Analysis of (Access Control) Policies
Outline

• Weighted Pushdown Systems
• Analysis of Security Policies
  ‣ SELinux analysis of mine and Stoller
  ‣ Program Analysis of Myers
Weighted Pushdown Systems

- A model of programs that uses weights to encode the effect of each statement on the data state of the program
  - PDS still represents control flow
  - Weights provide data abstraction
- Weights will need to support a variety of possible abstractions
Weighted Pushdown Systems

- Weight domains are a bounded idempotent semiring
  - Which is a tuple
    - Weight set \( D \)
    - Combine operator
    - Extend operator
    - \( \emptyset \) in \( D \) (identity element of combine)
    - \( \downarrow \) in \( D \) (identity element of extend)
- Weight domains must enable abstractions to be extended (values updated) and combined (via joins)
Combine and Extend

• (D, combine) is a **commutative monoid** with 0 as its neutral element
  ‣ Monoid – a set with a binary operation \( \cdot \) that satisfies
    • Closure: \( a, b \) in \( S \), \( a \cdot b \) in \( S \)
    • Associativity: \( (a \cdot b) \cdot c = a \cdot (b \cdot c) \)
    • Identity element: there exists \( e \) in \( S \), s.t., for all \( a \) in \( S \), \( a \cdot e = e \cdot a = a \)
  ‣ Commutative monoid – is endowed with its algebraic preordering
    \( x \leq y \), iff there exists a \( z \), s.t. \( z + x = y \) (enables join)

• Extend distributes over combine

• 0 is an annihilator wrt extend
Weighted Pushdown System

- Definition
- WPDS is a triple \((P, S, f)\) where
  - \(P\) is a PDS
  - \(S\) is a bounded idempotent semiring (weight domain)
  - \(f\) is a map that assigns a weight to each rule of \(P\)
WPDS expresses PDS

- A PDS $P$ is a WPDS $W$ with the boolean weight domain $S$
  - $S = (\{F, T\}, \text{OR}, \text{AND}, F, T)$
  - Weight assignment $f(r) = T$ for all rules in $P$
    - All rules are true
- $\text{JOVP}(C_1, C_2) = T$ iff there exists a path from a configuration in $C_1$ to a configuration in $C_2$
Finite-State Data Abstractions

- Can encode data abstractions for finite sets
- E.g., binary relations on a finite set
  - $S = (2^{G \times G}, \text{union}, \text{compose}, \text{null}, \text{id})$, where
    - $\text{Union}$ is combine and $\text{compose}$ (relational composition) is extend
    - Empty relation $\text{null}$ is $\emptyset$ and identity relation $\text{id}$ is $\top$
- Check properties of weight domain against definition
Finite-State Data Abstractions

• JOVP(C1, C2)
  ‣ From start to n, C1 = \{<p, start>\} and C2 = \{<p, nu>\}
    • Null if n cannot be reached
    • Otherwise, JOVP captures transformation on global state G through compose and union (join) creating the set of valuations that reach n

• Poststart(p, n1) in Fig 2.9 gives weight at n6 of w6, which represent possible values of x, y at that statement
Infinite-State Data Abstractions

• Number of states is infinite, such as integers

• Verify definition 2.2.10 is a weight domain
  ‣ Minpath semiring \( M = (\mathbb{N} \cup \{\text{infinity}\}, \text{min}, +, \text{infinity}, 0} \)

• Find shortest path trace
  ‣ E.g., give each rule a weight of 1
  ‣ Then, JOVP is length of shortest path (assuming a combine of min)
Weighted Relation

• A **weighted relation** is a function from \((C_1, C_2)\) to \(D\)
  
  ‣ Can compose two weighted relations
  
  ‣ \((R_1; R_2)(s_1, s_3) = \text{combine}\{w_1 \text{ extend } w_2 \mid \text{exists } s_2 : w_1 = R_1(s_1, s_2), w_2 = R_2(s_2, s_3)\}\)

  ‣ Can union two weighted relations
  
  ‣ \((R_1 \text{ union } R_2)(s_1, s_2) = R_1(s_1, s_2) \text{ combine } R_2(s_1, s_2)\)

• To find shortest path that exhibits some property \(R\)
  
  ‣ Weight = 1 if \((g_1, g_2)\) in \(R\)
  
  ‣ Weight = infinity if \((g_1, g_2)\) not in \(R\)
Affine Programs

- Programs for which *affine relation analysis* can be precisely performed
  - Where linear-equality constraints between integer-valued variables can be determined
- Constraints
  - $x_j = a_0 + \sum_{i=1}^{n} a_ix_i$
  - Or assignments can be non-deterministic
ARA Weight Domain

• Linear algebra formulation
  ‣ Represented by a column vector (matrix): $[a_0, \ldots, a_n]$
    • $n$ is the number of (global) variables
  ‣ An affine relation represents the set of all valuations of program variables that satisfies it
  ‣ A concrete valuation must be a subset of all satisficing valuations for affine relation
ARA Weight Domain

- Problem: Find all affine relations in a program
  - Abstract each statement as a set of matrices of size \((n+1) \times (n+1)\)
  - Weakest pre-condition transformer of matrices (more to finding this)

- Weight Domain
  - Basis of their linear span
  - Creates a vector space within which all valuations of program variables exists
  - \textit{Combine} creates the smallest vector space containing the input vector spaces
Solving for JOVP

• Defining prestar and poststar for WPDSs

• Like PDSs, create an W-automaton, which is a P-automaton where each transition of the automaton is labeled with a weight

  • Weight of a path in the automaton is obtained by taking an extend of the weights in the transitions in the path

  • Acceptance of a configuration c = <p, u> with weight w = A(c) occurs if w is the combine of weights of all accepting paths for u starting from state p in A

    ‣ Prestar(A) produces JOVP({c}, L(A)) – i.e., configurations accepted starting from c in A – and Poststar(A) does opposite

• Need both the forwards and backwards automata – why?
Does a security policy in a program or a system prevent vulnerabilities?

- What is a vulnerability?
- How do we check that?
Example Attack

From SANS : The Top Security Risks (Tutorial)
From SANS: The Top Security Risks (Tutorial)
**SANS Example**

From SANS: The Top Security Risks (Tutorial)

---

Step 2: Establish Reverse Shell Backdoor Using HTTPS

In Step 2, the attacker's exploit code installs a reverse shell backdoor program on the victim machine. This program gives the attacker command shell access of the victim machine, communicating between this system and the attacker using outbound HTTPS access from victim to attacker. The backdoor traffic therefore appears to be regular encrypted outbound web traffic as far as the enterprise firewall and network is concerned.

Steps 3 & 4: Dump Hashes and Use Pass-the-Hash Attack to Pivot

In Step 3, the attacker uses shell access of the initial victim system to load a local privilege escalation exploit program onto the victim machine. This program allows the attacker to jump from the limited privilege user account to full system privileges on this machine. Although vendors frequently release patches to stop local privilege escalation attacks, many organizations do not deploy such patches quickly, because such enterprises tend to focus exclusively on patching remotely exploitable flaws. The attacker now dumps the password hashes for all accounts on this local machine, including a local administrator account on the system.

In Step 4, instead of cracking the local administrator password, the attacker uses a Windows pass-the-hash program to authenticate to another Windows machine on the enterprise internal network, a fully patched client system on which this same victim user has full administrative privileges. Using NTLMv1 or NTLMv2, Windows machines authenticate network access for the Server Message Block (SMB) protocol based on user hashes and not the passwords themselves, allowing the attacker to get access to the file system or run programs on the fully patched system with local administrator privileges. Using these privileges, the attacker now dumps the password hashes for all local accounts on this fully patched Windows machine.

Step 5: Pass the Hash to Compromise Domain Controller

In Step 5, the attacker uses a password hash from a local account on the fully patched Windows client to access the domain controller system, again using a pass-the-hash attack to gain shell access on the domain controller. Because the password for the local administrator account is identical to the password for a domain administrator account, the password hashes for the two accounts are identical. Therefore, the attacker can access the domain controller with full domain administrator privileges, giving the attacker complete control over all other accounts and machines in that domain.

Steps 6 and 7: Exfiltration

In Step 6, with full domain administrator privileges, the attacker now compromises a server machine that stores secrets for the organization. In Step 7, the attacker exfiltrates this sensitive information, consisting of over 200 Megabytes of data.

The attacker pushes this data out to the...
From SANS : The Top Security Risks (Tutorial)
From SANS: The Top Security Risks (Tutorial)
Current Attacks

• Attack unprivileged processes first
  ‣ Then, escalate privilege incrementally via local exploits
  ‣ Leverage (unjustified) trust between processes-hosts to propagate attacks

• Such Attack Paths are ubiquitous in current systems
  ‣ Processes are tightly interconnected
    • Historically, all user processes have same privilege and can utilize system services
    ‣ Any control flow vulnerability can be leveraged to run any code
      • Return-oriented programming

• Claim: Adversaries will use any undefended path
Current Defenses

- We have made progress the last 10 years or so
  - Vulnerable network services galore → hardened, privilege-separated daemons (OpenSSH)
  - Default-enabled services → hardened configurations (IIS)
  - Root system processes galore → Mandatory access control (Linux, BSD)
  - Application plug-ins in same address space → Run application code in separate processes (Chrome, OP browsers)
  - Email attachments compromise system → Prevent downloaded content from modifying system (MIC, antivirus)
  - A process in one host can easily access another host → Limit open ports (host firewalls, labeled networking)
MAC Operating Systems

- Mandatory Access Control (MAC) operating systems
  - Define an immutable set of labels and assign them to every subject and object in the system
  - Define a fixed set of authorized operations based on the labels
- Now available in most commodity operating systems (Trusted Solaris, TrustedBSD, SELinux, AppArmor, Windows MIC*, etc)
Idealized Security

- **Multilevel Security** (MLS) for secrecy
  - **Secrecy requirement:** Do not leak data to unauthorized principals
  - Only permit information to flow from less secret to more secret principals/objects
  - E.g., Can only read a file if your clearance dominates that of the file

- **Biba Integrity**
  - **Integrity requirement:** Do not depend on data from lower integrity principals
  - Only permit information to flow from high integrity to lower integrity
  - E.g., Can only read a file if your integrity level is dominated by the file’s
Information Flows

- **Secrecy** (MLS): If the OS permits a secret application/object to flow to a public application/object, then there may be a leak (e.g., Trojan horse)

- **Integrity** (Biba): If the OS permits a low integrity input to flow to a high integrity application/object, then there may be a dependency (e.g., buffer overflow)
Practical vs. Ideal

- Do these idealized approaches based on information flow enable practical realization of OS enforcement?

- Secrecy is possible in some environments
  - Implemented in a paper world, previously
  - Still depend on many “declassifiers”

- Integrity has not been realized in practice
  - Many processes provide high integrity services to others

- Result: Depend on many applications to manage information flows
Example: logrotate

- **Logrotate** is a service that swaps logs
- It rotates logs through sequence
  - Secrecy: Logs may span all security levels on system
  - Thus, *logrotate* is trusted in MLS
- It reads a configuration to tell it what to do
  - Integrity: Logs must not leak into configuration files
  - Thus, *logrotate* is trusted to protect integrity
The OS trusts that privileged applications preserve system secrecy (30+ programs)

SELinux/MLS:

Policy management tools: secadm, load_policy, settrans, setfiles, semanage, restorecon, newrole

Startup utilities: bootloader, initrc, init, local_login

File tools: dpkg_script, dpkg, rpm, mount, fsadm

Network utilities: iptables, sshd, remote_login, NetworkManager

Auditing, logging services: logrotate, klogd, auditd, auditctl

Hardware, device mgmt: hald, dmidecode, udev, kudzu

Miscellaneous services: passwd, tmpreaper, insmod, getty, consoletype, pam_console
Integrity Situation Is Much Worse

- **Clients**
  - Lots of client programs are entrusted with information with different secrecy/integrity requirements
  - Email, browser, IM, VOIP, …

- **Servers**
  - Historically, many servers have enforced security policies because they handle multiple clients
  - Web servers, databases, mail, respositories, …

- *Information flow alone is not enough to build a secure system!*
Compliance Problem

- Evaluating whether a policy permits an adversary to have unauthorized access (i.e., contains an error) is a compliance problem:
  - **System Policy**: describes a system’s behavior
  - **Goal Policy**: describes acceptable behavior
  - **Mapping function**: relates elements from the system policy to elements in the goal policy
  - A compliant system policy is guaranteed to meet the requirements defined by the goal policy
We represent a single MAC policy with an information flow graph

- Used in analyses for SELinux by Tresys, Stoller, Li, Jaeger, etc.
The policy compliance problem for a single policy is set up as follows:

- **System policy** – The policy that we are analyzing is represented as a graph.

![Diagram showing relationships between different system components such as ftpd_t, etc_t, kernel_t, var_t, sbin_t, and installer_t. The diagram illustrates the network of relationships and dependencies between these components.]
Compliance Problem

- The policy compliance problem for a single policy is set up as follows:
  - *System policy* – The policy that we are analyzing is represented as a graph
  - *Goal* – The security goal is a lattice that defines integrity levels and rules that guarantee the integrity of the system
The policy compliance problem for a single policy is set up as follows:

- **System policy** – The policy that we are analyzing is represented as a graph
- **Goal** – The security goal is a lattice that defines integrity levels and rules that guarantee the integrity of the system
- **Mapping** - Assigns integrity levels to policy labels

![Diagram](image)
Compliance Problem

- The policy compliance problem for a single policy is set up as follows:
  - **System policy** – The policy that we are analyzing is represented as a graph
  - **Goal** – The security goal is a lattice that defines integrity levels and rules that guarantee the integrity of the system
  - **Mapping** - Assigns integrity levels to policy labels

Do all flows meet the requirements defined by the goal?
Other Compliance Problems

- Information flow compliance in programs
  - Data flow is determined by program data flows – security-typed languages, such as Jif, Sif, SELinks, FlowCaml

- Goal policy is not a lattice
  - Illegal reachability: no path from $u \rightarrow_{G} v$
  - Illegal sets of permissions: annotate edges with permissions

- Goals as functional requirements (e.g., obligations)
  - The presence of a node, edge, or path is required
  - These are functional constraints, rather than security
Find SELinux TCB

- Can we identify a TCB in SELinux Example Policy whose integrity protection can be managed (circa Linux 2.4.19)?
  - See [USENIX Security 2003]

- Tasks:
  - Can We Identify Trusted Programs?
  - Can We Define a Security Goal to Protect These Programs?
  - Can We Verify This Goal?
  - How Do We Debug Conflicts?
Type Enforcement

• Least privilege MAC policy used by SELinux
  ‣ Subjects have a label
  ‣ Objects have a label
  ‣ Permissions define object labels accessible to subject labels

• Several systems use (or have used) a form of TE
  ‣ SELinux uses labels called types

• TE policies are fine-grained and complex
  ‣ SELinux has 10,000s of rules

• SELinux has added abstractions, such as attributes and roles
Proposed Approach

• Propose a TCB from SELinux subjects
• Identify Biba integrity violations
• “Handle” integrity violations
  ‣ Classify integrity violations
  ‣ Remove violations that can be managed by application
    • Application is trusted to protect itself
  ‣ Revise TCB proposal
  ‣ Revise SELinux policy
• Result: All information flows are legal or accounted
Propose a TCB - Detail

- Use SELinux transition graph (i.e., who can exec whom) for server programs (e.g., httpd_t) to identify base subject types

- Ones that provide TCB services (e.g., authentication)

- Others that have many transitions (hard to contain)
Identify Integrity Violations

- Biba Integrity Analysis -- Gokyo, PAL, PALMS
- TCB subject types → read/exec perms
  - Generate corresponding “integrity-sensitive write” perms
- Others → write perms
  - Generate corresponding “integrity-sensitive read” perms
- Analysis
  - Do Others’ write to integrity-sensitive writes?
  - Do TCB Subjects read from integrity-sensitive reads?
Integrity Analysis

High Subject

Subject

Perm

Object Read

Object Write

Low Subject

Perm

Subject

Low Subject

Can Modify Input To High
Are There Integrity Violations?

• For Linux 2.4.19 -- SELinux Strict Policy
  • Permissions
    ‣ 129 perms used to “read down”
      • 57 socket perms, 25 fifo perms
    ‣ 1583 perms used to “write up”
  • Subjects
    ‣ 28 high integrity subjects “read down”
      • 35 for sysadm_t, 4 for load_policy_t
    ‣ 150 low integrity subjects “write up”
The subject-permission assignments that lead to a conflict result in a minimal cover of all conflicts.
Example Resolutions

- **Setfiles**
  - sysadm
  - sshd
  - logrotate

- **File_type**
  - read

- **User_ssh**
  - rw

- **File**
  - read

- **Logfile**
  - read

- **Lastlog**
  - read

- **Conflicts**
  - SSHD_TMP
  - rw

- **Exclude Object Type**
  - Deny Access

- **Attr Perm**
  - No Deep Read
  - Exclude Subject Type

- **S-P Assign**
  - user
  - httpd
  - admin
  - xdm

- **Low Subject Type**
  - user
  - httpd
  - admin
  - xdm
Integrity Resolutions

- Remove Subject Type or Object Type
- Reclassify Subject Type of Object Type
- Change Subject Type-Permission assignment
- Clark-Wilson reads
  - Allow reading of low integrity data that meet Clark-Wilson
- Deny Object Access
  - Track low integrity writes per object
- LOMAC Subject Type (sysadm)
  - Reduce integrity level of subject when reading low integrity data
Analysis Summary

• Conclusions
  ▶ Biba Information Flow Integrity
    • May not be so far off practical
    • But, we cannot force Biba (or other ideal models, e.g., LOMAC)
  ▶ Need to address conflicts
    • Identify resolutions

• Approach
  ▶ Compliance Problem
  ▶ Multiple types of resolutions
Questions