

CSE331 Computer Security Fundamentals

9/7/2017 **Policy, Models, and Trust**

Michalis Polychronakis

Stony Brook University

Security Policy

A definition of what it means for a system to be secure

Comprises a set of well-defined rules involving:

Subjects: entities that interact with the system

Objects: any resource a security policy protects

Actions: anything subjects can (or cannot) do on objects

Permissions: allowed (or not) subject-action-object mappings

Protections: rules or mechanisms that aid in enforcing a policy

A security policy typically places constraints on what actions subjects can perform on objects to achieve specific security goals

Security Model

An scheme for specifying and enforcing security policies

Systems are large, complex, and dynamic → hard to manage
Abstract the system into a simpler *model*

Deal with the *information confinement* problem

Information should not flow into the wrong parties

Access control: who has access to what

Need ways to *represent access control rights*

ACLs, Capabilities, ...

Need ways to *grant specific rights*

DAC, MAC, RBAC, ...

Access Control Matrix

Abstract representation of a system's security state

Each cell defines the access rights for the given combination of subject and object

Empty cells mean that no access rights are granted

Objects: set of protected entities

Files, directories, devices, resources, ...

Subjects: set of active objects

Users, groups, processes, systems, ...

Rights: set of possible actions

Read, write, execute, own, append, ...

Meaning may vary depending on the object involved

Access Control Matrix

	/etc/passwd	/usr/bin/	/home/bob/	/admin/
root	read, write	read, write, exec	read, write, exec	read, write, exec
bob	read	read, exec	read, write, exec	-
backup	read	read, exec	read, exec	read, exec

Useful conceptual representation, but not practical

Size: number of subjects/objects will be too large

Efficiency: too many cells will be empty (no access)
or the same (default permissions)

Management: addition/removal of subjects/objects
requires non-trivial operations

Access Control Lists

List of permissions attached to an object

Object-centered approach: enumeration of all subjects and their access rights for the given object

No entry in the ACL → subject has no rights

In essence: the columns of the access control matrix

Facilitates the efficient implementation of default permissions

Wasteful to have separate entries for *many* different subjects that have the same rights over a given object

Better idea: define a “wildcard” to match any unnamed subject

Groups of subjects can also be represented in a similar way

Tied to the object

Can be stored along with the object in the form of metadata

Access Control Lists

	<code>/etc/passwd</code>	<code>/usr/bin/</code>	<code>/home/bob/</code>	<code>/admin/</code>
<code>root</code>	read, write	read, write, exec	read, write, exec	read, write, exec
<code>bob</code>	read	read, exec	read, write, exec	-
<code>backup</code>	read	read, exec	read, exec	read, exec

ACL

Advantages

Dramatically smaller size (empty cells are collapsed)

Disadvantages

No efficient way to enumerate all the rights of a given subject

Example: Unix File Permissions

A form of abbreviated ACL

A list of permissions attached to each file

Users are divided in three classes: *owner, group, all*

Separate read (r), write (w), and execute (x) permissions per class

One octal digit (3 bits) per class, with R=4, W=2, X=1

`rw-r-----` == 640 == Owner: RW, Group: R, All: none

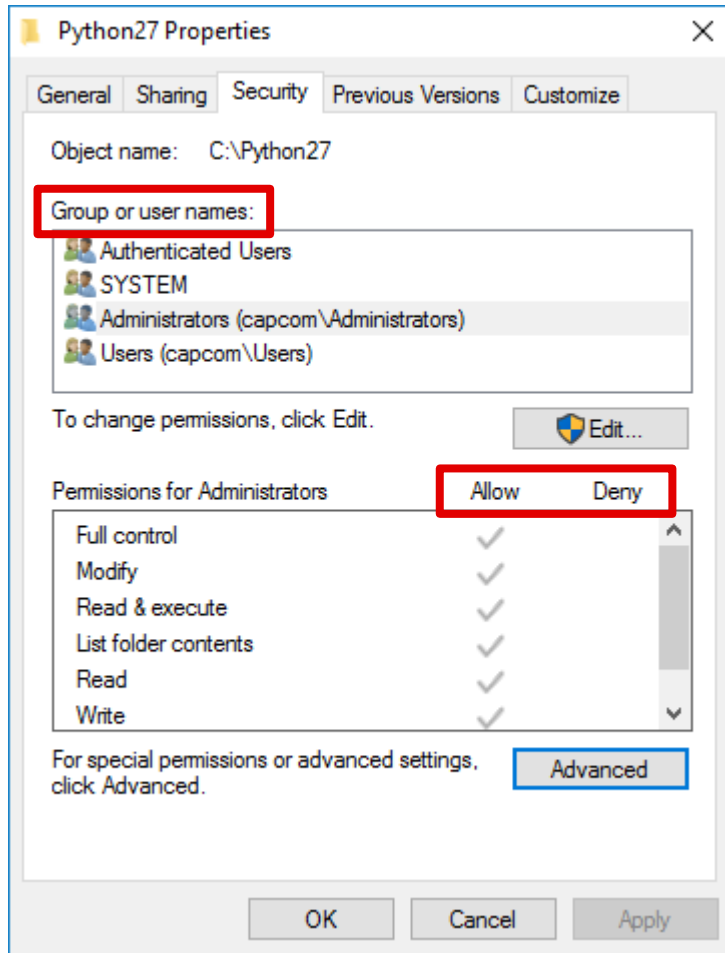
Coarse-grained control

How can we express “everyone except bob”?

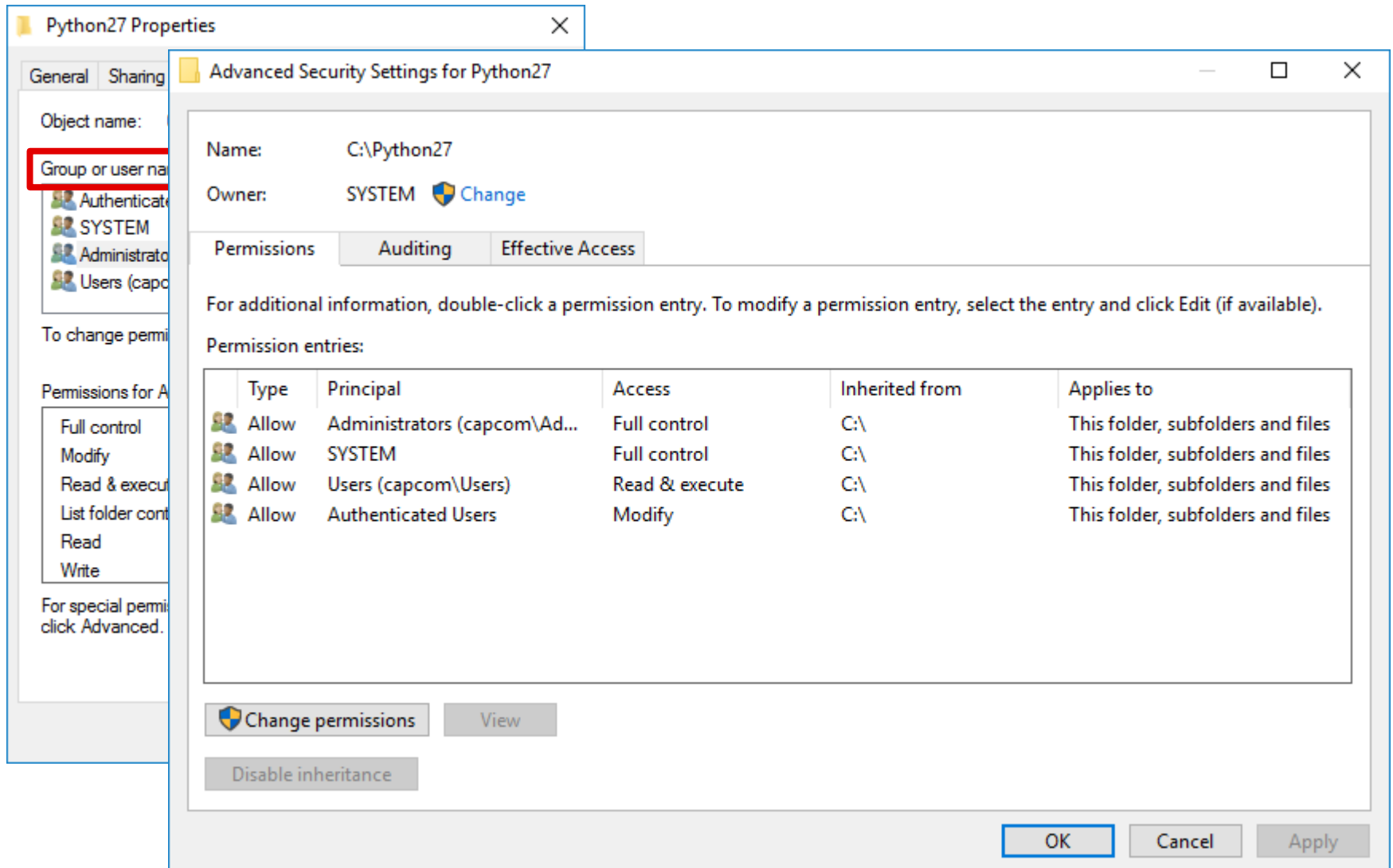
Non-standard arrangements require separate groups

Must create a group of all users except Bob (cumbersome: only root can create groups)

Example: Windows



Example: Windows File Permissions



The image shows a Windows File Permissions dialog box for the folder 'Python27'. The dialog is titled 'Advanced Security Settings for Python27'. It displays the following information:

- Name: C:\Python27
- Owner: SYSTEM [Change](#)

There are three tabs: 'Permissions', 'Auditing', and 'Effective Access'. The 'Permissions' tab is selected. Below the tabs, there is a text box with the following text:

For additional information, double-click a permission entry. To modify a permission entry, select the entry and click Edit (if available).

Below this text is a table of permission entries:

Type	Principal	Access	Inherited from	Applies to
Allow	Administrators (capcom\Ad...	Full control	C:\	This folder, subfolders and files
Allow	SYSTEM	Full control	C:\	This folder, subfolders and files
Allow	Users (capcom\Users)	Read & execute	C:\	This folder, subfolders and files
Allow	Authenticated Users	Modify	C:\	This folder, subfolders and files

At the bottom of the dialog, there are three buttons: 'Change permissions', 'View', and 'Disable inheritance'. The 'Change permissions' button is highlighted. At the very bottom of the dialog, there are three buttons: 'OK', 'Cancel', and 'Apply'.

NAME

acl - Access Control Lists

DESCRIPTION

This manual page describes POSIX Access Control Lists, which are used to define more fine-grained discretionary access rights for files and directories.

ACL TYPES

Every object can be thought of as having associated with it an ACL that governs the discretionary access to that object; this ACL is referred to as an access ACL. In addition, a directory may have an associated ACL that governs the initial access ACL for objects created within that directory; this ACL is referred to as a default ACL.

ACL ENTRIES

An ACL consists of a set of ACL entries. An ACL entry specifies the access permissions on the associated object for an individual user or a group of users as a combination of read, write and search/execute permissions.

An ACL entry contains an entry tag type, an optional entry tag qualifier, and a set of permissions. We use the term qualifier to denote the entry tag qualifier of an ACL entry.

The qualifier denotes the identifier of a user or a group, for entries with tag types of ACL_USER or ACL_GROUP, respectively. Entries with tag types other than ACL_USER or ACL_GROUP have no defined qualifiers.

[let's open the console]

Capabilities

List of access rights granted to a subject

Subject-centered approach: enumeration of all objects and the access rights a given subject has on them

In essence: the rows of the access control matrix

A capability can be a single token of authority

Unforgeable: must be validated by the OS

Communicable: can be passed on to other subjects or converted to less-privileged versions (in accordance to the enforced policy)

Tied to the subject

Typically stored in a privileged data structure

The subject should not be able to forge access rights or change the object a capability is related to

Capabilities

	/etc/passwd	/usr/bin/	/home/bob/	/admin/
root	read, write	read, write, exec	read, write, exec	read, write, exec
Capability bob	read	read, exec	read, write, exec	-
backup	read	read, exec	read, exec	read, exec

Advantages

Dramatically smaller size (empty cells are collapsed)

Disadvantages

No efficient way to enumerate all the rights on a given object

Example: Unix File Descriptors

To open a file, a process provides the file name and the desired access rights to the kernel

```
int fd = open("/etc/passwd", O_RDWR);
```

The kernel obtains the file's inode number by resolving the name through the file system hierarchy

The system then determines if the requested access should be granted using the access control permissions

If access is granted, the kernel returns a *file descriptor*

The variable `fd` in essence becomes a capability

The value of the file descriptor corresponds to an index in the process' file descriptor table

It can be *passed around* to other processes (e.g., as a result of `fork()` or by sending it through a socket)

NAME

capabilities - overview of Linux capabilities

DESCRIPTION

For the purpose of performing permission checks, traditional UNIX implementations distinguish two categories of processes: privileged processes (whose effective user ID is 0, referred to as superuser or root), and unprivileged processes (whose effective UID is nonzero). Privileged processes bypass all kernel permission checks, while unprivileged processes are subject to full permission checking based on the process's credentials (usually: effective UID, effective GID, and supplementary group list).

Starting with kernel 2.2, Linux divides the privileges traditionally associated with superuser into distinct units, known as capabilities, which can be independently enabled and disabled. Capabilities are a per-thread attribute.

Capabilities list

The following list shows the capabilities implemented on Linux, and the operations or behaviors that each capability permits:

CAP_AUDIT_CONTROL (since Linux 2.6.11)

Enable and disable kernel auditing; change auditing filter rules; retrieve auditing status and filtering rules.

CAP_AUDIT_READ (since Linux 3.16)

Allow reading the audit log via a multicast netlink socket.

[let's open the console]

Example: Linux Capabilities

Is this really necessary? What if ping is vulnerable?

```
$ ls -l /bin/ping
-rwSr-xr-x 1 root root 44K May  7  2014 /bin/ping*
```

Remember the principle of least privilege?

```
$ ls -l /bin/ping
-rwxr-xr-x 1 root root 55K Mar  1  2016 /bin/ping*
$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

Access Control Policies

Different approaches to granting access rights

Who is responsible? admin, user, owner, ...

Based on what? identity, role, group, rule, ...

Main Types of Access Control

Discretionary

Mandatory

Role-based

Rule-based

...

Discretionary Access Control

Subjects determine who has access to their objects

Key concept: *the owner*

Determines the access rights of other subjects on that object

A subject with a certain access permission can pass it on to any other subject

Commonly used in most operating systems

E.g., Linux and Windows allow non-root users to specify file and folder permissions based on ACLs

On existing files they can already access, or any new files they will create



NEW

Starred



Sharing settings

Link to share (only accessible by collaborators)

Who has access

	Private - Only you can access	Change...
	Michalis Polychronakis (you) mikepo@cs.stonybrook.edu	Is owner

Invite people:

Owner settings [Learn more](#)

- Prevent editors from changing access and adding new people
- Disable options to download, print, and copy for commenters and viewers

[Done](#)

foo



DETAILS

ACTIVITY

Today

12:30 PM



You renamed an item



foo

Untitled document

No recorded activity before September 7, 2017

Mandatory Access Control

An administrator grants all access rights

Cannot be altered by subjects → users cannot override decisions either accidentally or intentionally

MAC-enabled systems allow policy administrators to implement strict organization-wide security policies

Multilevel security (MLS) and specialized military systems

Getting traction in mainstream OSes as a means of minimizing abuse and preventing misconfigurations

Linux: SELinux, AppArmor

Windows: Mandatory Integrity Control

Example: The Bell-LaPadula Model

Inspired by the military multilevel security paradigm

Used for document classification and personnel clearance

Security levels

Each object is classified at one of the security levels

Each user obtains clearance at one of the security levels

Example: MLS classification

Unclassified → Confidential → Secret → Top Secret

Goal: protect the confidentiality of information

Prevent read access to objects at a security classification higher than the subject's clearance

Example: The Bell-LaPadula Model

Access to objects is controlled by two rules

Simple security property

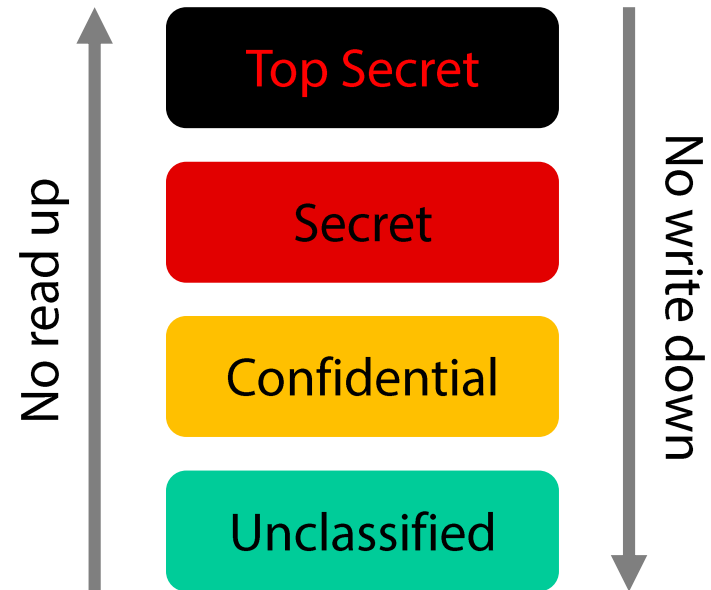
A subject cannot read an object of higher sensitivity

“no read up”

Star () property*

A subject cannot write to an object of lower sensitivity

“no write down”



Role-based Access Control

Access to an object is governed by the *role* of the subject within an organization

Administrators define roles, specify access rights for each role, and assign subjects to roles

Much more convenient than managing subjects individually

Example: roles for a computer science department

Student, faculty, administrative personnel, sysadmin, ...

Role hierarchies

More privileged roles may inherit the access rights of less privileged roles

Key difference from user groups

Rule-based Access Control

Allow or deny access to resources based on conditions other than the subject's identity

Time of day, location, type of device, ...

Attribute-based Access Control

Policies based on user, resource, environmental, or other attributes

Can be expressed in the form of complex Boolean logic rules

Context-based Access Control

Take into account relevant state information

E.g., block externally-initiated and allow internally-initiated TCP connections

...

Trust

A security policy is in essence a set of axioms that the policy makers *believe* can be enforced

Relies on several assumptions

- The policy correctly captures all possible secure and insecure states of the system

- The policy enforcement mechanisms prevent the system from entering an insecure state

- The associated risks have been adequately assessed

How can we trust the policy?

- Is it correct? Complete? Unambiguous?

How can we trust the mechanisms?

- Do they contain flaws? Are they configured correctly? Do they enforce all aspects of the policy? Is there a backdoor?