Last time

Processes and Users

DAC

Today

MAC

DAC

Discretionary access control

Organizing principle:

Files and directories have **owners** who have the **discretion** to say who gets to access them.

Major implementations:

Unix permissions
Access control lists (ACLs)

This image shows a Windows dialog for editing an access control list (ACL). In this scheme, an owner can write more complex security policies than the cross product of (owner, group, others) × (read, write, execute). Instead, the owner can specify a list of arbitrary size, granting or denying permission for any user, group or combination thereof to perform operations on an OS object like a file. ACLs are a very flexible mechanism for representing permissions, but the downside of giving people a lot of flexibility is that they might use it! It's easy to create an ACL so complex that you don't understand it. ACLs are necessary in some circumstances — e.g., centralized file servers that are accessed by thousands of employees from different departments. For a lot of circumstances, however, simple is better.

Recall: Superuser

a.k.a., `root` user

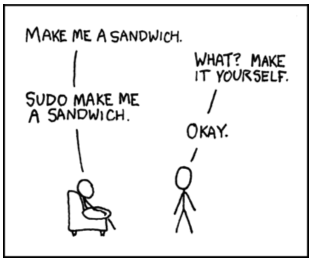
- UID 0
- can change file owner, `chmod` other users' files
- second-level objective for many attacks

The `root` user is allowed to violate the DAC policy, overriding the access control decisions made by a file's owner (and even ______________________!). To "get root" is to gain administrative control over a computer, whether legitimately becoming a system administrator ("yeah, I've got root on that box") or otherwise.

Many, many attacks against systems start by gaining ________________________ (running whatever the attacker wants within a process, with that process' credentials) and then a ______________________ attack against a service that allows the attacker to _____________ to administrative access.

Root-only programs

- lots of tools require *root privilege*:
  - filesystem management
  - package managers
  - service management
  - often via `sudo(8)`



Exercise: Consider how a user who can control all software installation on a computer could violate another user's security policy

We don't want just any user being able to, e.g., control a mounted filesystem or install a package. Why not?

For all of these examples, being in control of such a subsystem would allow a user to be able to ______________________________.

Root-only programs

- some programs require `root` privilege
- some programs must be runnable by anyone
- some are both!
- e.g., `ping(8)`, even `intel_backlight(1)`!

```
$ ls -l `which intel_backlight`
-r-sr-xr-x  1 root  wheel    16K Feb 26 17:03 /usr/local/bin/intel_bac
```

Since we don't want just anybody controlling critical subsystems, some programs require `root` privilege in order to do their work. For example, I can ______________________ on my machine from an ordinary user account, but I can only ____________________ to system locations (e.g., `/usr/local/bin`) as `root`.

Some programs, however, require privilege to do their job and ***also*** need to be run by ordinary users! We can implement such functionality, overriding the normal DAC policy, using `setuid` and `setgid` software.

`setuid/setgid programs`

`setuid`: set *effective UID* to file owner's UID on run
`setgid`: set *effective GID* to file group's GID on run

Can query *real* or *effective* UID/GID:

```c
#include <unistd.h>

uid_t   getuid(void);
uid_t   geteuid(void);
gid_t   getgid(void);
gid_t   getegid(void);
```

Example: getuid.c

DAC

Organizing principle:

Files and directories have **owners** who have the **discretion** to say who gets to access them.

Major implementations:

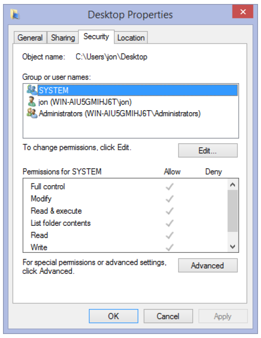
Unix permissions
Access control lists (ACLs)

So… how about that other implementation, ACLs?

ACLs: access control lists

(the other way of doing discretionary access control)

- explicit list of users, groups
- independent permissions for each
- useful for complex authorization on multi-user shared systems* implemented in NFSv4, POSIX 1e, Windows/SMB...



Object name: C:\Users\jon\Desktop

Group or user names:
SYSTEM
jon (WIN-AIU5GMIHJ6T\jon)
Administrators (WIN-AIU5GMIHJ6T\Administrators)

To change permissions, click Edit.

Permissions for SYSTEM	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
List folder contents	✓	
Read	✓	
Write	✓	

For special permissions or advanced settings, click Advanced.

ACLs are useful, but it's also very easy to write an ACL that you yourself don't understand! Just look at some of the literature around trying to make ACLs understandable by users:

Intentional access management: making access control usable for end-users Cao and Iverson, ***SOUPS '06: Proceedings of the second symposium on Usable privacy and security, July 2006***. DOI: https://doi.org/10.1145/1143120.1143124

Relating declarative semantics and usability in access control Krishnan, Tripunitara, Chik and Bergstrom, ***SOUPS '12: Proceedings of the Eighth Symposium on Usable Privacy and Security***, July 2012. DOI: https://doi.org/10.1145/2335356.2335375

The poor usability of OpenLDAP Access Control Lists, Chen, Punchhi and Tripunitara, ***IET Information Security*** 17(1), January 2023. DOI: https://doi.org/10.1049/ise2.12079

If people can publish "how to make ACLs usable" over three decades... maybe there's a problem with ACLs.

MAC: mandatory access control

Organizing principle:

System administrators can impose access control policies that file owners **cannot control or circumvent**.

If users can't be trusted... which happens a lot!

History

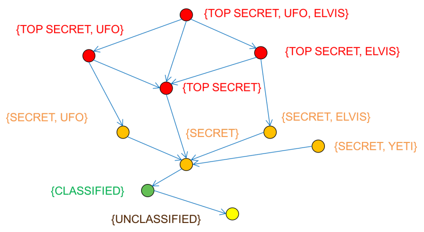
Confidentiality:

"Very hush-hush"

Need-to-know

Formal classification levels*

Lattices



Source: Paul Krzyzanowsi

* See, e.g., the Government of Canada's Levels of security
† See, e.g., the declassified NSA document Examples of Lattices

As soon as you start to deal in any quantity of confidential information, it becomes important to describe just ________________________ some information is relative to other information. The phrases "loose lips sink ships" and "it's all very hush-hush" both imply that some information ought to be confidential, but one implies a general disposition towards confidentiality, whereas the other implies that something specific and special requires particular care.

Loose definitions of confidentiality can be described with informal but still well-understood terminology such as "need to know". This approach to maintaining confidentiality is useful, but the human judgement involved is still insufficient for making ***automated*** decisions about who should be able to access information.

Large, complex organizations that need to specify clear rules for accessing information — traditionally, the military and intelligence services — have created formal classification levels that allow rules to be applied very clearly and definitively, with little judgment required: "is this person cleared to view information with this classification marking?"

Lattices make things even more complex, as they add an ___________________ system of code words.

The Anderson Report*†

Trusted computing base

Reference monitor

Policy vs mechanism

So what policies should be enforced?

---

* No relation!
† Anderson, "Computer Security Technology Planning Study", Tech. Rep. ESD-TR-73-51, Vol. II, US Air Force, 1972.

The Anderson report introduced several key terms and concepts that we rely on today. We've already talked about TCBs, but Anderson also introduced the concept of a ***reference monitor***: a system that can ____________________ to information and ________________ about them. This allows __________ to be encoded separately from ______________: a system provides a "how": how ___________________________, and system administrators can supply the "what": ___________________________.

Multi-level security (MLS)

One computer

Many labels

Who can do what to what? *It depends!*

Who can be trusted to specify access control policy?

MAC answer: **System administrators** can impose access control policies that file owners cannot control or circumvent.

Bell-LaPadula

No read up (confidentiality)

No write down (the **-property*)

Administrative burden and high-water marks

Reference: Bell and LaPadula, "Secure Computer Systems: Mathematical Foundations", *The Mitre Corporation, AD-770 768*, 1973.

It's a lot of work to label every object in a system. One way to cope with this tsunami of labeling is to allow objects to "float" to the highest label that has written data into them (the "high-water mark"). If a Secret process writes into a Confidential file, instead of disallowing the write, the file can be relabeled as Secret. Thus, any Confidential processes will lose access to the file.

Biba

Confidentiality not our only goal!

Reads and writes

LOMAC

Windows

Reference: Biba, "Integrity Considerations for Secure Computer Systems", *The Mitre Corporation, MTR-3153*, 1975.

Security isn't just about confidentiality. In some cases, ________________ of data is more important than its confidentiality. In almost all cases __________________________ is a necessary prerequisite to providing ***any*** security properties!

We see this used extensively in contemporary operating systems: a process can read from a higher-integrity object (e.g., a file), but not write to them.

LOMAC refers to ______________________ MAC. This is the logical dual of the ______________ _________ of confidentiality. Mandatory Access Control code that was originally developed for organizations that care about confidentiality can now be used to label objects as "downloaded via a browser", and thus lower-integrity than other files.

Modern version of Windows have four integrity levels: low, medium, high and system. Even if a program is running on behalf of the Administrator, it can't overwrite critical OS files that are labeled with System integrity unless it is itself a System-integrity process (e.g., Windows Update).

[Domain and] Type Enforcement

Categories for subjects and objects

DTE and DTEL

FLASK

SELinux and AppArmor

Badger, Sterne, Sherman, Walker and Haghighat, "A domain and type enforcement UNIX prototype", *USENIX Comput. Syst., vol. 9, no. 1*, pp. 47–83, 1996.

Badger et al.'s ***Domain and Type Enforcement*** allowed a conventional UNIX machine to be partitioned into various ***domains***, and to have MAC policies enforced to constrain the flow of information between them. This included a language for expressing DTE policy (DTEL), and it led to further work on enhancing the security of UNIX and UNIX-like operating systems: TrustedBSD, FLASK, SELinux and AppArmor.

Linux Security Modules

Patches and problems

"Can you make it a module?"

Comprehensive *hooks* that call arbitrary modules

Separation of *mechanism* from *policy*

Wright et al., "Linux Security Modules: General Security Support for the Linux Kernel", in *Proceedings of the 11th USENIX Security Symposium*, 2002.

This separation of mechanism from policy allows lots of different policies to be enforced, from traditional MAC policies to access control schemes such as Role-Based Access Control and beyond.

FreeBSD MAC Framework

Hooks:

```
#ifdef MAC
    error = mac_vnode_check_chdir(td->td_ucred, vp);
    if (error != 0)
        return (error);
#endif
```

Phones

Another example of MAC hooks scattered through an operating system is the FreeBSD MAC Framework, which came out of the TrustedBSD project. Hooks exist to allow a reference monitor to make an access control decision based on a _______________ (who wants to make the access), an ______________ (what's being accessed, in this case a file's ***vnode*** — more about that in ECE 8400 / ENGI 9875) and a currently-installed ______________ (which may actually be a composition of ______________________).

The FreeBSD MAC Framework is most famously used, not for FreeBSD itself, but to provide a foundation for application sandboxing on iOS and macOS!

MAC summary

History

MLS

MAC in practice

- Linux Security Modules
- FreeBSD MAC Framework