Type Enforcement & Pluggable MAC

Harvard CS252r Fa'13

Practical Domain and Type Enforcement for UNIX

- Badger, et al. 1995
- Trusted Information Systems

Domain and Type Enforcement

- Every Subject is associated with a *Domain*
- Every Object is associated with a *Type*

Boebart and Kain ’85 [7], page 23 introduces this
a given user could have multiple subjects operating in different domains
dynamically, it seems a subject and an object are associated with just one, statically: could be anything

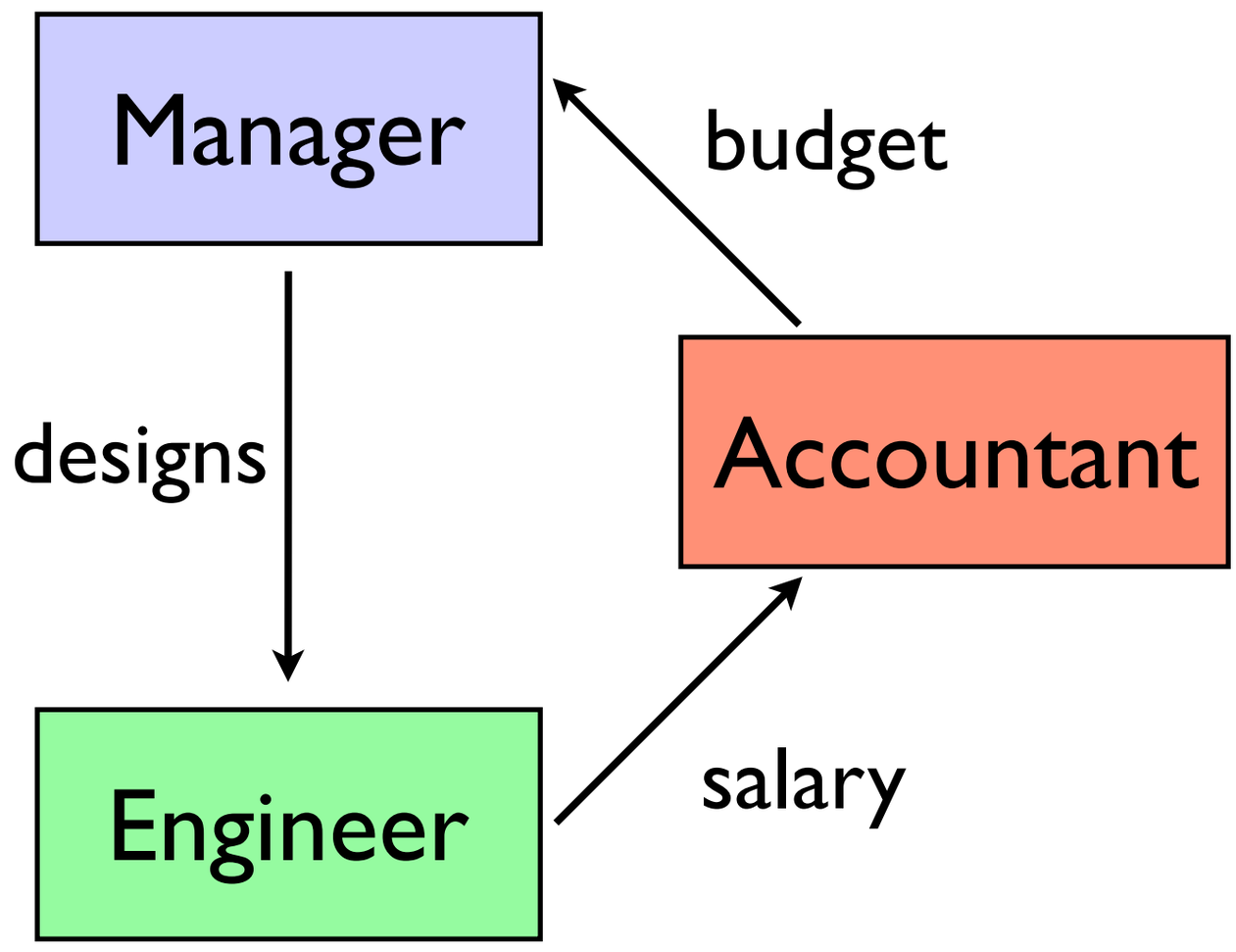
Domain and Type Enforcement

- Domain Definition Table
  - Domain's Access Rights to each Type
  - read, write, etc.
- Domain Interaction Table
  - Domain's Access Rights to other Domains
  - signal, create, destroy, etc.

Domain and Type Enforcement

- Strict Superset of Lattice-Expressible Security Policies

Domain and Type Enforcement



DTEL

Domain and Type Enforcement Language

- Encompasses Existing Security Policies
- Hierarchical Types
- Simple, Declarative Language

DTEL

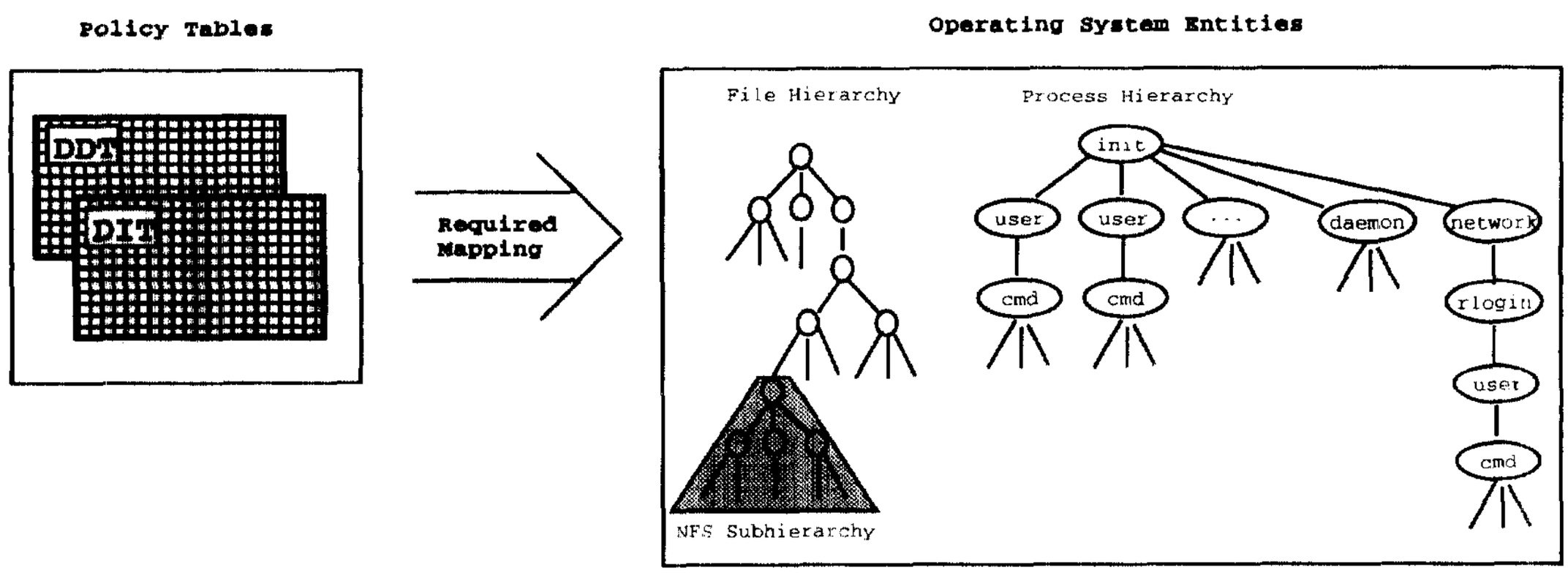


Figure 1: Mismatch Between Policy Concepts and System Structures

DTEL

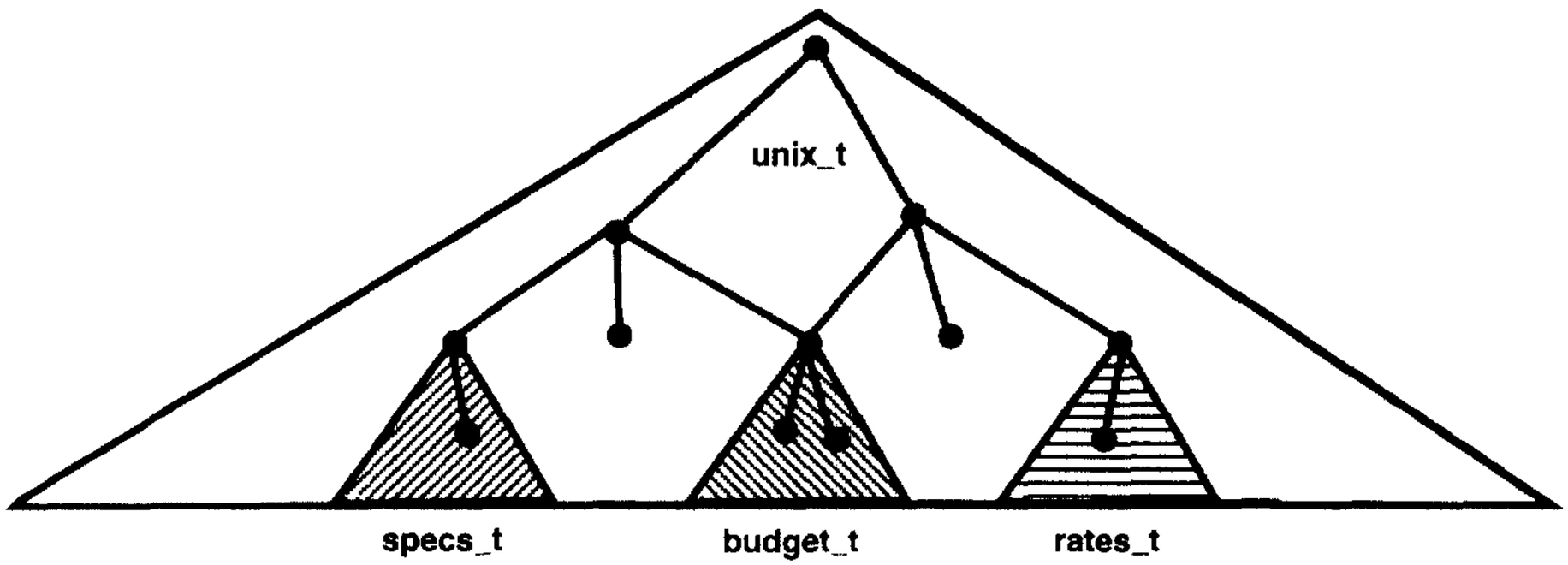


Figure 2: Implicit Types

running example of Unix system, Engineers, Managers, and Accountants

DTEL

```
/*
*       DTEL Commercial Policy.
*/

type    unix_t,         /* normal UNIX files, programs, etc.  */
        specs_t,        /* engineering specifications */
        budget_t,       /* budget projections */
        rates_t;        /* labor rates */

#define DEF             (/bin/sh), (/bin/csh), (rxd->unix_t) /* macro */

domain  engineer_d      = DEF, (rwd->specs_t);
domain  project_d       = DEF, (rwd->budget_t), (rd->rates_t);
domain  accounting_d    = DEF, (rd->budget_t), (rwd->rates_t);
domain  system_d        = (/etc/init), (rwxd->unix_t), (auto->login_d);
domain  login_d         = (/bin/login), (rwxd->unix_t), (exec->engineer_d,
                                                         project_d,
                                                         accounting_d);

initial_domain system_d;        /* system starts in this domain */

assign  -r -s           unix_t              /;      /* default for all files */
assign  -r -s           specs_t             /projects/specs;
assign  -r -s           budget_t            /projects/budget;
assign  -r -s           rates_t             /projects/rates;
```

Figure 3: Example DTEL Policy

-s : static, not permitted to change at run-time
-r : recursive, applies to subfiles/subdirectories
NB: use of CPP

Implementation

- Domain Definition Table and Domain Information Table are in kernel memory
- Object-Type association in kernel memory

Implementation

- File creation and deletion update policy
- Policy is mirrored on disc in plaintext
- Periodic snapshot

unless -s enforces a static policy

Implementation

- Write to disk when users writes to disk
- Thus, nearly match UNIX performance

Implementation

also ported the prototype to TMach Version 0.2. The DTE modifications consist of roughly 17,000 lines of commented C, lex, and yacc code, of which 3,600 lines comprise the DTEL processor.

Implementation

have also made several applications DTE-aware. Most significantly, we have implemented a DTE version of the login program that authenticates users for specific roles [18, 20, 4, 30] and then confines user sessions to specific environments using domain transitions authorized by the DTE policy. To allow users to view and, within DTE constraints, manipulate DTE attributes, we have implemented DTE-aware versions of the ls, ps, mkdir, and ln programs. These programs list directory contents and process states like the standard versions except that they also accept new arguments to display security attributes. To analyze DTE policies prior to use in the UNIX kernel, we have linked the DTE subsystem library into a test harness that checks for syntactic and contextual correctness and prints various reports. We have also implemented a modified version of the Emacs text editor that displays type attributes of file buffers and allows users to simultaneously view and manipulate labeled information in multiple windows.

- implemented a login program for UNIX
- reimplemented basic UNIX commands to be domain and type enforcement aware
- wrote a “contextual” correctness tester which prints “various reports”
- of course ... an Emacs mode

Design and Implementation of the TrustedBSD MAC Framework

- Watson, et al. 2003
- Network Associates Laboratories

- Network Associates Laboratories

Spiritual Successor to
Previous Paper's Lab

Design and Implementation of the TrustedBSD MAC Framework

- Dynamic Security Extensions
- Policy-Agnostic Object Labeling Services

Previous Work

- Direct Kernel Modification
- Duplicated Source Trees

- kernel modification is hard
- tracking changes to kernel is hard
- no confidence that other parts of kernel are secure

Previous Work

- Race Conditions
- Lock Order Problems

- classic setuid problem, kernel checks if file is OK, in between checking and execution, malicious thread swaps in a malicious executable
- coordinating security system locks with base kernel system locks is hard, must release locks on files in order to allow subsytem to load things, this creates a race with malicious threads

Previous Work

- Non-Compositional

- source code conflicts
- functionality conflicts

Previous Work

- rolling your own kernel patches is hard

Kernel Framework Approach

- Provide a MAC interface to kernel services
- Provide a MAC interface to security policies

- allow policies to main labels on kernel objects
- allow policies to select subsets of interfaces relevant to themselves

Kernel Framework Approach

- Access Control Entry Points
  - file system
  - IPC
  - network stack
  - etc.
- Interested policies may reject an action

Implemented Policies

- Biba: Hierarchical Fixed-Label Integrity
- BSDExtended: “file system firewall”
- ifoff: interface silencing
- MLS: Multi-Level Security w/ compartments
- sebsd: port of SELinux/FLASK/TE

User Applications

- Need not be aware of any non-UNIX access control
- Awareness enables appropriate policy manipulation and reading

Future Work

- Dynamic OS policy change in response to hostile environment

Further Discussion

- Capabilities in the context of DTE
- Role Based Access Control and Domains

- LSM vs TrustedBSD
- What is the relationship to SELinux?
- Is KeyKOS implementable as a TrustedBSD policy?

Capabilities and DTE

- Domains may execute programs in other domains
- Domains may have capabilities to other domains

- Combine a domain capability with a program capability to run a program?

- One capability to multiple different objects?

RBAC and DTE

- Subjects may be spawned in various domains
- During execution subject has exactly one domain
- A domain *U* with execution access to a set of domains *R* is a user permitted perform roles in *R*
- Roles are associated with a set of programs which may be run in that role
- Role inheritance?

LSM vs TrustedBSD

- LSM doesn't have compositionally (?)
- LSM only permits one policy to store labels
- SELinux is an LSM module

LSM vs TrustedBSD

(most commonly via FLASK). Unlike LSM, the MAC Framework place a strong focus on supporting infrastructure (such as labelling semantics and policy-agnostic label system calls) and the kernel synchronisation model, offering stronger guarantees for policy authors. Apple's Kernel Authorization framework (kauth) also provides kernel extensibility with the intention of supporting anti-virus systems, and has been adopted by NetBSD [9, 35], but has proven insufficiently expressive to support mandatory protection schemes, leading Apple to also adopt the TrustedBSD MAC Framework in their Mac OS X and iOS operating systems.

Watson, Robert N. M. *New approaches to operating system security extensibility.* University of Cambridge Technical Report. April 2012

SELinux

- Implemented as a Linux Security Module
- Implemented by the NSA and others
- Grew from the FLASK project around 2001
- Integrated into linux 2.6 September 2003

The Security-enhanced Linux prototype was developed by NSA in conjunction with research partners from NAI Labs, Secure Computing Corporation (SCC), and the MITRE Corporation. Since the initial public release, many other contributions have followed.

https://lkml.org/lkml/2003/7/14/286
https://lkml.org/lkml/2003/7/14/286

KeyKOS in TrustedBSD

- KeyKOS domains “obey” programs
- Domains hold keys
- Keys are capabilities
- Gate keys allow domain-domain comms
- Meter keys

KeyKOS in TrustedBSD

- MAC_PERFORM (respond to events with side effects)
- MAC_CHECK (authorize/deny with error message)

MAC_BOOLEAN use for “special case scenarios wher epolicies augment an existing kernel service decision rather than returning an access control result”.

KeyKOS in TrustedBSD

- TrustedBSD does not permit modifying results of system calls
- Disable all file system calls
- Provide alternative API