CSE543 - Introduction to Computer and Network Security

Module: Reference Monitor

Professor Trent Jaeger
Living with Vulnerabilities

• So, software is potentially vulnerable
  ‣ In a variety of ways

• So, how do you run a secure system when many processes may be compromised?
Living with Vulnerabilities

• So, software is **potentially vulnerable**
  ‣ In a variety of ways

• So, how do you run a secure system when many processes may be compromised?
  ‣ Only depend on secure code
Living with Vulnerabilities

• So, software is potentially vulnerable
  ‣ In a variety of ways

• So, how do you run a secure system when many processes may be compromised?
  ‣ Only depend on small amount of secure code
  ‣ Protect yourself from insecure code
  ‣ (Know the difference)
Trusted Computing Base (TCB)

• The trusted computing base is the infrastructure that you assume will behave correctly
  ‣ Hardware (keyboard, monitor, …)
  ‣ Operating Systems
  ‣ Implementations
  ‣ Local networks
  ‣ Administrators
  ‣ Other users on the same system

• **Axiom**: the larger the TCB, the more assumptions you must make (and hence, the more opportunity to have your assumptions violated).
Operating System and Security

• What guarantees do you depend on the operating system to enforce?
Operating System and Security

• What guarantees do you depend on the operating system to enforce?
  • Ideally
  • Integrity
    ‣ Adversaries cannot modify your process
  • Secrecy
    ‣ Adversaries cannot steal data from your process
  • Availability
    ‣ Adversaries cannot prevent your process from using hardware resources
Operating System and Security

- What **guarantees** do you depend on the operating system to enforce?
- **Really...**
- **Integrity**
  - Control modifications to process (memory) and resources used by process (files, etc.)
- **Secrecy**
  - Control access to process (memory) and resources used by process (files, etc.)
- **Availability**
  - Control shared use of hardware
Integrity Threat

• Confused Deputy
  ‣ *Process is tricked into performing an operation on an adversary’s behalf that the adversary could not perform on their own*
  • E.g., Malicious input via files, network, IPC, ...
Secrecy Threat

• Trojan Horse
  ‣ Some process of yours is going to give away your secret data
  • E.g., download a malicious app
Access Policy Enforcement

• A *protection system* answers authorization queries using a protection state \((S)\), which can be modified by protection state methods \((M)\)
  ‣ Authorization query: Can subject perform requested operation on object? Y/N

• A *protection state* \((S)\) relates subjects, objects, and operations to authorization query results
  ‣ E.g., in mode bits, ACLs, ... --- the policy

• A *protection state methods* \((M)\) can change the protection state
  ‣ Add/remove rights for subjects to perform operations on objects
The Access Matrix

- An **access matrix** is one way to represent a protection state.
  - Conceptual

- Columns are objects, subjects are rows.
  - To determine if $S_i$ has right to access object $O_j$, find the appropriate entry.
  - Often entries list the set of operations permitted for that subject-object pair

- The access matrix represents $O(|S| \times |O|)$ rules

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>$S_2$</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>$S_3$</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
The Access Matrix

- Suppose the private key file for J is object $O_1$
  - Only J can read
- Suppose the public key file for J is object $O_2$
  - All can read, only J can modify
- Suppose all can read and write from object $O_3$
- What’s the access matrix?

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>$O_2$</th>
<th>$O_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>$S_2$</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>$S_3$</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
The protection domain restricts access of processes to our computing system’s resources.

How is this done today?
- Memory protection
- E.g., UNIX protected memory, file-system permissions (rwx...)

The set of protection domains is the protection state of the system.
Protection Domains

Protection domain

- The protection domain restricts access of processes to our computing system’s resources
- How is this done today?
  - Memory protection
  - E.g., UNIX protected memory, file-system permissions (rwx…)

The set of protection domains is the protection state of the system
Access Policy Enforcement

• A protection system uses a *reference validation mechanism* to produce and evaluate authorization queries
  ‣ **Interface**: Mediate *security-sensitive operations* by building queries to evaluate
  ‣ **Module**: Determine relevant protection state entry (ACLs, capabilities) to evaluate authorization query
  ‣ **Store**: Store/Retrieve protection state entries

• How do we know whether a reference validation mechanism is correct?
Security-Sensitive Operations

• Broadly, operations that **enable interaction among processes** that violate secrecy, integrity, availability

• Which of these are security-sensitive? Why?
  ‣ Read a file (*read*)
  ‣ Get the process id of a process (*getpid*)
  ‣ Read file metadata (*stat*)
  ‣ Fork a child process (*fork*)
  ‣ Get the metadata of a file you have already opened? (*fstat*)
  ‣ Modify the data segment size? (*brk*)

• Require protection for all of **CIA**?
Reference Monitor

• Defines a set of requirements on reference validation mechanisms
  ‣ To enforce access control policies correctly
• Complete mediation
  ‣ The reference validation mechanism must always be invoked
• Tamperproof
  ‣ The reference validation mechanism must be tamperproof
• Verifiable
  ‣ The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured
Commodity OS Security

• UNIX and Windows Protection Systems
  ‣ Do they satisfy the reference monitor concept?
The UNIX FS access policy

- Really, this is a bit string ACL encoding an access matrix
- E.g.,
  
  \[ \text{rwx rwx rwx} \]
  
  - World
  - Group
  - Owner

- And a policy is encoded as “r”, “w”, “x” if enabled, and “-” if not, e.g,
  
  \[ \text{rwxrwx--x} \]

- Says user can read, write and execute, group can read and write, and world can execute only.
Caveats: UNIX Filesystem

- Access is often not really this easy: you need to have certain rights to parent directories to access a file (execute, for example)
  - The reasons for this are quite esoteric
- The preceding policy may appear to be contradictory
  - A member of the group does not have execute rights, but members of the world do, so …
  - A user appears to be both allowed and prohibited from executing access
  - Not really: these policies are monotonic … the absence of a right does not mean they should not get access at all. If any of your identities have that right in any class (world, group, owner), you are authorized.
Windows Grows Up ...

- Windows 2000 marked the beginning of real OS security for the windows systems ...
Tokens

• Like the UID/GID in a UNIX process
  ‣ User
  ‣ Group
  ‣ Aliases
  ‣ Privileges (predefined sets of rights)

• May be specific to a domain

• Composed into global SID

• Subsequent processes inherit access tokens
  ‣ Different processes may have different rights
Access Control Entries

- DACL in the security descriptor of an object
  - e.g., like “rwx”
  - List of access control entries (ACEs)

**ACE structure (proposed by Swift et al)**

1. **Type** (grant or deny)
2. **Flags**
3. **Object Type**: global UID for type (limit ACEs checked)
4. **InheritedObjectType**: complex inheritance
5. **Access rights**: access mask
6. **Principal SID**: principal the ACE applies to
ACE Authorization

- The ACEs for a particular request are totally ordered.
- Start form the top and check each:
- Checking algorithm
  - Authorizing for SIDs in token on set of rights
    1. if ACE matches SID (user, group, alias, etc)
      a. ACE denies access for specified right -- deny
      b. ACE grants access for some rights -- need full coverage
    2. If reach the bottom and not all granted, request denied
Access Checking with ACEs

- Example
Commodity OS Security

• UNIX and Windows Protection Systems
  ‣ Do they satisfy the reference monitor concept?
  ‣ Mediation? Tamperproofing? Verifiability?
Mediation

- Nothing fundamentally prevents UNIX or Windows from providing complete mediation
  - However, experience demonstrated that UNIX implementations did not satisfy complete mediation
    - See Linux Security Modules next time
  - Windows is more flexible in mediation
    - E.g., can define new object types for controlling their access
    - But, not formally assured
Tamperproofing

• Commodity operating systems have a fundamental problem here
Administrative Operations

• A protection system includes a set of methods for modifying that state

• UNIX and Windows
  ‣ Add right: If the user is the owner of the object, then the user can add an operation to set of operations of another user
  ‣ Also processes running as that user can perform “add right” using that user’s permissions

• Called discretionary access control because protection state is at user’s (and their process’s) discretion
Verifiability

• From reference monitor concept
  ‣ The *reference validation mechanism* must be *small enough* to be subject to analysis and tests, the completeness of which can be assured
• UNIX and Windows both consist of tens of millions of lines of code
  ‣ At present, no way to make testing complete
  ‣ Also, limited by lack of type safety, presence of assembly code, ad hoc changes to OS (loadable modules), etc.
• Not happening any time soon
Take Away

- **Goal**: Run your process security on a system with other potentially vulnerable processes
  - Need to protect *integrity, secrecy, availability* of OS resources

- **Approach**: Protection system
  - Protection state: Relates subjects, objects, operations to authorized permissions
  - Methods for modifying the protection state

- **Requirement**: Enforce protection state
  - Reference validation mechanism must satisfy reference monitor concept
    - Complete mediation, tamperproofing, verifiability

- **UNIX and Window implement protection systems**
  - But they do not satisfy the reference monitor concept
  - Provide “protection,” but not “security” (show later)