Clark-Wilson Integrity as a Security Goal for SELinux Policies

Trent Jaeger – IBM Research, Watson
Reiner Sailer – IBM Research, Watson

Problem: Policy Development

- Develop an ***Access Control Policy*** relative to an ***Application***
  - Satisfies a ***Functional Goal*** that enables the application to run
  - Satisfies a ***Security Goal*** that guarantees protection of the application
- Current Approach for SELinux
  - Base ***Access Control Policy*** == SELinux Example Policy
  - Test/modify against ***Functional Goal*** == no denial audits
  - Test/modify against ***Security Goal*** → ad hoc
- Examine a security goal called Analytic Integrity
  - Similar to Clark-Wilson, but some important distinctions
- Show what is necessary to support its use in policy development and deployment

Security Goal – Information Flow-Based

- Information Flow
  - If Subject *s* reads an object that Subject s_i's modifications can reach, then we have an information flow from Subject s_i to Subject *s*

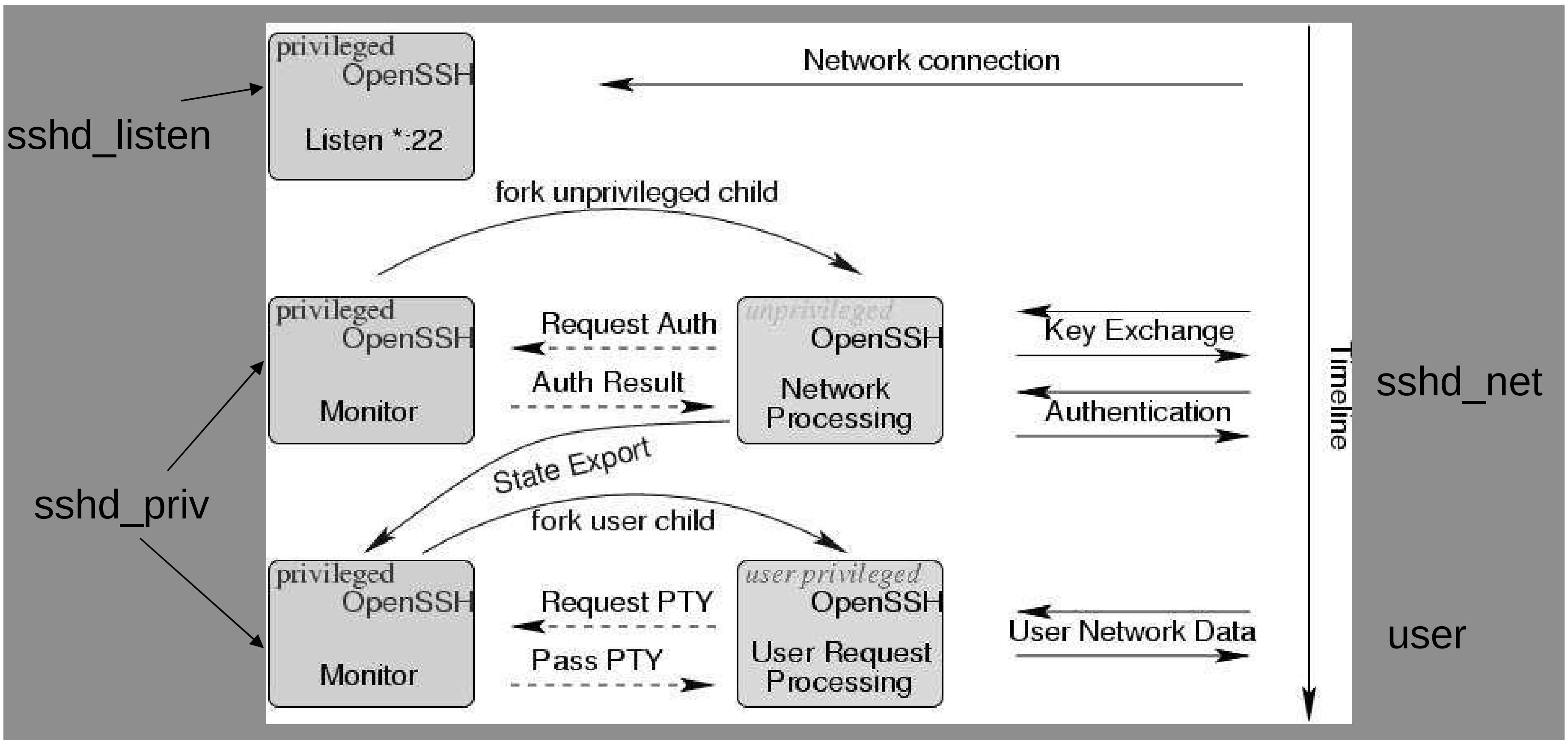
    $mod(s_i,o)$ and $obs(s,o) \rightarrow flow(s_i, s)$

- Information Flow Properties
  - Secrecy
  - Integrity
  - Assured Pipelines

- Integrity Is Focus

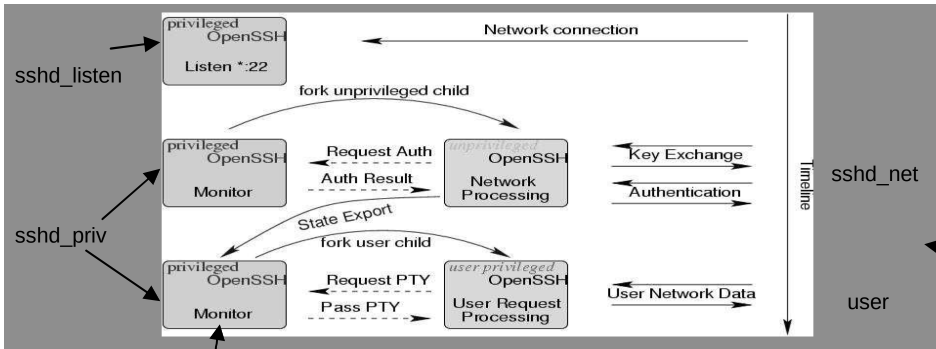
- Integrity Interpretation
  - ***Low integrity*** subjects write ***low integrity*** data
  - ***High integrity*** subjects write ***high integrity*** data
  - A ***low-to-high information flow*** occurs if a low integrity subject can write to an object that a high integrity subject can read
  - Information flows are ***intransitive***

Target Subject: Privilege Separated OpenSSH



Picture from Niels Provos

OpenSSH in an SELinux System



Picture from Niels Provos

Read Host Key
Modify Libraries
Modify Upgrades

Low Integrity Subject

Modify Executable File
Modify Configuration
Modify Environment

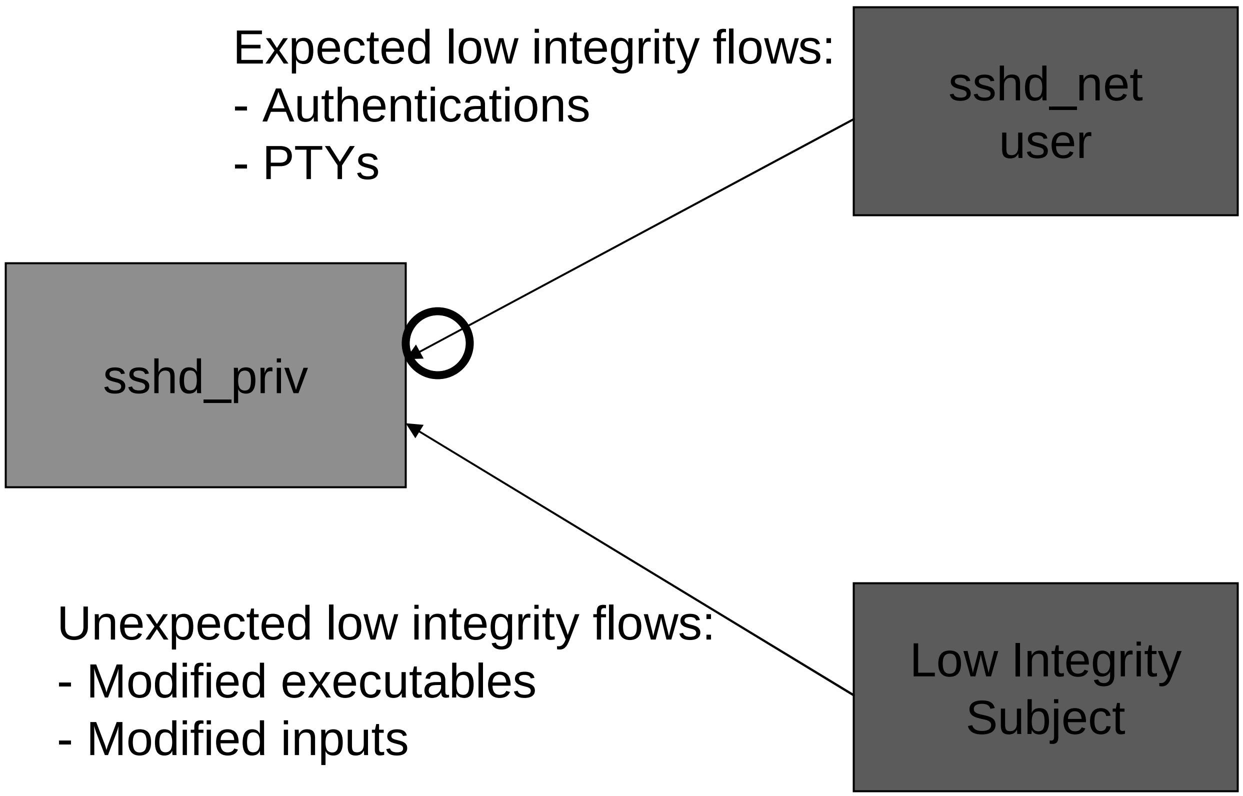
Low Integrity Subject

Low Integrity Subject

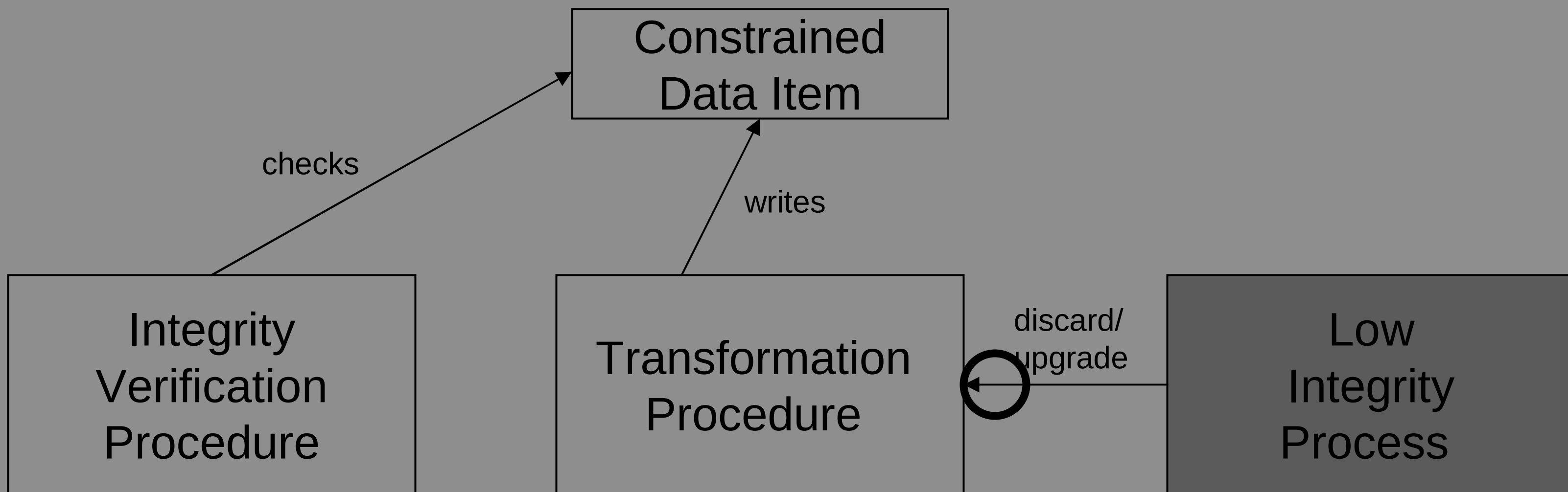
Low Integrity Subject

...

SSH Integrity Flow



Clark-Wilson Integrity Model



Effectiveness of IVP and TP are guaranteed based on assurance

Analytic Integrity

- Classical Integrity
  - Integrity is verified at time *t*
  - No leakage of secrets that enable masquerading as high integrity subject
  - For all flows $f(s_i, s)$ after time *t*

    $f(s_i, s) \rightarrow int(s_i) \geq int(s)$
- Interface Impact (requires justification for trust in programmer)
  - Consider flow to interfaces

    $mod(s_i, o)$ and $obs(s, o, x) \rightarrow f(s_i, s, x)$

    Discard/upgrade interfaces D(s, x)
- Analytic Integrity
  - Integrity is verified at time *t*
  - No leakage of secrets that enable masquerading as high integrity subject
  - For all flows $f(s_i, s, x)$ after time *t*

    $f(s_i, s, x) \text{ and not } D(s, x) \rightarrow int(s_i) \geq int(s)$

Policy Design Approach

- Start with
  - SELinux policy (example)
  - Analytic Integrity goal: Target subject types and Discard/upgrade interfaces
  - Functional scenarios: Permissions required and Other subjects required
- ***Find*** and ***Resolve*** Low-to-High Integrity Information Flows
  - Policy analysis
- ***Verify*** Application-level requirements
  - Presence of Discard/Upgrade Interfaces for low integrity flows
  - Prevent leakage of secrets (e.g., across fork)
  - Trust programmer not to sabotage Discard/Upgrade Interfaces
- ***Enable*** SELinux enforcement
  - Analytic Integrity enforcement

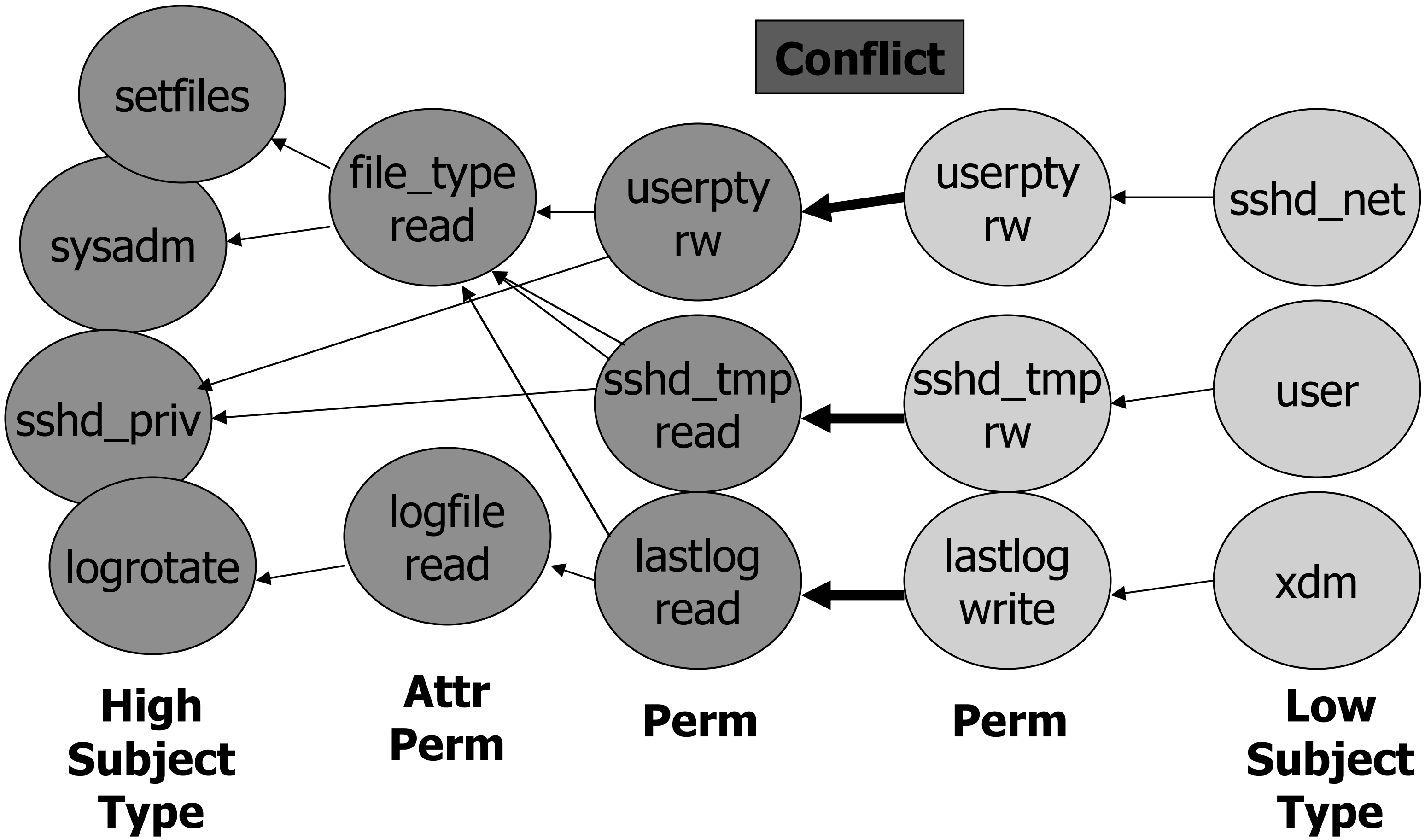
SELinux Integrity Information Flows

- SELinux cases
  - Low, High, and Any are subject types
  - Direct: Low writes type A; High reads type A
  - Indirect: Low writes type B; Any relabels type B objects to type A; High reads type A
  - Indirect: Low writes type B1; Any relabels type B1 to Bn to A; High reads type A
- Direct are *intransitive*
  - Low → High : bad
  - Low → Low : OK – only care about Low modifying High inputs
  - High → High : OK – Highs are trusted to discard/upgrade
- Indirect
  - Result of relabels indicates read/write relationship
  - Then treat as direct

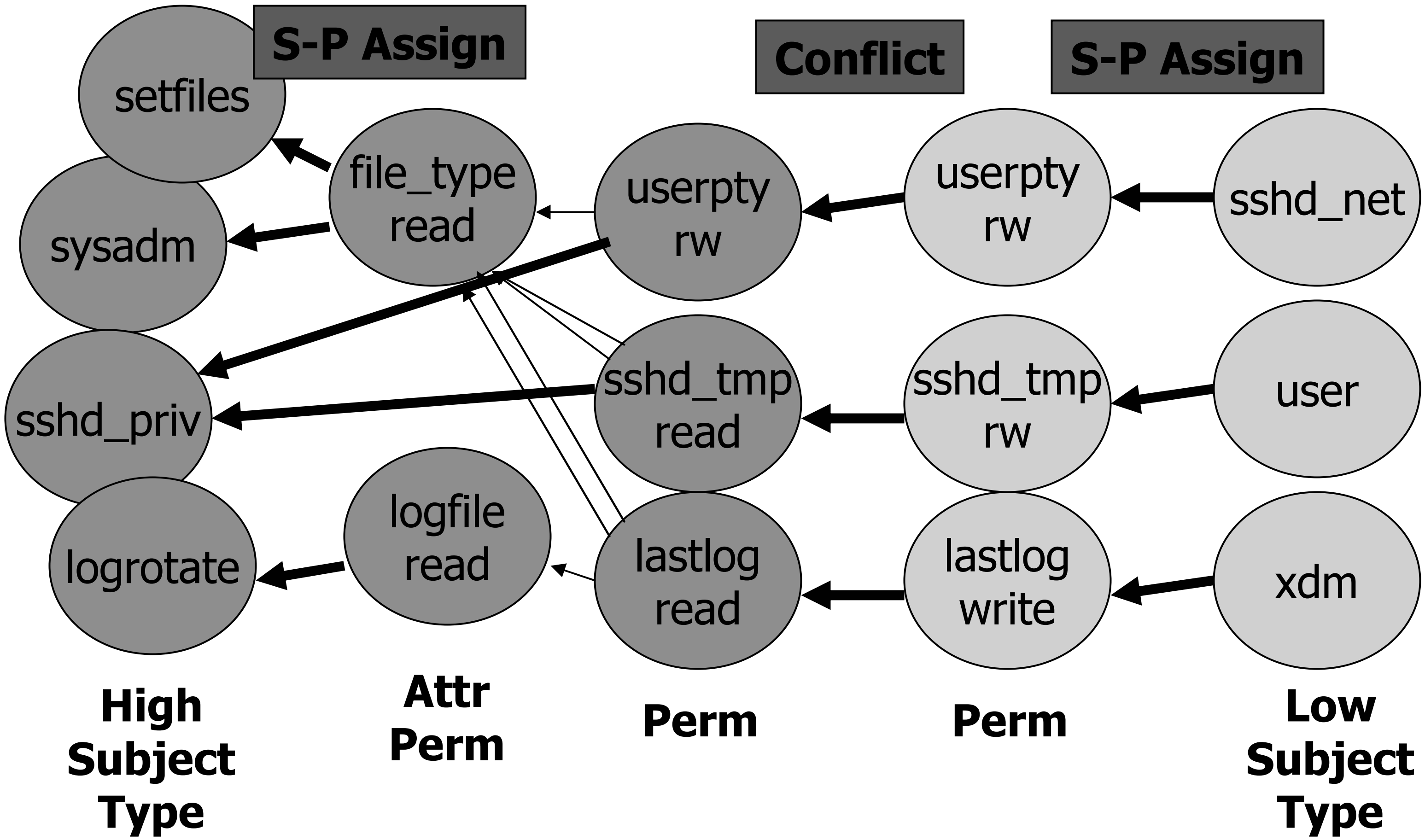
Gokyo Policy Analysis Tool

- Load entire SELinux example policy
  - Only needs attributes, types, and permission assignments
- Manage state of analysis for SELinux policy
  - Constraint file: Express constraints (2 lines)
  - Config File: Maintain configuration of high integrity and excluded subjects
  - Filter File: Maintain filter specification describing which permissions are excluded and which permissions a subject can filter
- Display conflicts in terms of **minimal cover set**
- Compute **basic impacts** for conflicts
- Enable resolution and re-evaluate
- Resulting policies achieve Analytic Integrity
  - Assuming verification of interfaces that discard or upgrade low integrity data
  - Assuming verification of no secret leakage
  - Assuming trust in programmer not sabotaging interfaces
- Does not enable SELinux module to enforce resolutions

SELinux Integrity Problem



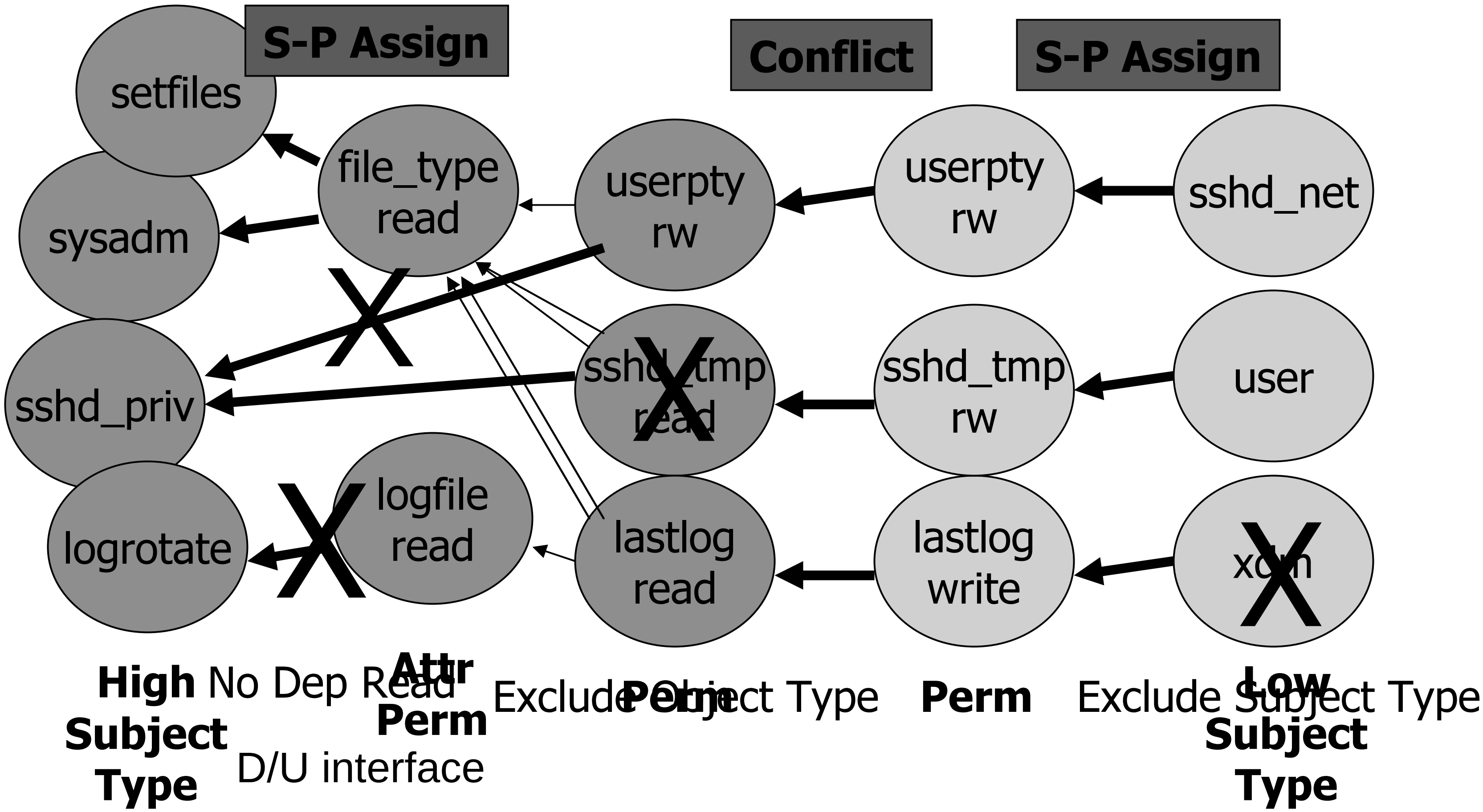
Minimal Cover Set



Integrity Resolutions

- Remove Subject Type or Object Type
  - Apply on nodes in assignments that lead to many conflicts
  - Try to remove subjects types that lead to many conflicts (sendmail, X)
- Reclassify Subject Type
  - Make low integrity subject a high integrity subject or vice versa
  - Apply on nodes in assignments that lead to many conflicts
  - Various kernel (init), admin (SELinux, install), authentication (sshd)
- Not dependent reads and D/U interfaces
  - Apply to assignments with few conflicts (no aggregation) on high integrity side
  - User requests, /var, terminals; Also, some reads are not dependent (e.g., logrotate)
- Change Subject Type-Permission assignment
  - Could apply to low integrity or high integrity side
  - Apply to assignments with few conflicts
- Deny Object Access
  - Track low integrity writes per object
  - Apply to assignments with no aggregation on high integrity side
- LOMAC Subject Type (sysadm)
  - Reduce integrity level of subject when reading low integrity data
  - Apply on nodes in assignments that lead to many conflicts

Example Resolutions



OpenSSH Information Flows

- **Use Gokyo to find information flows**
  - 5.6M SELinux example policy for Linux 2.6.6
  - 1200 subjects, 76000 permissions, 10^5 assignments
  - High-level goal: no low integrity inputs to sshd_priv and sshd_listen
- **Identify trusted/excluded subjects/objects**
  - 62 trusted subjects, 73 excluded subjects
  - Few excluded objects (mainly for X)
- **Determine if remaining permissions are really necessary**
  - sysadm:fifo – should be removed via LOMAC for sysadm
  - sshd_tmp – remove from perms
  - security:file – remove information flow from low integrity subject (crond)
- **Resolve remaining permissions**
  - userpty:chrfile – open for unprivileged, D/U interface
  - fd & network communication – D/U interface

Enable SELinux Enforcement

- **Distinguish between Safe and D/U Permissions**
  - sshd_priv_t – safe permissions
  - Filtering principal – allow_filter sshd_priv_t userpty:chr_file {…
  - Should not require filter per interface
    - Interfaces define strict requirements for what they allow
- **Ensure that Analytic Integrity permissions are only used at D/U interfaces**
  - do_filter_perms()
  - stop_filter_perms()
- **Supporting tools**
  - Use gdb & SELinux to find where D/U interfaces are in code
  - Verify no leakage of secret data across fork
  - Verify D/U interfaces
- **LOMAC subjects – sysadm**
- **Enables Privilege Separation**
  - Small privileged components
  - Limited information flow
  - Add information flow controls

Information Flow Analysis

- Gokyo
  - Identifies information flow conflicts
    For integrity, intransitive flows are sufficient
  - Represents all conflicts by minimal cover set
  - Supports iterative resolution

- Polgen/SLAT
  - Polgen finds functional goal
  - Security Goals in terms of information flow rules
  - Detect illegal flows 1 by 1

- SETools including Apol
  - Binary files may be the basis
  - Transitive information flow analysis
    Is there an information flow from type A to type B?
  - Weighted information flow permissions

Policy Design Approaches Reconsidered

- Still debugging at “assembler” level
  - Problems occur due to conflicts in low-level permissions assignments
  - Macros don’t really help – just an aggregate of low-level assignments
- Higher-level policy expression
  - State Analytic Integrity ***security goal*** for each target application
    - Allowed dependencies and interfaces of low integrity dependencies
  - Information flow expression of ***functional goal***
    - Required functional dependencies
  - These form the basis
  - Benefit from a policy knowledge base
    - Collect ***Trusted subjects*** and ***Non-dependent permissions***
    - Specify D/U interfaces
    - Leaves only information flows that must be filtered via D/U interfaces
- Does not achieve least privilege
  - Expert design using the assembler tools
    - Closer to least privilege
  - Common customization based on higher-level
    - More likely to be achievable

Summary

- Problem: Policy design
  - Basic approach
    Security Goal (testable), Functionality Goal (testable), Basis for Start
  - Current situation
    SG (none really); FG (it runs); Basis (selinux example)
- Proposed Information Flow as Basis for Security Goal
  - Integrity is fundamental, but very hard
  - Lots of illegal flows
- Evolved Clark-Wilson model to Analytic Integrity model
- Demonstrated policy development for Analytic Integrity
  - OpenSSH example
- Extended SELinux to support Analytic Integrity
- Considered extension of SELinux analysis tools to support Analytic Integrity
- Future – broader consideration of alternative policy development