Last time

• Controls against security flaws in programs

• Various controls applicable to each of the stages in the software development lifecycle

• To get the best chance of controlling all of the flaws:
  • Standards describing the controls to be used
  • Processes implementing the standards
  • Audits ensuring adherence to the processes
Security controls—Documentation

• How can we control security vulnerabilities through the use of documentation?

• Write down the choices you made
  - And why you made them

• Just as importantly, write down things you tried that didn't work!
  - Let future developers learn from your mistakes

• Make checklists of things to be careful of
  - Especially subtle and non-obvious security-related interactions of different components
Security controls—Maintainance

• By the time the program is out in the field, one hopes that there are no more security flaws
  – But there probably are

• We've talked about ways to control flaws when modifying programs
  – Change management, code review, testing, documentation

• Is there something we can use to try to limit the number of flaws that make it out to the shipped product in the first place?
Standards, process, and audit

- Within an organization, have rules about how things are done at each stage of the software lifecycle
- These rules should incorporate the controls we've talked about earlier
- These are the organization's standards
- For example:
  - What design methodologies will you use?
  - What kind of implementation diversity?
  - Which change management system?
  - What kind of code review?
  - What kind of testing?
Standards, process, and audit

• Make formal *processes* specifying how each of these standards should be implemented
  – For example, if you want to do a guided code review, who explains the code to whom? In what kind of forum? How much detail?

• Have *audits*, where somebody (usually external to the organization) comes in and verifies that you're following your processes properly

• This doesn't guarantee flaw-free code, of course!
This time

- Protection in General-Purpose Operating Systems
  - History
  - Separation vs. Sharing
  - Segmentation and Paging
  - Access Control Matrix
  - Access Control Lists vs. Capabilities
Operating System

- An operating system allows different users to access different resources in a **shared way**

- The operating system needs to control this sharing and provide an interface to allow this access

- **Identification** and **authentication** are required for this access control

- We will start with memory protection techniques and then look at access control in more general
History

- Operating systems evolved as a way to allow multiple users use the same hardware
  - Sequentially (based on executives)
  - Interleaving (based on monitors)
- OS makes resources available to users if required by them and permitted by some policy
- OS also protects users from each other
  - Attacks, mistakes, resource overconsumption
- Even for a single-user OS, protecting a user from him/herself is a good thing
  - Mistakes, malware
Protected Objects

- CPU
- Memory
- I/O devices (disks, printers, keyboards,...)
- Programs
- Data
- Networks
Separation

- Keep one user's objects separate from other users
- **Physical** separation
  - Use different physical resources for different users
  - Easy to implement, but expensive and inefficient
- **Temporal** separation
  - Execute different users' programs at different times
- **Logical** separation
  - User is given the impression that no other users exist
  - As done by an operating system
- **Cryptographic** separation
  - Encrypt data and make it unintelligible to outsiders
  - Complex
Sometimes, users do want to share resources
  - Library routines (e.g., libc)
  - Files or database records

OS should allow flexible sharing, not “all or nothing”
  - Which files or records? Which part of a file/record?
  - Which other users?
  - Can other users share objects further?

What uses are permitted?
  - Read but not write, view but not print (Feasibility?)
  - Aggregate information only
  - For how long?
Memory and Address Protection

- Prevent program from corrupting other programs or data, operating system and maybe itself
- Often, the OS can exploit hardware support for this protection, so it’s cheap
- (See CS 350 memory management slides)
- Memory protection is part of translation from virtual to physical addresses
  - Memory management unit (MMU) generates exception if something is wrong with virtual address or associated request
  - OS maintains mapping tables used by MMU and deals with raised exceptions
Protection Techniques

- **Fence register**
  - Exception if memory access below address in fence register
  - Protects operating system from user programs
  - Single user only

- **Base/bounds register pair**
  - Exception if memory access below/above address in base/bounds register
  - Different values for each user program
  - Maintained by operating system during context switch
  - Limited flexibility
Protection Techniques

• Tagged architecture
  • Each memory word has one or more extra bits that identify access rights to word
  • Very flexible
  • Large overhead
  • Difficult to port OS from/to other hardware architectures

• Segmentation

• Paging
Segmentation

- Each program has multiple address spaces (segments)
- Could use different segments for code, data, and stack
  - Or maybe even more fine-grained, e.g., different segments for data with different access restrictions
- Virtual addresses consist of two parts:
  \(<\text{segment name, offset within segment}>\)
- OS keeps mapping from segment name to its base physical address in Segment Table
- OS can (transparently) relocate or resize segments and share them between processes
- Each segment has its own memory protection attributes
Segment Table

<table>
<thead>
<tr>
<th>Logical Program</th>
<th>Segment Translation Table</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>MAIN c</td>
<td>0</td>
</tr>
<tr>
<td>SEG_A</td>
<td>SEG_A g</td>
<td>a</td>
</tr>
<tr>
<td>SUB</td>
<td>SUB a</td>
<td>b</td>
</tr>
<tr>
<td>DATASEG</td>
<td>DATA_SEG h</td>
<td>c</td>
</tr>
</tbody>
</table>

Location 20 Within Segment DATA_SEG

FETCH<DATA_SEG,20>
Review of Segmentation

● Advantages:
  • Each address reference is checked for protection by hardware
  • Many different classes of data items can be assigned different levels of protection
  • Users can share access to a segment, with potentially different access rights
  • Users cannot access an unpermitted segment

● Disadvantages:
  • External fragmentation
  • Dynamic length of segments requires costly out-of-bounds check for generated physical addresses
  • Segment names are difficult to implement efficiently
Paging

- Program (i.e., virtual address space) is divided into equal-sized chunks (pages)
- Physical memory is divided into equal-sized chunks (frames)
- Frame size equals page size
- Virtual addresses consist of two parts:
  - `<page #, offset within page>`
    - # bits for offset = \(\log_2(\text{page size})\), no out-of-bounds possible for offset
- OS keeps mapping from page # to its base physical address in Page Table
- Each page has its own memory protection attributes
Review of Paging

• Advantages:
  • Each address reference is checked for protection by hardware
  • Users can share access to a page, with potentially different access rights
  • Users cannot access an unpermitted page

• Disadvantages:
  • Internal fragmentation
  • Assigning different levels of protection to different classes of data items not feasible
x86 Architecture

- x86 architecture provides both segmentation and paging
  - Linux uses a combination of segmentation and paging
    - Only simple form of segmentation to avoid portability issues
    - Segmentation cannot be turned off on x86
  - Same for Windows
- Memory protection bits indicate no access, read/write access or read-only access
- Recent x86 processors also include NX (No eXecute) bit, forbidding execution of instructions stored in page
  - Enabled in Windows XP SP 2 and some Linux distros
  - Helps against some buffer overflows
Access Control

- Memory is only one of many objects for which OS has to run access control.

- In general, access control has three goals:
  - **Check every access**: Else OS might fail to notice that access has been revoked.
  - **Enforce least privilege**: Grant program access only to the smallest number of objects required to perform a task.
    - Access to additional objects might be harmless under normal circumstances, but disastrous in special cases.
    - Examples?
  - **Verify acceptable use**: Limit types of activity that can be performed on an object.
    - E.g., for integrity reasons (ADTs).
Access Control Matrix

- Set of protected objects: \( O \)
  - E.g., files or database records
- Set of subjects: \( S \)
  - E.g., humans, processes acting on behalf of humans or group of humans/processes
- Set of rights: \( R \)
  - E.g., \{read, write, execute, own\}
- Access control matrix consists of entries \( a[s,o] \), where \( s \in S \), \( o \in O \) and \( a[s,o] \subseteq R \)
Example Access Control Matrix

<table>
<thead>
<tr>
<th></th>
<th>File 1</th>
<th>File 2</th>
<th>File 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>orw</td>
<td>rx</td>
<td>o</td>
</tr>
<tr>
<td>Bob</td>
<td>r</td>
<td>orx</td>
<td></td>
</tr>
<tr>
<td>Carol</td>
<td></td>
<td>rx</td>
<td></td>
</tr>
</tbody>
</table>
Implementing Access Control Matrix

- Access control matrix is hardly ever implemented as a matrix
  - Matrix would likely be sparse
  - Updates would likely be tedious
- Instead, an access control matrix is typically implemented as
  - a set of access control lists
    - column-wise representation
  - a set of capabilities
    - row-wise representation
  - or a combination
Access Control Lists (ACLs)

- Each object has a list of subjects and their access rights
  - E.g., File 1: \{Alice:orw, Bob:r\}, File 2: \{Alice:rx, Bob:orx, Carol:rx\}
  - ACLs are implemented in Windows file system (NTFS), user entry can denote entire user group (e.g., “Students”)
  - Classic UNIX file system has simple ACLs. Each file lists its owner, a group and a third entry representing all other users. For each class, there is a separate set of rights. Groups are system-wide defined in /etc/group, use chmod/chown/chgrp for setting access rights to your files

- Which of the following can we do quickly for ACLs?
  - Determine set of allowed users per object
  - Determine set of objects that a user can access
  - Revoke a user’s access right to an object or all objects
Capabilities

- A capability is an **unforgeable token** that gives its owner some access rights to an object
  - E.g., Alice: {File 1:orw}, {File 2:rx}, {File 3:o}
- Unforgeability enforced by having OS store and maintain tokens or by cryptographic mechanisms
  - One such mechanism, digital signatures (see later), allows tokens to be handed out to processes/users. OS will detect tampering when process/user tries to get access with modified token.
- Owner of token might be allowed to transfer token to others
- Some research OSs (e.g., Hydra) have fine-grained support for tokens
  - Caller gives callee procedure only minimal set of tokens required
- Answer questions from previous slide for capabilities
Combined Usage of ACLs and Cap.

- In some scenarios, it makes sense to use both ACLs and capabilities
  - E.g., for efficiency reasons
- In a UNIX file system, each file has an ACL, which is consulted when executing an open() call
- If approved, caller is given a capability listing type of access allowed in ACL (read or write)
  - Capability is stored in memory space of OS
- Upon read() / write() call, OS looks at capability to determine whether type of access is allowed
- We cannot withdraw access from a user if user has already opened file
Recap

• Protection in General-Purpose Operating Systems
  • History
  • Separation vs. Sharing
  • Segmentation and Paging
  • Access Control Matrix
  • Access Control Lists vs. Capabilities
Next time

• Role-Based Access Control

• User Authentication
  • Authentication Factors
  • Passwords
  • Attacks on Passwords
  • Biometrics