CIS 433/533 - Computer and Network Security
Roles, MLS, Trusted Computing

Prof. Kevin Butler
Winter 2013
Access Control

• What’s a discretionary access control system?

• What’s a mandatory access control system?
  ‣ What’s a protection system?

• What is the difference in terms of access policy?
  ‣ Now?
  ‣ In the future?
Safety Problem

• For a protection system
  ‣ (ref mon, protection state, and administrative operations)

• Prove that any future state will not result in the leakage of an access right to an unauthorized user
  ‣ Q: Why is this important?

• For most discretionary access control models,
  ‣ Safety is \textit{undecideable}

• Means that we need another way to prove safety
  ‣ Restrict the model (no one uses)
  ‣ Test incrementally (constraints)

• How does the safety problem affect MAC models?
Access Control Models

• What language should I use to express policy?
  ‣ Access Control Model

• Oodles of these
  ‣ Some specialize in secrecy
    • Bell-LaPadula
  ‣ Some specialize in integrity
    • Clark-Wilson
  ‣ Some focus on jobs
    • RBAC
  ‣ Some specialize in least privilege
    • SELinux Type Enforcement

• Q: Why are there so many different models?
Groups

• Groups are collections of identities who are assigned rights as a collective

• Important in that it allows permissions to be assigned in aggregates of users …

• This is really about “membership”
  ‣ Standard DAC
  ‣ Permissions are transient
Job Functions

• In an enterprise, we don’t really do anything as ourselves, we do things as some job function
  ‣ E.g., student, professor, doctor

• One could manage this as groups, right?
  ‣ We are assigned to groups all the time, and given similar rights as them, i.e., mailing lists
Roles

- A **role** is a collection of privileges/permissions associated with some function or affiliation.
- NIST studied the way permissions are assigned and used in the real world, and this is it …
- Important: the permissions are static, the user-role membership is transient.
- This is not standard DAC.
Role Based Access Control

• Role based access control is a class of access control not direct MAC and DAC, but may one or either of these.
• A lot of literature deals with RBAC models
• Most formulations are of the type
  ‣ $U$: users -- these are the subjects in the system
  ‣ $R$: roles -- these are the different roles users may assume
  ‣ $P$: permissions --- these are the rights which can be assumed
• There is a many-to-many relation between:
  ‣ Users and roles
  ‣ Roles and permissions
• Relations define the role-based access control policy
RBAC Sessions

• During a **session**, a user assumes a subset available roles
  ‣ Known as **activating** a set of roles
  ‣ The user rights are the union of the rights of the activated roles
  ‣ Note: the session **terminates** at the user’s discretion

• **Q**: Why not just activate all the roles?
Constraints

• You want to constrain evolution of protection states
  ‣ Constraints are explicit ways of doing just this
  ‣ Constraints available (in RBAC)
    • role assumption
    • perm-role assignment
    • user-role assignment

• Examples in RBAC:
  ‣ Required inclusion: You must be acting as an employee of the University of Oregon to be a professor
    • You must assume a (parent) role to assume another (child) role
  ‣ Mutual exclusion: can not be both CFO and auditor for the same company (unless you work for Enron)
  ‣ Cardinality constraint: only one (or n) of a particular role
Constraint Example

- **Mutual Exclusion**: No entity can activate student and faculty roles at the same time?
  - Give yourself credits, etc.
  - Or, in this case buy faculty tickets at student prices?
Multilevel Security

• A multi-level security system tags all object and subject with security tags classifying them in terms of sensitivity/access level.
  ‣ We formulate an access control policy based on these levels
  ‣ We can also add other dimensions, called categories which horizontally partition the rights space (in a way similar to that as was done by roles)
US DoD Policy

• Used by the US military (and many others), the Lattice model uses MLS to define policy

• Levels:

  UNCLASSIFIED < CONFIDENTIAL < SECRET < TOP SECRET

• Categories (actually unbounded set)

  NUC(lear), INTEL(igence), CRYPTO(raphy)

• Note that these levels are used for physical documents in the governments as well.
Assigning Security Levels

• All subjects are assigned clearance levels and compartments
  ‣ Alice: (SECRET, \{CRYPTO, NUC\})
  ‣ Bob: (CONFIDENTIAL, \{INTEL\})
  ‣ Charlie: (TOP SECRET, \{CRYPTO, NUC, INTEL\})

• All objects are assigned an access class
  ‣ DocA: (CONFIDENTIAL, \{INTEL\})
  ‣ DocB: (SECRET, \{CRYPTO\})
  ‣ DocC: (UNCLASSIFIED, \{NUC\})
Evaluating Policy

• Access is allowed if

subject clearance level >= object sensitivity level and
subject categories ⊇ object categories (read down)

• Q: What would write-up be?
Bell-La Padula Model

- A Confidentiality MLS policy that enforces:
  - *Simple Security Policy*: a subject at specific classification level cannot read data with a higher classification level. This is shorthand for “no read up”.
  - *(star) Property*: also known as the confinement property, states that subject at a specific classification cannot write data to a lower classification level. This is shorthand for “no write down”.

```
(Top Secret, {nuclear, crypto})

(Top Secret, {nuclear})
(Top Secret, {})   (Top Secret, {crypto})

(Secret, {nuclear, crypto})
(Secret, {nuclear})
(Secret, {crypto})

(Secret, {crypto})
```
How about integrity?

• MLS as presented before talks about who can “read” a document (confidentiality)

• Integrity considers who can “write” to a document
  ‣ Thus, who can effect the integrity (content) of a document
  ‣ Example: You may not care who can read DNS records, but you better care who writes to them!

• Biba defined a dual of secrecy for integrity
  ‣ Lattice policy with, “no read down, no write up”
    • Users can only create content at or below their own integrity level (a monk may write a prayer book that can be read by commoners, but not one to be read by a high priest).
    • Users can only view content at or above their own integrity level (a monk may read a book written by the high priest, but may not read a pamphlet written by a lowly commoner).
Integrity, Sewage, and Wine

• Mix a gallon of sewage and one drop of wine gives you?
• Mix a gallon of wine and one drop of sewage gives you?

*Integrity is really a contaminant problem:* you want to make sure your data is not contaminated with data of lower integrity.
Biba (example)

• Which users can modify what documents?
  ‣ Remember “no read down, no write up”

Bob: (CONF., {INTEL})

Charlie: (TS, {CRYPTO, NUC, INTEL})

Alice: (SEC., {CRYPTO, NUC})

DocA: (CONFIDENTIAL, {INTEL})

DocB: (SECRET, {CRYPTO})

DocC: (UNCLASSIFIED, {NUC})
• **Low-Water Mark integrity**
  ‣ Change integrity level based on actual dependencies
• Subject is initially at the highest integrity
  ‣ But integrity level can change based on objects accessed
• Ultimately, subject has integrity of lowest object read
Clark-Wilson Integrity

• Map Integrity in Business (e.g., accounting) to Computing
• High Integrity Data (objects)
  ‣ “Constrained Data Items” (CDIs)
• High Integrity Processes (programs)
  ‣ “Transformation Procedures” (TPs)
• Check Integrity of Data Initially (verification)
  ‣ “Integrity Verification Procedures” (IVPs)
• Premise
  ‣ If the IVPs verify initial integrity
  ‣ and high integrity data is only modified by TPs
  ‣ Then, the integrity of computation is preserved
Clark Wilson Permissions

User → User
User → User
User → User
User → User

CDI → CDI
CDI → CDI
CDI → CDI
CDI → CDI
CW Permissions (cont.)
CW Permissions (cont.)

- A user can access an CDI using TP iff
  1. The user has been granted CDI access
  2. The TP has been granted CDI access
  3. The user has been granted access to the TP
Clark-Wilson Issues

- Assure Function
  - Certify IVPs, TPs to be ‘valid’ (i.e., correct) (C1,C2)
  - Is there a general way of defining correctness?

- Handle Low Integrity Data
  - A TP must upgrade or discard any UDI (low integrity data) it receives (C5)

*Reality*: this is a nice model, but too heavyweight in general for most applications. CW-lite (Jaeger) is an alternative that is tractable to implement.
Security Typed Languages

• Key:
  ‣ tag data & monitor flows
  ‣ e.g., language: Jif

• RMs tag actual data
  ‣ all data/processes have label
  ‣ central security monitor checks operations, data access against policy

• Security-typed languages use virtual tags
  ‣ data types are labeled
  ‣ type checker validates flows
Build on type safety

• A type-safe language maintains the semantics of types. E.g. can’t add int’s to Object’s.

• Type-safety is compositional. A function promises to maintain type safety.

Example 1
Object obj;
int i;
obj = obj + i;

Example 2
String proc_obj(Object o);
...
main()
{
  Object obj;
  String s = proc_obj(obj);
  ...
}
Labeling types

Example 1

```c
int{high} h1,h2;
int{low} l;
l = 5;
h2 = l;
h1 = h2 + 10;
l = h2 + l;
```

X

- Key insight:
  - label types with security levels

- Security-typing is compositional

Example 2

```c
String{low}
proc_obj(Object{high} o);
...
main()
{
  Object{high} obj;
  String{low} s;
  s = proc_obj(obj);
  ...
}
```
public class SecretMessages[principal alice, principal bob] {

    String{alice:} aliceInstructions;
    String{bob:} bobInstructions;

    public SecretMessages(String{alice:} ai, String{bob:} bi) {
        aliceInstructions = ai;
        bobInstructions = bi;
    }

    ...

    public String{bob:} leak() {
        bobInstructions = aliceInstructions;
        return bobInstructions;
    }

}
public class SecretMessages[\texttt{label alice, label bob}] {
    String{*alice} aliceInstructions;
    String{*bob} bobInstructions;

    public SecretMessages(String{*alice} ai, String{*bob} bi) {
        aliceInstructions = ai;
        bobInstructions = bi;
    }

    public String{*bob} implicitLeak() {
        try {
            if (aliceInstructions.equals("Attack at dawn"))
                bobInstructions = "Attack at dawn";
        }
        catch (NullPointerException e) {}
        return bobInstructions;
    }
}
Declassification

• MLS is too restrictive
• Examples:
  ‣ Encryption
  ‣ Distributed auction
  ‣ Password check
• Solutions:
  ‣ Declassification
    • Reduce the level of data -- tolerable leakage
Problem

• My computer is running a process
• It makes a request to your computer
  ‣ Asks for some secret data to process
  ‣ Provides an input you depend on
• How do you know it is executing correctly?
• Example
  ‣ ATM machine is uploading a transaction to the bank
  ‣ How does the bank know that this ATM is running correctly, so the transaction can be considered legal?
What would you do?

- Nothing
- Proof by authority (*Certificates*)
  - Tells you who, but not what
- Constrain the system (*Secure Boot*)
  - Effective, limiting, but proof is implied
- Inspect the runtime state (*Authenticated Boot*)
  - Flexible, attestable, but difficult to prove semantics
Secure Boot

- Check each stage in the boot process
  - Is code that you are going to load acceptable?
  - If not, terminate the boot process

- Must establish a **Root-of-Trust**
  - A component trusted to speak for the correctness of others
  - Assumed to be correct because errors are **undetectable**
Trusted(?) Computing

- The Trusted Platform Module (TPM) brought authenticated boot into the main stream
- Essentially, the TPM offers few primitives
  - Measurement, cryptography, key generation, PRNG
  - Controlled by physical presence of the machine
  - BIOS is Core Root of Trust for Measurement (CRTM)
- Spec only discussed how to measure early boot phases and general userspace measurements
• The **Trusted Platform Module** is a tamper-resistant secure microcontroller.

  ‣ Manages cryptographic keys and functionality it uses to support security relevant operations.
  ‣ Measures the code loaded by the system (firmware, BIOS, OS kernel, device drives, application processes, ...)

• Measurements are hashes of loaded code (PCRs)
Authenticated Boot

• A lot of FUD was generated around what it does & doesn’t do

• Paladium/NGSCB architecture (Microsoft, 2002)
  ‣ Use virtualization to split system
  ‣ Measure the “trusted” part to prove its integrity before responding

• “Meet the emerging requirements of an interconnected world” – Microsoft

• Take over the world – Ross Anderson and others
Linux Integrity Measurement

- **Problem**: How can we verify the software environment of networked systems?
- **Solution**: Extend TPM measurement architecture to measure system’s runtime (Software Stack)
- **Additional Goals**
  - Load-time integrity
  - Unobtrusive
  - *Tamper-evident*
  - Usability
Integrity Measurement

- Means used to determine the state of the host
- Relies on *measurement* (i.e., hash fingerprinting of the code)
  - Hardware support emanates from the core *root of trust for measurement* (CRTM), secured on the host
  - Subsequent measured steps: BIOS, bootloader (stage 1 & 2), OS
- Attestation of the code is performed with TPM *Quote* operation
Limitations

- **Static** root of trust for measurement (*reboot*)
- Coarse-grained, measures entire system
  - Requires hundreds of integrity measurements just to boot
  - Every host is different
    - firmware versions, drivers, patches, apps, spyware, …
  - What does a PCR mean in this context?
  - TCB includes entire system!
- Integrity measurements are done at **load-time**, not at run-time
  - Time-of-check-time-of-use (TOCTOU) problem
  - Cannot detect any dynamic attacks
  - No guarantee of execution
DRTM (Late Launch)

• *Dynamic* Root of Trust for Measurement
• Involves both CPU and TPM v1.2
• Security properties similar to reboot... without a reboot
• Removes many things from TCB
  • BIOS, boot loader, DMA-enabled devices, …
  • Long-running OS and Apps if done right
• When combined with virtualization
  ‣ VMM can be measured (MVMM), potentially lengthy uptimes
  ‣ Integrity of loaded code can be attested
  ‣ Untrusted legacy OS can coexist with trusted software
• Allows introduction of new, higher-assurance software without breaking existing systems