Protection and OS security

Admin

Administrivia: readings on the web page.

`http://www.cs.berkeley.edu/~daw/cs261/`

Definitions

Computer security:

Techniques for computing in the presence of adversaries.

Cryptography:

Techniques for communicating in the presence of adversaries.

Definitions

Protection:

> The subset of computer security which deals with controlling access to information stored locally.

Traditionally excludes issues arising from networks.

Traditionally associated with operating systems, and in particular with centralized mechanisms.
Note that the typical role of an OS is to manage multiplexed access to resources; protection is about access control, but not more general resource management issues.

Protection

Three categories of security goals:

- Confidentiality: preventing unauthorized release of information
- Integrity: preventing unauthorized modification of information
- Availability: preventing denial of service attacks

Protection is about providing all of these in the setting of a single machine.

Protection

The protection problem can be approximately factored into three sub-problems:

- Access control: enforcement of restrictions on who may access what
- Policy selection: a human-level statement of goals for the security subsystem
- Policy specification: human-computer interface

Access control

The access control problem is to decide whether a given request should be granted.

Can be conceptually divided into two steps

- Authentication: verifying the identity of an entity making a request.
- Authorization: verifying whether the policy allows the request.

Policy

Policy:

- A specification of the desired security properties of the system
- Roughly speaking, the goals which security mechanisms are intended to achieve
- "Human intent"
- Thus, policy cannot be automated; it is a fundamentally human-centered decision about "what ought to be"
- A good rule of thumb for formulating computer security policy is that it should roughly correspond to real-world policies
  - For example, at a bank, two officers may be required to both authorize the opening of a bank vault

- Further constraints might be placed (i.e. that the vault may only be opened between the hours of 9:00–3:00)
- If documents are moved from the vault and stored on bank computers, the computer security policy should probably uphold those same properties

Subjects and objects

A useful way of thinking about access control is in terms of *subjects* and *objects*.

A *subject* is an entity making a request. (An executing program, a user sitting at a terminal, etc.) Also called a *principal*. In Unix, this is just a user-id.

An *object* is an item to which access is requested. (A byte in memory, a file on disk, a printer, etc.)

Subjects and objects, cont.

Then, the policy specification can be translated into an *access control matrix*.

The rows of the matrix are labelled by the subjects, and the columns by the objects. The entry at row S and column O specifies whether S may access O.

In this language, we see that authentication is the problem of identifying the subject making the request, and checking authorization just comes down to checking a matrix entry.

Access control, cont.

Usually, "all-or-nothing" access control is insufficient; we want more fine-grained controls.

In this case, we may establish a set of *permissions*. (For example: read, write, execute, etc.) Then, the matrix entry at location (S, O) is filled with the set of permissions which S may acquire on O.

This provides a way to view the system which is convenient for analysis. For instance, suppose we want to find whether there is any way which a subject S may send a message to another subject S'. Then we may simply search for all objects O which S has write permission to, and check whether S' has read access to them.

Mandatory access control

Discretionary access control: owner of each object gets to set the permissions on that object.

Mandatory access control (or MAC): system administrator gets some control over the access controls on every object. Usually found in military systems (e.g. anything marked "classified" may not be sent to a printer in an unclassified facility). We'll discuss this in detail next week.

Objects, cont.

What are objects in real life?

Some examples: pages of memory, files on a storage system, I/O devices, communications channels to other subjects.

Access control lists

One typical way to implement access control is via *access control lists*, or ACLs.

In this approach, each object is tagged with an ACL. The ACL identifies a list of subjects S which may access the object. For each subject S, the ACL lists the permissions S has on the object.

An example:

```
allow read,write Alice,Jim
allow read John,Jim,Bob
allow delete Alice
deny read Eric
```

Capabilities

Capability: An unforgeable "ticket" which presents proof that the presenter is authorized to access the specified object.

Example:

$$(O, \text{read})$$

means the holder of this capability has read permissions on object O.

Permissions may be further restricted, e.g. by including an expiration date, etc.

Capabilities, cont.

Example implementations:

- Some old computer architectures supported capabilities in hardware. The memory system provided the unforgeability properties by tagging each memory word; only special privileged processes (e.g. the OS kernel) could create new capabilities.
- More commonly, enforced in the OS: the OS keeps track of all your capabilities for you, and manipulating them requires a system call.
- Pointer solutions: a capability is a pointer to the object in question. The address space is so large that one cannot guess the pointer by pure trial and error.
- Cryptographic solutions: a capability is a ticket encrypted under a key only the OS knows.

- Safe languages: a type-safe language gives capabilities a different type and enforces strict rules on the use of that type.

Comparing ACLs and capabilities

Access control decisions:

- ACLs: made at time of each access request (late binding).
- Capabilities: made once when capability is granted (early binding).
- Thus, capabilities potentially allow much higher performance by caching access control decisions.

Passing privileges to another subject:

- ACLs: usually not possible without explicit provisions for delegation.
- Capabilities: usually may be copied and passed to other entities at will.

Comparing ACLs and capabilities, cont.

Revocation of priviliges:

- ACLs: easy; just change the ACL list.
- Capabilities: impossible; once a capability is granted, it usually cannot be revoked. (Expiration dates help somewhat.)

It's clear that all of the interesting tradeoffs come in the *dynamic* behavior of the system.

ACLs and capabilities

A useful implementation technique:

> When an ACL is too slow, inflexible, or unwieldy, grant access to a capability by checking an ACL and then grant access to objects according to the capability.

This is known as a hybrid approach.

For example, in Unix, when you `open()` a file, it checks the file's ACL. The `open()` returns a file descriptor, which is just a capability to access the file; the file descriptor may be passed around at will.

Address spaces

For a nice example, let's look at how memory protection works in Unix.

Element of protection is a memory page.

Enforcement is provided by the combined effort of the OS kernel and the hardware memory management subsystem.

The main idea is to use virtual memory for protection: authorization is checked by OS when you load a page of memory into your virtual address space, and hardware ensures that memory accesses are restricted to valid pages.

You can think of the set of the virtual memory map loaded into the hardware as capabilities granting access to physical memory; the OS kernel enforces an ACL on physical pages of memory.

Dynamics

Enforcing access control on a static system with an unchanging access control policy is “easy.” The problem becomes much harder when you introduce *dynamics*, e.g. the ability to modify policy while the system is running.

A useful permission (like read, write, etc.) in this context is the permission to modify the ACL for the object in question.

But note how difficult it becomes to analyze the system. For instance, suppose we want to find whether there is any way which a subject S may affect an object O in any way. Then it is not enough to just check whether S has write permission on O; one must also check whether S has permission to modify the permissions for O, or whether S can modify anything which can modify O, etc.

Groups

Often it is useful to group a list of subjects together and specify a common set of permissions for the group. This is intended to simplify the specification (and management)

Examples: graduate-students, cs261-students, developers.

Then an ACL for this course's syllabus might look like

```
allow  read,write   Eric Brewer, Ian Goldberg, David Wagner
allow  read         cs261-students, graduate-students
```

Warning: be careful with dynamics of group membership!

Authentication

Authentication is about identifying the person who initiated a given request.

A few categories:

- Something you know: passwords, cryptographic keys
- Something you have: hardware tokens
- Something you are: biometrics
  - e.g. fingerprints, iris scans, hand-geometry, key-typing rates, etc.

Strongest approach is to use several of these in combination.

Password authentication

Many systems just require the user to enter a secret *password* known to the system.

Disadvantage: the sys-admin knows your password.
Fix: when the account is created, apply a non-invertible one-way hash function to the secret password and store the result. Then a login request may be validated by hashing the password provided and comparing it to the value stored earlier.

Passwords, cont.

Weaknesses:

- Guessability: most people pick poor passwords.
  - The Internet Worm (1988) included automatic password-guessing code.
- Eavesdropping: passwords are usually sent in the clear over the network, where they may be intercepted.
  - In 1995, CERT issued a security advisory noting that intruders had broken into many ISPs and backbone routers and intercepted tens of thousands of account passwords.

    CERT's advice? Every Internet user should change their password.

These weaknesses have been known for decades.

Secure authentication

For authentication across a network, passwords are obsolete technology.

The only known solution is cryptography; wait two weeks.

Enforcement

How is access control enforced?

- Hardware
- Operating system kernel
- Trusted subsystems

In practice, hardware provides very simple mechanisms, which are used to bootstrap an OS kernel. Then the security policy is refined by trusted programs run under the OS.

Hardware architectures

The concept of privileged execution states is critical.

- One simple concept: two modes of execution, *supervisor* mode (or *privileged* mode) and *unprivileged* mode (or *user* mode).
- May only access some features (memory mapping management, I/O, etc.) in supervisor mode.
- OS kernel runs in supervisor mode; normal programs run in unprivileged mode.
- May only switch to *privileged* mode by way of predefined gates (e.g.: a system call) which contain special checks and guards.

A refinement of this approach is *rings*, where there are multiple levels of privilege.

Upgrading privileges

How does a call to a privileged-mode process work?

- Trap to operating system
- Operating system checks arguments
  - Be careful with call-by-reference arguments and return addresses!
- OS executes privileged system call code
- Return to user-mode

Saltzer/Schroeder paper

General notes:

- In 1975, most systems didn't provide much security, or only recently contemplated the issues.
- Military a driving force behind much research on protection.
- Not much distinction made between hardware architectures and software operating systems.
- Lots of basic ideas known in 1975 keep getting independently "re-discovered."
- Some of it (e.g. details of Section II: descriptor-based protection systems) is basically outdated.

More on Wednesday. . .