Complex Access Control

Access Control Matrix

- List all proceses and files in a matrix
- Each row is a process (“subject”)
- Each column is a file (“object”)
- Each matrix entry is the access rights that subject has for that object

Sample Access Control Matrix

Subjects p and q
Objects f, g, p, q
Access rights r (read), w (write), x (execute), o (owner)

	f	g	p	q
p	rwo	r	rwx	w
q	-	r	r	rwxo

Other Permissions

- Append
- Delete file
- Owner (can change ACL)
- Many more are possible

Access Control Matrix Operations

- System can transition from one ACM state to another
- Primitive operations: create subject, create object; destroy subject, destroy object; add access right; delete access right
- Transitions are, of course, conditional

Conditional ACM Changes

Process *p* wishes to give process *q* read access to a file *f* owned by *p*.

```
command grant_read_file(p, f, q)
    if o in a[p, f]
    then
        enter "r" into a[q, f]
    else
        (signal error condition)
    fi
end
```

Safety versus Security

- *Safety* is a property of the abstract system
- *Security* is a property of the implementation
- To be secure, a system must be safe *and* not have any access control bugs

Access Control Formalisms

- Access control can be modeled formally. What does this buy us?
- There are theorems that can be proved
- For example, if ACLs permit negation there are undecidable questions

Access Control Formalisms (cont.)

- For the general case:
- Model using a Turing machine.
- Turing machine enters a special state if the access control is faulty.
- Contradiction!

An Undecidable Question About Access Control

Query: given an ACM and a set of transition rules, will some access right ever end up in some cell of the matrix?

- Model ACM and transition rules as Turing machine
- Machine will halt if that access right shows up in that cell
- Will it ever halt?
- Clearly undecidable
- Conclusion: We can never tell if an access control system is safe (Harrison-Ruzzo-Ullman (HRU) result)

Will This Program Halt?

```
main(int argc, char *argv[])
{
        return 0;
}
```

We can *sometimes* tell if a program will do a certain thing—but we can't *always* tell.

Conclusion: we cannot prove the safety, let alone the security, of an arbitrary system.

Complex Access Control

- Simple user/group/other or simple ACLs don’t always suffice
- Some situations need more complex mechanisms

Temporal Access Control

- Permit access only at certain times
- Model: time-locks on bank vaults

Implementing Temporal Access Control

- Obvious way: add extra fields to ACL
- Work-around: timer-based automatic job that changes ACLs dynamically

Problems and Attacks

- Is your syntax powerful enough for concepts like holidays? On what calendar? Do you support all relevant religious calendars? When is Eid ul Fitr next year? (When was it this year?)
- What time zone are employees in? Do any of them travel to other time zones?
- What if the clock is wrong?
- Can the enemy change the clock?
- How is the clock set? By whom or what?

Time Protocols

```
yellowstone.ntp > time.nist.gov.ntp: NTPv4 client, strat 0
time.nist.gov.ntp > yellowstone.ntp: NTPv4 server, strat 1
yellowstone.ntp > meow.febo.com.ntp: NTPv4 client, strat 0
meow.febo.com.ntp > yellowstone.ntp: NTPv4 server, strat 2
```

Changing the ACL

- Who changes it?
- What are the permissions on the clock daemon's tables?
- Is there a race condition at permission change time?
- What if the daemon's tables get out of sync with reality? Suppose a new file or directory is added?
- We have introduced new failure modes!

Role-Based Access Control

- Permissions are granted to *roles*, not users
- Map users to roles
- David Wheeler: “Any software problem can be solved by adding another layer of indirection”
- Mapping can change; should be reasonably dynamic
- Example: substitute worker; replacement worker

Using RBAC

- RBAC is the mechanism of choice for complex situations
- Often, it isn't used where it should be, because it's more complex to set up.
- Example: giving your administrative assistant your email password
- Does this create new weaknesses?

Using RBAC

- RBAC is the mechanism of choice for complex situations
- Often, it isn't used where it should be, because it's more complex to set up.
- Example: giving your administrative assistant your email password
- Does this create new weaknesses?

☞ New attack: corrupt the mapping mechanism between users and roles

A Completely Different Approach

- Traditional access control protects *files*
- Governments want to protect *information*

Discretionary versus Mandatory Access Control

- Discretionary access control (DAC): file owners can grant permissions
- Mandatory access control (MAC): someone else has that authority
- (Sometimes called the *site security officer*)

US Government Classification Model

- Documents are classified at a certain level
- People have certain clearances
- You’re only allowed to see documents that you’re cleared for

Classifications

- Levels: Confidential, Secret, Top Secret
- Compartments: Crypto, Subs, Planes, . . .
- To read a document, you must have at least as high a clearance level *and* you must be cleared for each compartment
- Systems that support this are known as *multi-level security* (MLS) systems

Examples

Pat is cleared for **Secret**, *Subs*
Chris is cleared for **Top Secret**, *Planes*

We have the following files:

`warplan`	**Top Secret**	*Troops, Subs, Planes*
`runway`	**Confidential**	*Planes*
`sonar`	**Top Secret**	*Subs*
`torpedo`	**Secret**	*Subs*

Who can read which file?

Examples

- Pat cannot read `warplan`; they aren't cleared high enough and they don't have *Troops* or *Planes* clearance
- Chris can't read it, either; he doesn't have *Troops* or *Subs* clearance
- Chris can read `runway`; Pat can't
- Pat can't read `sonar`; they has *Subs* clearance but only at the **Secret** level
- They can, however, read `torpedo`

Comparing Clearances

- Who has a higher clearance, Chris or Pat?
- Which is higher, $\langle$**Secret**, *Subs*$\rangle$ or $\langle$**Top Secret**, *Planes*$\rangle$
- Neither—they aren't comparable

Formally Comparing Labels

- A label is the tuple $\langle L, C \rangle$, where L is the hierarchical level and C is the set of compartments
- $S \geq O$ if and only if $L_S \geq L_O$ and $C_S \supseteq C_O$

Lattices

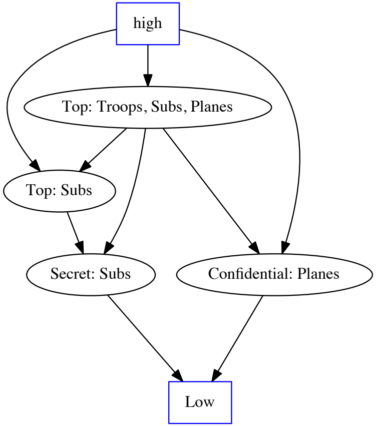
- Clearances here are represented in a *lattice*
- A lattice is a directed graph
- We say that label A *dominates* label B if there is a valid path down from A to B
- Expressed differently, if A dominates B, information is allowed to flow from B to A. We write $B \leq A$.
- Known as the Bell-LaPadula model

Properties of Lattices

- Lattices are a *partial ordering*
- Lattice domination is transitive, reflexive, anti-symmetric:

Transitivity	If $C \leq B$ and $B \leq A$, then $C \leq A$
Reflexivity	$A \leq A$
Anti-symmetry	$B \leq A$ and $A \leq B$ implies $A = B$

A Sample Lattice



⟨**Top**, *Subs*⟩ dominates ⟨**Secret**, *Subs*⟩

⟨**Top**, *Troops*, *Subs*, *Planes*⟩ dominates ⟨**Confidential**, *Planes*⟩, ⟨**Top**, *Subs*⟩,
and ⟨**Secret**, *Subs*⟩

⟨**Confidential**, *Planes*⟩ and ⟨**Secret**, *Subs*⟩ are not comparable,

Using this Scheme

- Processes are *subjects*
- Files are *objects*
- A process can read a file if its label dominates the file's label
- Known as "no read up"
- File labels are typically subject to mandatory access control

Writing Files

- Suppose there are three labels, *A*, *B*, and *C*, such that *A* dominates *B* and *B* dominates *C*
- A process with label *A* can read a file with label *B* or label *C*. A process with label *C* can read a file labled *C* but not *B*
- Suppose that a process with label *A* reads *B* and then writes the contents to a file labeled *C*.
- Can a *C*-labeled process now read this?
- No—a process can only write to a file if the file's label dominates it
- Known as "no write down"; either the file's label must change or the write must be disallowed

(Network Communication)

- Network communication is almost always bidirectional
- ☞ In TCP, ACK packets go back the other way
- Conclusion: if network packets are labeled (and they can be), communication is only permissible if the labels match *exactly*

A Problem with “No Write Down”

- Should a process at **Confidential** be able to overwrite a **Top Secret** file?
- Is that an attack on availability?
- The usual practice is that a process can only write to a file whose label is an exact match

Formal Version

Simple Security Condition S can read O if and only if $l_o \leq l_s$

*-property S can write O if and only if $l_s \leq l_o$

Basic Security Theorem If Σ is a system with secure inital state σ_0 and T is a set of state transitions that preserve the simple security condition, every state $\sigma_i, i \geq 0$ is secure

Combining MAC and DAC

- The Bell-LaPadula model includes DAC as well as MAC
- Users control DAC settings; the site security officer controls the MAC values
- To read or write a file, both MAC and DAC conditions must be satisfied

Confidentiality versus Integrity

- This scheme is geared towards confidentiality
- We can use it for integrity, too
- Make sure that all system files are labeled **Low**
- All labels dominate **Low**
- Thus, no process can write to it (“no write down”)
- Overwriting a system file appears to the access control mechanism as a confidentiality violation!
- Known as Biba integrity

Thinking Semantically

- Simpler permission schemes protect *objects*
- Bell-LaPadula schemes protect *information*
- Information flow is a dynamic concept

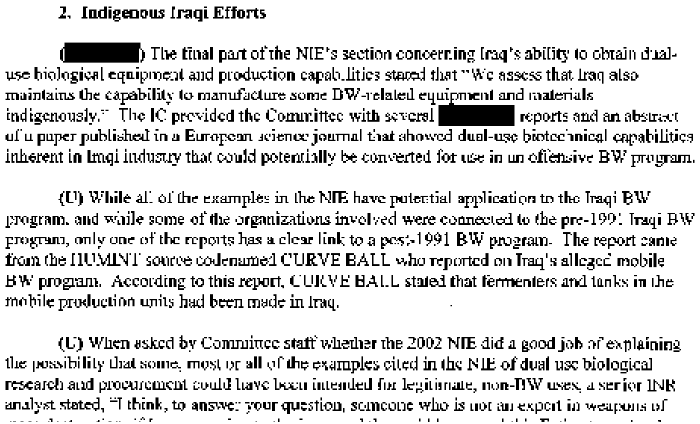
Implementing Bell-LaPadula

- Does anyone actually use this stuff?
- First implemented in Multics
- Available today some versions of Linux
- Part of many DoD-certified systems
- But—such systems are rarely used outside of DoD, and not often within it
- The assurance process is too slow and expensive

Exporting Labels

- Labels have to stay with the data
- Transmitted in network packets
- Printed on output
- Recorded on CDs, etc.
- What happens if a labeled CD is physically carried to—and from—a non-MLS (or otherwise untrusted) machine?

They Even Mark Classified Documents



2. Indigenous Iraqi Efforts

([redacted]) The final part of the NIE's section concerning Iraq's ability to obtain dual-use biological equipment and production capabilities stated that "We assess that Iraq also maintains the capability to manufacture some BW-related equipment and materials indigenously." The IC provided the Committee with several [redacted] reports and an abstract of a paper published in a European science journal that showed dual-use biotechnical capabilities inherent in Iraqi industry that could potentially be converted for use in an offensive BW program.

(U) While all of the examples in the NIE have potential application to the Iraqi BW program, and while some of the organizations involved were connected to the pre-1991 Iraqi BW program, only one of the reports has a clear link to a post-1991 BW program. The report came from the HUMINT source codenamed CURVE BALL who reported on Iraq's alleged mobile BW program. According to this report, CURVE BALL stated that fermenters and tanks in the mobile production units had been made in Iraq.

(U) When asked by Committee staff whether the 2002 NIE did a good job of explaining the possibility that some, most or all of the examples cited in the NIE of dual use biological research and procurement could have been intended for legitimate, non-BW uses, a senior INR analyst stated, "I think, to answer your question, someone who is not an expert in weapons of

Note the blacked-out security label on the the per-paragraph classificatiosn. Note also that the blacked-out classification label occupies a space too long for “S” or “TS”, and hence presumably gives a compartment. . .

The Commercial Uselessness of Bell-LaPadula

- Most commercial data isn’t as rigidly classified as is military data
- Few commercial operating systems support it
- It’s hard to transfer labels across networks, among heterogeneous systems
- *Downgrading* is hard

Downgrading Information

- Suppose we have a web server as a front end for a sensitive database
- We can label the database **Top Secret**
- To read it, the web server needs to have **Top Secret** privileges
- But the end user—the web client—isn't trusted to that level
- Where does the downgrade operation take place?
- Downgrade is a *very* sensitive operation and can only be done by a trusted module. Is your web server that trusted?

Program-Based Control

- Sometimes, there’s no general enough model
- There are constraints that cannot be expressed in any table
- Common example: some forms of digital rights management (DRM),
- It requires a program

Digital Rights Management

- Allow publisher to control use of content
- Prevent arbitrary redistribution of copyrighted materials
- Change sales terms from *physical purchase* to *license*

Many Different Types!

☞ Different content owners have different demands

- Different restrictions
- Different policies
- Example: policies which may include forcing a user to scroll through a license agreement and then click “yes”

Apple’s iTunes

- “personal, noncommercial use”
- “five Apple-authorized devices at any time”
- “shall not be entitled to burn video Products or ring tone Products”
- “burn an audio playlist up to seven times”
- (`http://www.apple.com/support/itunes/legal/terms.html`)

Microsoft’s Media DRM

- Media files are encrypted, and contain pointer to license source
- User obtains license from clearing house
- License includes terms and conditions as well as decryption key
- “Licenses can have different rights, such as start times and dates, duration, and counted operations.”
- “may allow the consumer to . . . copy the file to a portable device”
- “Licenses, however, are not transferable.”
- (http://www.microsoft.com/windows/windowsmedia/howto/articles/drmarchitecture.aspx)

All Bets are Off

- Is the program correct?
- Is it secure?
- Who wrote it?
- Who can change it?
- Who can change its data or configuration files?
- Does it do what you want?

Questions?

(Ruby-throated hummingbird, Central Park, September 20, 2019)