Operating System Security, Continued
CS 239
Computer Security
February 25, 2004

Outline

- Buffer overflows
- Designing secure operating systems
- Assuring OS security
- Logging and auditing

Buffer Overflows

- One of the most common causes for compromises of operating systems
- Due to a flaw in how operating systems handle process inputs
  - Or a flaw in programming languages
  - Or a flaw in programmer training
  - Depending on how you look at it

What Is a Buffer Overflow?

- A program requests input from a user
- It allocates a temporary buffer to hold the input data
- It then reads all the data the user provides into the buffer, but . . .
- It doesn't check how much was provided

For Example,

```
int main(){
  char name[31];
  printf(“Please type your name:  “);
  gets(name);
  printf(“Hello, %s”, name);
  return (0);
}
```

- What if the user enters more than 32 characters?

Well, What If the User Does?

- The code continues reading data into memory
  - That's how `gets()` works
- The first 32 bytes go into `name`
- Where do the remaining bytes go?
- Onto the stack

Munging the Stack

- The temporary variable `name` is allocated on the stack
  - Close to the record of the function currently being run
- The overflow will spill into whatever's next on the stack
- Commonly, that's effectively going to overwrite the instruction pointer

Using Buffer Overflows to Compromise Security

- Carefully choose what gets written into the instruction pointer
- So that the program jumps to something you want to do
  - Under the identity of the program that's running
- Such as, execute a command shell

Effects of Buffer Overflows

- Remote or unprivileged local user gets to run a program with greater privileges
- If buffer overflow is in a root program, gets all privileges, essentially
- Common mechanism to allow attackers to break into machines

Are Buffer Overflows Common?

- You bet!
- Weekly occurrences in major systems/applications
- Probably one of the most common security bugs

Fixing Buffer Overflows

- Check the length of the input
- Use programming languages that prevent them
- Put in OS controls that prevent overwriting the stack
- Why aren't these things commonly done?
- Presumably because programmers and designers neither know nor care about security

Desired Security Features of a Normal OS

- Authentication of users
- Memory protection
- File and I/O access control
- General object access control
- Enforcement of sharing
- Fairness guarantees
- Secure IPC and synchronization
- Security of OS protection mechanisms

Extra Features for a Trusted OS

- Mandatory and discretionary access control
- Object reuse protection
- Complete mediation
- Audit capabilities
- Intruder detection capabilities

How To Achieve OS Security

- Kernelized design
- Separation and isolation mechanisms
- Virtualization
- Layered design

Advantages of Kernelization

- Smaller amount of trusted code
- Easier to check every access
- Separation from other complex pieces of the system
- Easier to maintain and modify security features

Reference Monitors

- An important security concept for OS design
- A *reference monitor* is a subsystem that controls access to objects
  – It provides (potentially) complete mediation
- Very important to get this part right

Assurance of Trusted Operating Systems

- How do I know that I should trust someone’s operating system?
- What methods can I use to achieve the level of trust I require?

Assurance Methods

- Testing
- Formal verification
- Validation

Secure Operating System Standards

- If I want to buy a secure operating system, how do I compare options?
- Use established standards for OS security
- Several standards exist

Some Security Standards

- U.S. Orange Book
- European ITSEC
- U.S. Combined Federal Criteria
- Common Criteria for Information Technology Security Evaluation

The U.S. Orange Book

- The earliest evaluation standard for trusted operating systems
- Defined by the Department of Defense in the late 1970s
- Now largely a historical artifact

Purpose of the Orange Book

- To set standards by which OS security could be evaluated
- Fairly strong definitions of what features and capabilities an OS had to have to achieve certain levels
- Allowing "head-to-head" evaluation of security of systems
  - And specification of requirements

Orange Book Security Divisions

- A, B, C, and D
  - In decreasing order of degree of security
- Important subdivisions within some of the divisions
- Requires formal certification from the government (NCSC)
  - Except for the D level

Some Important Orange Book Divisions and Subdivisions

- C2 - Controlled Access Protection
- B1 - Labeled Security Protection
- B2 - Structured Protection

The C2 Security Class

- Discretionary access
  - At fairly low granularity
- Requires auditing of accesses
- And password authentication and protection of reused objects
- Windows NT has been certified to this class

The B1 Security Class

- Includes mandatory access control
  - Using Bell-La Padua model
  - Each subject and object is assigned a security level
- Requires both hierarchical and non-hierarchical access controls

The B3 Security Class

- Requires careful security design
  - With some level of verification
- And extensive testing
- Doesn’t require formal verification
  - But does require “a convincing argument”
- Trusted Mach is in this class

The Common Criteria

- Modern international standards for computer systems security
- Covers more than just operating systems
- Design based on lessons learned from earlier security standards
- Lengthy documents describing Common Criteria

Basics of Common Criteria Approach

- Something of an alphabet soup –
- The CC documents describe
  - The Evaluation Assurance Levels (EAL)
- The Common Evaluation Methodology (CEM) details guidelines for evaluating systems

Another Bowl of Common Criteria Alphabet Soup

- TOE – Target of Evaluation
- TSP – TOE Security Policy
  - Security policy of system being evaluated
- TSF – TOE Security Functions
  - HW, SW used to enforce TSP
- PP – Protection Profile
  - Implementation-dependent set of security requirements
- ST – Security Target
  - Predefined sets of security requirements

What's This All Mean?

- Highly detailed methodology for specifying :
  1. What security goals a system has
  2. What environment it operates in
  3. What mechanisms it uses to achieve its security goals
  4. Why anyone should believe it does so

Logging and Auditing

- An important part of a complete security solution
- Practical security depends on knowing what is happening in your system
- Logging and auditing is required for that purpose

Logging

- No security system will stop all attacks
  - Logging what has happened is vital to dealing with the holes
- Logging also tells you when someone is trying to break in
  - Perhaps giving you a chance to close possible holes

Access Logs

- One example of what might be logged for security purposes
- Listing of which users accessed which objects
  - And when and for how long
- Especially important to log failures

Other Typical Logging Actions

- Logging failed login attempts
  - Can help detect intrusions or password crackers
- Logging changes in program permissions
  - Often done by intruders

Problems With Logging

- Dealing with large volumes of data
- Separating the wheat from the chaff
  - Unless the log is very short, auditing it can be laborious
- System overheads and costs

Log Security

- If you use logs to detect intruders, smart intruders will try to attack logs
  - Concealing their traces by erasing or modifying the log entries
- Append-only access control helps a lot here
- Or logging to hard copy
- Or logging to a remote machine

Auditing

- Security mechanisms are great
  - If you have proper policies to use them
- Security policies are great
  - If you follow them
- For practical systems, proper policies and consistent use are a major security problem

Auditing

- A formal (or semi-formal) process of verifying system security
- "You may not do what I expect, but you will do what I inspect."
- A requirement if you really want your systems to run securely

Auditing Requirements

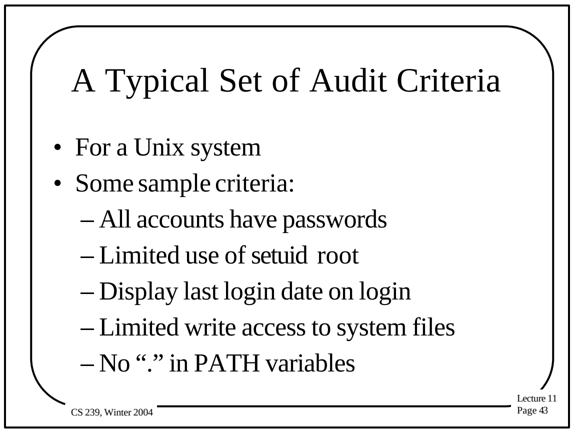
- Knowledge
  - Of the installation and general security issues
- Independence
- Trustworthiness
- Ideally, big organizations should have their own auditors

When Should You Audit?

- Periodically
- Shortly after making major system changes
  - Especially those with security implications
- When problems arise
  - Internally or externally

Auditing and Logs

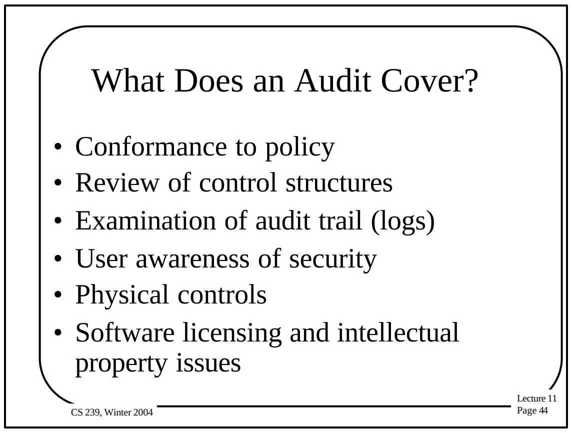
- Logs are a major audit tool
- Some examination can be done automatically
- But part of the purpose is to detect things that automatic methods miss
  - So some logs should be audited by hand



A Typical Set of Audit Criteria

- For a Unix system
- Some sample criteria:
  - All accounts have passwords
  - Limited use of setuid root
  - Display last login date on login
  - Limited write access to system files
  - No "." in PATH variables

CS 239, Winter 2004 — Lecture 11 Page 43



What Does an Audit Cover?

- Conformance to policy
- Review of control structures
- Examination of audit trail (logs)
- User awareness of security
- Physical controls
- Software licensing and intellectual property issues

CS 239, Winter 2004 — Lecture 11 Page 44