What is access control?

- In its broadest sense: “The prevention of unauthorized use of a resource, including the prevention of use of a resource in an unauthorized manner.” (ITU-T Recommendation X.800)

- Often used to mean this same issue restricted to one computer: Which principals are allowed to perform which operations on which objects
Relation to other security pieces

• Authentication comes first before access control is an issue (i.e., logon).

• Authorization is the meta-level thing of deciding what your access rights are.

• Audit is independent review of everything else.
History

• Old subject as part of single-computer Operating System (OS); 1970s or earlier

• Same issues for centuries in physical world; also same issues in databases

• Historic heart of computer security; e.g., distinguishes field from cryptography
Operating Systems

- The special computer program that is always running on any computer, and that is responding to keystrokes, mouse clicks, etc.
  - I.e., provides user (programs) with controlled access to hardware components.

- Today, overwhelmingly, general computers (laptops, desktops, servers) running OS from one of two families: Windows, and Unix

- N.B.: An oligopoly (but not quite a duopoly)
File System

• Broadly, any method of storing computer files and the data they contain to make it easy to find and access them.

• For this discussion, the collection of programs in an OS that manages and stores the files and data in secondary storage, and the data itself.

  • Secondary storage: “hard drive,” capacity of 100’s of GB; retains data when computer off.

• Where access control does its thing today
All OS contain bugs

- OS is very large; there will be a certain bug rate

- Issue: in last 10–15 years, the kernel, the part that really controls access to hardware, has grown hugely. Once upon a time, at least the kernel might have had very few bugs that were security vulnerabilities.
Unix

• Operating System developed in the 1970s at AT&T Bell labs, also slightly later U. C. Berkeley.

• Today, two main strains:
  1. Linux
  2. Mac OS X

• Historically considered more secure, better access control model than Windows (e.g., vs. Win 98)
File system basics

• Tree-like structure, with “folders” and files grouped inside of other files hierarchically.

• Top level corresponds to physical device (e.g., hard drive) or logical partition thereof.

  • Called “drives” and get letter in Windows, called mount points in Unix-based systems.

• File system security controls access to files and folders, and is generally controlled by user and group permissions.
Access control types

- Traditionally distinguish **Mandatory Access Control (MAC)** from **Discretionary Access Control (DAC)**

- MAC: Also known as Multi-Level Security; concerned especially with classified information (Mil Sec) at different levels on one computer.
  - Information flow is key.
  - Stimulated much research.
- DAC: Information owner determines access
DAC: Access (Control) Matrix

- (Lampson ‘71)

- **Principals** (processes? users?) perform operations on **objects** (files? files and other resources like comm ports?)

- Intuitively, giant matrix of Principals vs. objects with entries showing which operations will be allowed.

- *Reference monitor* built into OS allows operation only if permission is present.
Access matrix (cont)

- I.e., access matrix $a$ contains in entry $a[i,j]$ the set of permissions that Principal $i$ has for operating on object $j$.

- *Permission* is right to perform an operation, typically read, write, execute, append.

- Access matrix sparse and/or uniform. I.e., either most entries blank, or tons of entries all just say “read”.

- Also need to record who is owner of each object.
## Example

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
<th>Device</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>read, write, execute, own</td>
<td>read</td>
<td>read</td>
<td>control</td>
</tr>
<tr>
<td>P2</td>
<td>read</td>
<td>write</td>
<td>write</td>
<td>wakeup</td>
</tr>
</tbody>
</table>
To complete a system

• What rules decide how the access matrix evolves over time?

• E.g., each object is born with exactly one owner, and owner can fill in its column

• Various rules have been studied; can get quite complex. (At surprisingly low level complexity of rules, the analysis of what might happen gets very complicated.)
Access matrix issues

• Might want a third dimension of “With what program?” I.e., I can write the Research Funds database with the accounting program, but not a text editor

• Even in 2 D, will not scale well to 10,000 users and 1 million objects

• In practice, keep rows (“capabilities”) or columns (“access control list—ACL”) and severely limit different options.
• Permissions kept per object; effectively ACL.

• 3 kinds of permission: read, write, execute (also an owner)

• Only 3 different classes of users: Owner, group, other (all other users on this computer)

• System has one superuser/root
SUID Vulnerability

- Notice access control mentions only users and objects, no mention of which program the user runs (3rd dimension).

- Problem: I want to change my password. I should not be allowed to alter the password file, but I should be able to run the password-changing program and it should be able to change the file.

- Solution: Certain programs run with privileges of program owner, typically root, not privileges of user running the program. Mechanism is called “set user ID (SUID)”.

Windows File System Security

- NTFS: NT File System, inspired by Unix.
- Much like Unix, but more total distinct permissions.
- Many additional issues and features with Windows Server; will not cover.
- Unix and Windows implement users and groups differently, with different scopes.
Intermezzo: MAC

• Variously known (and with subtle differences) as “multi-level security,” mandatory access control, lattice-based access control.

• Classic example is Mil Sec: unclassified, confidential, secret, top secret, with extreme concern re information flow.

• Tolerably well understood; not the center of most of today’s computer security issues.

• Most famous early model: Bell-LaPadula
OS: What goes wrong

• Typically Black Hats consider it easy to gain super user access to a machine once they have gotten ordinary user access

  • Passwords are the (weak!) keys to the kingdom

• One (among several) excellent ways to do so is the infamous *buffer overflow*, aka *buffer overrun* aka *smashing the stack*.

  • Recent years: smashing the stack attacks have dropped and are now only about half of all technical attacks on OSes.
Smashing the stack

• Can view the computer’s memory (say RAM) as a very big collection of slots each of which holds one word (2, 4, or 8 bytes).

• Both data storage and the actual program instructions all live somewhere in there.

• One section is set aside for managing the running of currently executing program, rather than generic storage. This is the (run-time) stack.
Stack smashing (cont.)

- One thing that definitely goes in run-time stack are arguments to programs.

- E.g., the “Ichabod” part of lookup(“Ichabod”)

- Programmer has to say how much space to leave for this argument. For an arbitrary string that is supposed to be a person’s name, might allocate 32 characters, or 256 characters to be really safe.

- To attack, give e.g., 100,000 characters!
Buffer overflow (cont.)

• The 100,000 characters bleed over much of the memory, and the end of them wind up in the section of the memory where program instructions live.

• E.g., write 30,000 “No op” lines (to allow for slop in lining things up) followed by short program to (1) create new superuser account with empty password, and (2) erase the audit log files.
What's C got to do with it?

- Many OS are written in C, the great new language of the late 1970s.

- By default, C does not check length of inputs to character arrays in, e.g., strcpy.

- Can check this using C; it's just not automatic. (strncpy with length, not strcpy).

- Is automatically checked in newer languages, e.g., a standard library String of C++.
History of buffer overflow

• According to Anderson, well known weakness in 1960s.

• Vulnerability exploited by Robert Morris Internet Worm of 1988. (Day without Internet.)

• >1/2 of early 2000s course MCS 494 on Unix security holes of DJB.

  • (Question: Why did DJB say he was teaching on Unix rather than Windows security holes?)
Environmental creep again

- Unix was created for environment with trusted competent users who occasionally messed up at Bell Labs in the 1970s.

- This still mostly described users and the young Internet of the mid 1980s.

- Today, mostly technically incompetent users (drivers not automotive engineers) and some malicious.
Contrarian view

- We are moving back to one computer per person. Access control not necessarily important.

- Access control is to separate different users from one another on one machine.

- And machines that are not one user are instead one purpose (e.g., web server).