - Depict information flow
  - Arrow from resource is a read, arrow to resource Is a write
  - Only connects domains to resources







CDSFramework Concepts - Decomposition



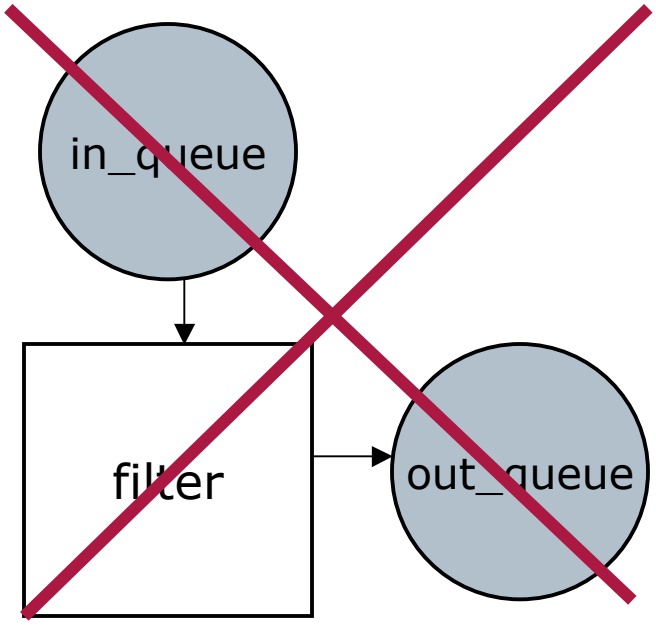
- Top-level design shows broad security parameters
- Decomposition offers better least-privilege
  - Child domains constrained by parent domains
  - Resources cannot be decomposed
  - Domains can be decomposed to any level

Real-world Challenges

- Concepts do not always mesh with SELinux details
  - High-level language must reflect reality
    - Be honest to policy authors
- Must integrate well with the Base Policy
  - Base Policy is the policy for the OS and non-CDSFramework applications
  - Need interfaces and labels from the Base Policy
  - Cannot break OS or other applications

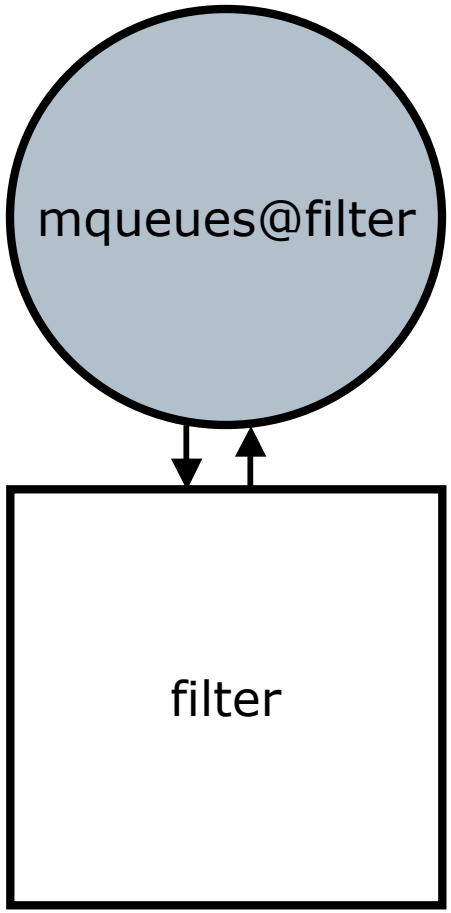
Real-world Challenges - IPC

- Some resources cannot be labeled ahead of time
  - Message queues, shared memory, semaphores
    - Gets label from the process that created it
- A domain cannot create two message queues with different permissions
  - Same label on both queues
  - Security-equivalent



Real-world Challenges – IPC

- CDSFramework solution: Control Resources
  - Identified as <IPC type>@<domain name>
    - Indicates that the IPC shares label with the domain
  - Only one control resource of each type per domain
    - Indicates security equivalence

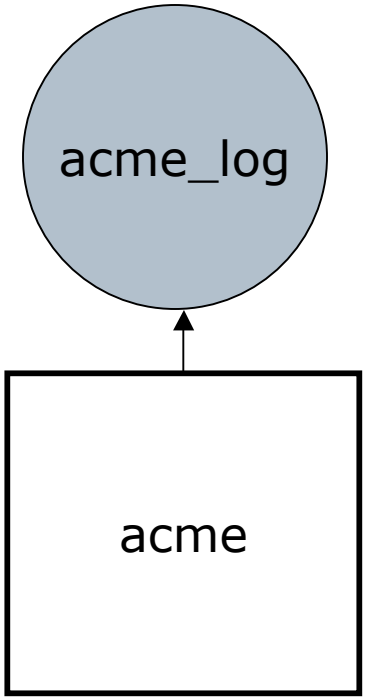


Real-world Challenges - Labeling

- Numerous ways to label files in SELinux
  - Several kinds: file, lnk_file, blk_file, chr_file, dir
  - New files inherit the label of the parent directory
    - Unless a transition rule says otherwise
  - Labeling rules can also use regular expressions
- Too complex for a high-level language
  - Need to abstract away low-level SELinux details
- Labeling must respect security goals

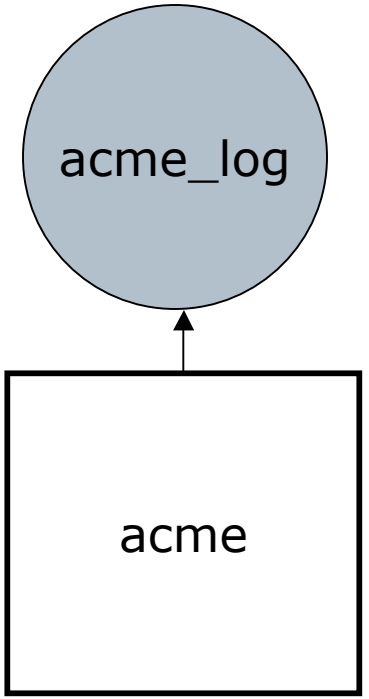
Real-world Challenges – Labeling

- Simplest for authors – just specify the path
- Example: author wants to put /var/acme/log in the acme_log shared resource
- This approach raises several questions
  - Is this path a file or a directory?
  - Does it exist at install-time, or is it created later?
    - If not, acme needs permission to add files in parent dir
      - acme can create any file it wants in the parent dir
  - Serious risk of information leakage
    - Major concern for a CDS
    - acme can put arbitrary information in the filename, any other domain which can read the parent directory can read that information



Real-world Challenges - Labeling

- Solution – label directories only
  - No risk of information leakage
  - Require directory to exist at install-time
  - No need for transition rules
  - Easy for policy authors
  - Forces application authors to follow good security engineering practices
    - Apps cannot add files to directories they do not own

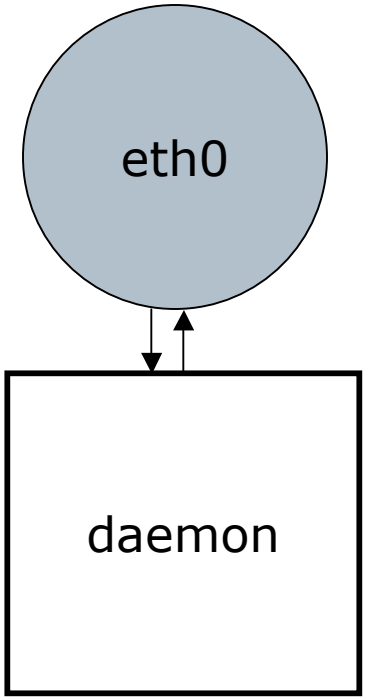


Real-world Challenges - Linkage

- CDSFramework domains need permissions on types in the base policy
  - Base policy resources
    - Network interfaces, /dev files, home directories, etc.
    - Relabeling would break other policies
  - Base policy domains: The starting point
    - What type can transition to a CDSFramework domain?
  - Integration
    - How can a CDSFramework domain use shared libraries or be managed by the kernel?

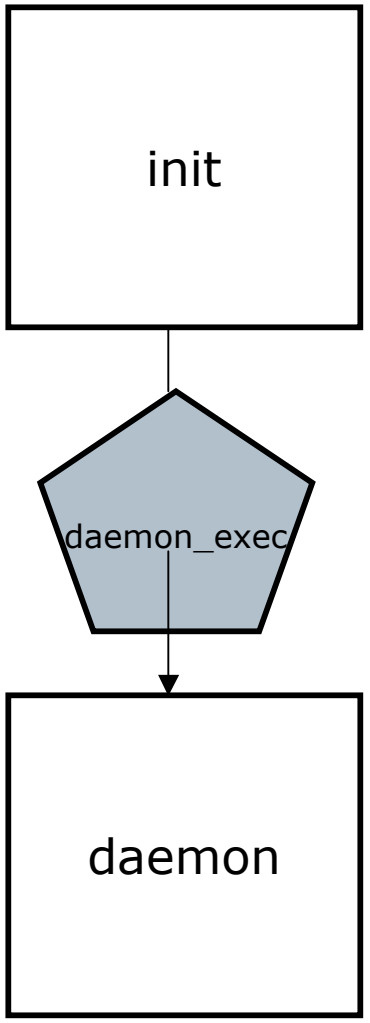
Real-world Challenges: baseresources

- Example: a daemon needs to use the network
- All of the networking rules are found in the base policy's corenetwork module
- Solution: wrap the object in a *baseresource*
  - Define read and write accesses that use interfaces from the base policy
  - Only one instance of each baseresource in a CDSFramework policy
    - Be honest about information flow



Real-world Challenges - basedomains

- Example: daemon is started by init
  - What type does init have?
  - What role does it run in?
  - What tty is it using?
- Solution: wrap init in a *basedomain*
  - Simply specify type, role, and tty type
  - Only one instance of each basedomain allowed in a CDSFramework policy
    - Be honest about information flow



Real-world Challenges - Integration

- What does it mean to be a domain or a resource?
  - What do all domains need to function?
    - Shared libraries, domain attribute, etc.
    - Manageable by the kernel
  - How will the resources be managed by the OS?
    - RPM, relabeling, etc.
- Solution: custom CDSFramework linkage macros
  - Compiler translates declarations into macros
    - Call base policy interfaces that all domains will need
  - Portability layer
    - To use CDSFramework with a different base policy, just update the linkage macros

Future Work

- Expand functionality in CDS problem space
  - Integrate with recommended guard libraries
    - IPC
    - Logging and auditing
- Generalize for broader SELinux adoption
  - Graphical editor makes policy easier to write
  - Tricky SELinux internals are covered

Future Work

- Graphical Policy Editor
  - To be added to Eclipse IDE
  - Drag-and-drop domains, resources, and entrypoints
  - 1-to-1 mapping between graphical and text representations
  - Draw the architecture in CDSFramework
    - Automatically produces corresponding security policy

QUESTIONS?