CS 6324: Information Security
More on Trusted Computing & Access Control

Dr. Junia Valente

Department of Computer Science
The University of Texas at Dallas

(contains some material from: Ariel Segall (MITRE), Dr. Zhiqiang Lin, Dr. Alvaro Cardenas)
Announcement:

- Quiz next **Tuesday, April 2**nd based on Lectures 17 & 18 (trusted computing only).
- Posted homework # 2 on eLearning (due on Friday, April 5 @ 10 a.m.)
More on Trusted Computing & Access Control

- Trusted Computing
- Access Control
- Before logging into a computer, I know the software running is benign.

- Machines that are not up-to-date will be routed to perform updates *before* connecting to a network.

- Servers can confirm exactly which machines they are talking to and whether those machines are running good software--*before* providing sensitive data.

- All of my data, including secret keys, are protected by hardware and cannot be stolen over the network.

*We’re not there yet, but we are moving in the right direction.*
Trusted Platform Module (TPM) summary:

- Last class we talked about the Trusted Platform Module

- In summary, TPM provides:
  - **Root-of-Trust for Reporting** and **Root-of-Trust for Storage**
  - **Limited internal storage:**
    - Platform Configuration Registers (PCRs)
    - Key storage
    - Data storage
  - Random number generation
  - Highly constrained **crypto functions**
**TPM historical perspective:**

- TPM 1.2 specifications released in October 2003
- Limited crypto
- **Targeted at PC**
  - Protection from malware from the network
  - Protection for stolen laptops
- Accepted as an ISO standard in 2009
- 100’s of millions have been shipped
- Current version TPM 2.0
What is TPM good for?  
Example applications: Machine Authentication

- **Machine Authentication:**
  - We can use TPM keys to reliably identify a machine
  - Keys are cryptographically bound to a particular TPM
  - Decryption-based authentication
    - example: “Only machine X can read this data”
  - Signing-based authentication
    - example: “This data passed through machine X, because X signed it.”
What is TPM good for?
Example applications: Attestation

- **Attestation:**
  
  The presentation of verifiable evidence about machine state to a remote party

- A remote verifier can check boot state of machine

- Potentially very powerful!
  
  - “Is this machine running the right image?”
  
  - “Is the software trustworthy?”
What is TPM good for?

Example applications: Data Protection

- **TPM is not** suitable for bulk data encryption
  - Too slow! Public key encryption only, cheap processor
  - No fast symmetric ciphers due to export regulations

- **Use to encrypt small, high-value data**; for example:
  - Symmetric keys usable for bulk encryption
  - Software-held private keys (e.g., user identities)
  - Password stores

- Can be used for hard drive encryption, if supported (e.g., Bitlocker)

- “Tamper resistance” hardware protection for sensitive data (more on this soon)
Myth: The TPM Controls Boot:

“TPMs can stop your machine from booting if bad software is running” <-- not true.

- It has no control over the rest of your machine; it's a purely passive device.
- Nor does it have any awareness of what’s happening on the system.
- The TPM can, in **highly controlled situations**, limit data access to only good software.
- High-security, predictable systems (designed with this in mind) can use the TPM to limit bad boots:
  - Bitlocker
  - TPM-enabled device encryption
Myth: The TPM is Tamper-Proof:

- TPMs are tamper-resistant…for consumer products.
- Tremendously good for their cost!
  - Cost < $1
  - Breaking cost researcher >$100,000; destroyed several in the process
- **Not** designed with government tamper-resistance standards in mind.
- ARM’s TrustZone
  - Two virtual processors backed by hardware-based access control
  - Application core can switch between two worlds:
    - Rich Execution Environment
    - and a Trusted Execution Environment
Intel Software Guard Extensions (SGX):

- SGX: a powerful architecture for managing secret data

- Extension to Intel processors that support:
  
  - **Enclaves**: for running code & having memory isolated from the rest of system
  
  - **Attestation**: prove to local or remote party what code is running in enclave
  
  - **Minimum Trusted Computing Base (TCB)**: processor is trusted, nothing else.
    
    - All **writes** to memory must be encrypted
  
  - Enables processing of data that cannot be read by anyone, except for code running in enclave.
Outline

- Trusted Computing

- ACCESS CONTROL
Basic concepts in systems security:

- How to get access to resources?
  - **Authentication** (password / crypto / etc.)
    - Who are you?
  - **Authorization** (Access control)
    - What are you allowed to do.
    - Focus is policy
  - **Enforcement Mechanism**
    - How it is policy implemented / enforced
Basic concepts in systems security:
Examples

- How to get access to resources?
  - **Authentication** (password / crypto / etc.) - ID Check
    - Who are you?
  - **Authorization** (Access control)
    - What are you allowed to do.
    - Focus is policy
  - **Enforcement Mechanism**
    - How it is policy implemented / enforced

- Over 18 allowed in
- On VIP list – allowed to access VIP area
- Walls, doors, locks, bouncers.
A Simple Example on Access Control:

![Windows Security Window]

The server www.utdallas.edu at System Security and Malicious Code Analysis requires a username and password.

Warning: This server is requesting that your username and password be sent in an insecure manner (basic authentication without a secure connection).

- cs6324
- **********
- Remember my credentials

[OK] [Cancel]
Popular Example of Access Control Policy:

![Facebook Privacy Settings](image-url)
Access Control:

- Control which **Principals** (persons, processes, machines) have access to which **Resources** in the system (which files they can read, programs they can execute, etc.)

- Use a **Reference Monitor** to enforce a policy

- Provide **Accountability** by keeping audit logs / and monitoring

- Implemented at different layers: Hardware, OS, Application, Networks
Access Control History:

- One of the first works in computer security (Lampson 1974)
- “Who can do what to whom in an OS?”
- **Subjects**: users or programs
- **Objects**: programs, files, directories, resources
### Access Control Matrix:

<table>
<thead>
<tr>
<th></th>
<th>$O_1$</th>
<th>...</th>
<th>$O_n$</th>
<th>$S_1$</th>
<th>...</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S_n$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Subjects $S = \{s_1, ..., s_n\}$

Objects $O = \{o_1, ..., o_m\}$

Rights $R = \{r_1, ..., r_k\}$

$A[s_i, o_j] = \{r_x, ..., r_y\}$ means subject $s_i$ has rights $r_x, ..., r_y$ over object $o_j$
### Access Control Matrix = Security Policy

<table>
<thead>
<tr>
<th>subjects</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>r</td>
<td>r/w</td>
<td>r</td>
<td>-</td>
</tr>
<tr>
<td>bob</td>
<td>r</td>
<td>r</td>
<td>-</td>
<td>r/w</td>
</tr>
<tr>
<td>charlie</td>
<td>-</td>
<td>-</td>
<td>w</td>
<td>-</td>
</tr>
<tr>
<td>dave</td>
<td>r/w</td>
<td>-</td>
<td>w</td>
<td></td>
</tr>
</tbody>
</table>
Simplifying the Access Control Matrix:

- A general matrix is far too difficult to manage
- Compressing the rights
  - Take the columns and associate each object to a list: **Access Control List**
  - Take the rows and associate each subject to a list (based on what it can do).
    We call the elements of the list **capabilities** or **tickets**
- Compressing the users:
  - groups or roles as subjects: **role-based access control**
Access Control List Example:
Bouncer has a list and enforces entrance

Capabilities:
If you are wearing a bracelet, they let you in
Simplifying the Access Control Matrix:

1) Access Control List

- **Access Control List** (ACL)

  - each object has a set of <user, right> tuples

  - e.g., object B { <alice, r/w>, <bob, r>, <dave, r/w> }

- Properties:

  - Good for many applications (file systems)

  - Can grow quite large

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>alice</td>
<td>r</td>
<td>r/w</td>
<td>r</td>
<td>-</td>
</tr>
<tr>
<td>bob</td>
<td>r</td>
<td>r</td>
<td>-</td>
<td>r/w</td>
</tr>
<tr>
<td>charlie</td>
<td>-</td>
<td>-</td>
<td>w</td>
<td>-</td>
</tr>
<tr>
<td>dave</td>
<td>r/w</td>
<td>-</td>
<td>w</td>
<td>-</td>
</tr>
</tbody>
</table>
```
1) Access Control Lists (ACL):

- The basic access control mechanism in Unix-based systems (Linux, OS X, etc.)

- Access Control Lists are great where access control is set up by a central authority

- Not good when users want to delegate their rights, or where the user population is large or changing

- Not good for revocation or for checking system-wide the access rights of all users (as it would require to go through millions of files)
Reference Monitor:

- Provides a centralized access control
- Designed to **enforce** an access control policy over subjects’ ability to perform operations (e.g., read and write) on objects (e.g., files) on a system.
1) Access Control Lists (ACL):

![Access Control List Diagram]

ACL Owner: /usr/bin/foobar

<table>
<thead>
<tr>
<th>Key</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jeanne</td>
<td>Read, Write, Execute</td>
</tr>
<tr>
<td>Joe</td>
<td>Read, Write</td>
</tr>
<tr>
<td>Jim</td>
<td>Read, Change Icon Color</td>
</tr>
<tr>
<td>Users</td>
<td>Read</td>
</tr>
<tr>
<td>Admins</td>
<td>read, Write, Execute, Change Icon Color</td>
</tr>
</tbody>
</table>
1) Access Control Lists (ACL): Unix family

- Each file (and program) has an **owner** and a **group**
- Each file has three types of permission: **read**, **write**, and **execute**

```
-rw-r--r-- 1 <owner> <group> 64 Mar 25 21:46 foobar
```

- If A asks to operate on a file, the operating system checks in order:
  - If A is the file owner, then the **owner permission set** decides whether A can carry out the operation
  - If A is a member of the file’s group, then the **group permission set** decides if A can carry out the operation
  - Otherwise, the **other permission set** (i.e., everyone else) decides.
1) Access Control Lists (ACL): Unix Permissions

```
-rw-rw-r--  1 pbg  staff  31200  Sep 3 08:30  intro.ps
drwx------  5 pbg  staff  512   Jul 8 09:33  private/
drwxrwxr-x  2 pbg  staff  512   Jul 8 09:35  doc/
-drwxrwx---  2 jwg  student 512  Aug 3 14:13  student-proj/
-rw-r--r--  1 pbg  staff  9423  Feb 24 2012  program.c
-rwxr-xr-x  1 pbg  staff  20471  Feb 24 2012  program
-drwx--x--x  4 tag  faculty 512  Jul 31 10:31  lib/
-drwx------  3 pbg  staff  1024  Aug 29 06:52  mail/
-drwxrwxrwx  3 pbg  staff  512   Jul 8 09:35  test/
```
- **Capabilities**

  - break the matrix down by rows: user \{<object, rights>\} tuples
  - e.g., Alice \{ <A, r>, <B, r/w>, <C, r> \}

- Properties:
  - Natural model for delegation (the rights are coupled to object)
2) Access Control Lists (ACL): Capabilities

- Capability of “Alice” is her row on the access control matrix
- It is easier to create more complex policies (e.g., “Alice” can delegate rights to other subjects)
- It is harder to find who can modify files, etc.
- Capabilities are becoming more important recently
Simplifying the Access Control Matrix:

3) Compressing the users

- Subjects

  - Groups:

    e.g., staff = \{alice, dave\}; students = \{bob, charlie\}

- Objects

  - Types:

    e.g., system_file = \{A, B\}, user_file = \{C, D\}
3) Compressing the users: Role-based access control

Role-based access control (RBAC)

- Instead of defining rules for 1,200 individuals (for example).

We can define rules for roles

- E.g., in a Bank we can have: teller, branch accountant, branch manager, etc.

- Only a few needs individual access rights
Modern Trends:

- The role of OS security has shifted from protecting multiple users from each other, toward protecting a single user from untrustworthy applications.

- **Now**: interested of consumers, phone vendors, application authors, online services, etc. must be mediated by OSes that were designed for another place and time.

- E.g., classic Unix model failed to meet the needs of ISPs, firewalls, smartphones, etc.

- **Need**: access-control extensibility, catering to diverse requirements.
  - Must support many policy models.
Discretionary Access Control:

- **Discretionary Access Control (DAC)**
  
  Similar to access control lists in Unix files, allows object owners to protect (or share) objects at their own discretion.

- **Mandatory Access Control (MAC)**
  
  Enforces systemwide security invariants, regardless of user preference.

  Considered costly to implement in operating systems (for decades used only in military systems) but now implemented in Apple OS X, Apple iOS, FreeBSD, etc.
Historical Examples of MAC:

- First foundational results in security
- Resulted in the Orange Book (1983): standards to assess the effectiveness of computer security controls in a computer system
- Used to select systems being considered to process classified information
- Replaced by the common criteria standard in 2005