

Operating System Security,
Continued
CS 136
Computer Security
Peter Reiher
January 29, 2008

Outline

- Designing secure operating systems
- Assuring OS security
- TPM and trusted computing

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
- Layered design
- Separation and isolation mechanisms
- Virtualization

Secure OS Kernels

- Basic idea is to specify a core set of OS functions
- Keep it small, build it carefully
- All other services build on top of this kernel
- Key idea: if the kernel is safe, everything else must be, too

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

A Disadvantage of Kernelization

- Introduces boundaries in the OS
- Stuff inside is cheaper to work with than stuff outside
 - Since checks and limitations at the boundaries
- Temptation is to keep moving stuff in
 - An irresistible temptation in all major kernelization efforts

A Major Challenge for Kernelization

- What's in and what's out?
- What must you trust to ensure that the rest of the system is secure?
- Depends heavily on how you define “secure”
- Certain types of known attacks still possible against certain “secure” systems
 - They left those attacks out of their definition

Layered OS Design

- A generalization of kernelization
- Define inner layer with high security
- Next layer out builds on that
 - Allowing lower security
- Next layer out provides even lower security
- Outer layers use inner layer services through strong interfaces

Multics and Layered Security

- Multics came before Unix
 - And was a lot more sophisticated and powerful
- Key element of Multics was this layered security model
- Multics is still one of the most sophisticated secure OS designs

Separation and Isolation Mechanisms

- Divide system into components
- Define a secure interface for each
 - Allow communication only over interfaces
- Might ensure no bad stuff crosses boundaries
- Can separate on user or process boundaries
 - Not just functionality
- A pretty successful OS security approach

Uses of Separation and Isolation

- The core idea behind page table security
- Also the core idea behind virtual memory process security
- Domain and type enforcement
 - E.g., as used in SE Linux

Domain and Type Enforcement

- A way of confining security problems into a single domain
 - Commonly abbreviated DTE
- Allows system to specify security domains
 - E.g., the printing domain
- And to specify data types
 - E.g., the printer type

Using DTE

- Processes belong to some domain
 - Can change domains, under careful restrictions
- Only types available to that domain are accessible
 - And only in ways specified for that domain

A DTE Example

- Protecting the FTP daemon from buffer overflow attacks
- Create an FTP domain
- Only the FTP daemon and files in the FTP directory can be executed in this domain
 - And these executables may not be written within this domain
- Executing the FTP daemon program automatically enters this domain

What Happens On Buffer Overflow?

- The buffer overflow attack allows the attacker to request execution of an arbitrary program
 - Say, `/bin/sh`
- But the overflowed FTP daemon program was in the FTP domain
 - And still is
- `/bin/sh` is of a type not executable from this domain
 - So the buffer overflow can't fork a shell

DTE in SE Linux

- SE Linux provides substantial DTE support
- Each process has a domain
- Each object has a type
- Configuration files specify domain interactions and what types they access
- Starting specified programs puts them in particular domains

Types in SE Linux

- Domains are actually specified as types in SE Linux
- Access control matrix specifies which types can interact with other types
- So a process is given a type
 - Which implies what other types it can access

Example of SE Linux Type Enforcement

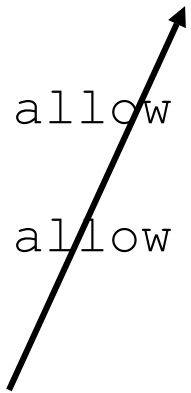
- Files in `/etc` are mostly limited to access by few sysadmin process types
- But `/etc` also contains `/etc/aliases`
 - Which the mail program must access
 - And everyone uses the mail program
- So rules are set up to allow the sendmail process' type to access `/etc/aliases`

Types in the Example

- The sendmail process is assigned type `sendmail_t`
- The `/etc/aliases` file is assigned type `etc_aliases_t`
- Other mail related files and directories also get their own types

The SE Linux sendmail Rules

```
allow sendmail_t etc_aliases_t:file
    { read write };
allow sendmail_t etc_mail_t:dir
    { read search add_name remove_name };
allow sendmail_t etc_mail_t:file
    { create read write unlink };
```



This rule allows processes of `sendmail_t` type to access files of `etc_aliases_t` type for read and write

Without regard for which user started the process

Contrast With Standard Unix File Access Control

- What permissions do you put on `/etc/aliases`?
- Must be sufficient to allow normal work
 - So must allow read and write
- But not too much to allow anyone to read and write anything there

Standard Unix Solution

- Run `sendmail` setuid to a special user named `mail` or something
- Set ownership of `/etc/aliases` to `mail` user
- Allow any user to run the `sendmail` program
- Why is SE Linux approach better?

Some Differences

- Don't need to create fake users like `mail`
- You've centralized the security-critical access control rules
 - No worry that a file somewhere had the wrong permission bits
- The `sendmail` process runs under the identity of the calling user
 - No need for “real” and “effective” uids
- Clean, extensible abstraction

Virtualization

- A popular modern approach
- Run untrusted stuff in a virtual machine
- Only allow VM to access things you don't worry about
- Thus, untrusted stuff can't screw you over

Approaches to Virtualization

- Native OS virtualization facilities
 - Meta-OS runs various virtual machines on same real machine
 - Developed in 1970s for mainframes
- Programming language based VM
 - E.g., Java
- VM package tacked on to operating system
 - E.g., VMWare and Parallels

Challenges to Using Virtualization

- Securely confining code to a VM
 - Often, there are ways for it to get out
- Proper allocation of processes and resources to a VM
 - If things have to share data, must they be in the same VM?
 - If not, how do you keep them in?
- Efficiency
- Multiplexing real hardware

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

Testing

- Run a bunch of tests against the OS to demonstrate that it's secure
- But what tests?
- What is a sufficient set of tests to be quite sure it works?
- Not a strong proof of system security
- But what is used most often

Formal Verification

- Define security goals in formal terms
- Map either OS design or implementation to those terms
- Use formal methods to “prove” that the system meets security goals

Challenges in Formal Verification

- Defining security goals properly
- Accurate mapping of real system to formal statements
 - This one is a real killer
- High overhead of running verification methods for realistic systems

Validation

- Define desired system security
- In terms of:
 - Features provided
 - Architectural design
 - Processes used in creating the system
 - Evaluation methodology
 - Possibly other dimensions
- Use standardized procedure to demonstrate your system fits this profile

Validation and Standards

- Validation is usually done against a pre-defined standard
- Wide agreement that standard specifies a good system
- So you just have to demonstrate you fit the standard

Benefits of Validation

- Allows head-to-head comparisons of systems
- Allows varying degrees of effort to determine system security
- Allows reasonably open and fair process to determine system security

Disadvantages of Validation

- Only as good as its standards
- Doesn't actually prove anything
- Can be very expensive

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 control
 - At fairly low granularity
- Requires auditing of accesses
- And password authentication and protection of reused objects
- Windows NT was certified to this class

The B1 Security Class

- Includes mandatory access control
 - Using Bell-La Padula 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 was in this class

Why Did the Orange Book Fail?

- Expensive to use
- Didn't meet all parties' needs
 - Really meant for US military
 - Inflexible
- Certified products were slow to get to market
- Not clear certification meant much
 - Windows NT was C2, but didn't mean NT was secure in usable conditions
- Review procedures tied to US government

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 describe the 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

How Does It Work?

- Someone who needs a secure system specifies what security he needs
 - Using CC methodology
 - Either some already defined PPs
 - Or he develops his own
- He then looks for products that meet that PP
 - Or asks developers to produce something that does

How Do You Know a Product Meets a PP?

- Dependent on individual countries
- Generally, independent labs verify that product meets a protection profile
- In practice, a few protection profiles are commonly used
- Allowing those whose needs match them to choose from existing products

Status of the Common Criteria

- In wide use
- Several countries have specified procedures for getting certifications
 - And there are agreements for honoring other countries' certifications
- Many products have received various certifications

Problems With Common Criteria

- Expensive to use
- Slow to get certification
 - Ensuring certified products are behind the market
- Practical certification levels might not mean that much
 - Windows 2000 was certified EAL4+
 - But kept requiring security patches . . .
- Perhaps more attention to paperwork than actual software security

TPM and Trusted Computing

- Can special hardware help improve OS security?
- Perhaps
- TPM is an approach to building such hardware
- The approach is commonly called “trusted computing”

What Is TPM?

- Special hardware built into personal computers
 - And other types of machines
- Tamperproof, special purpose
- Effective use requires interaction with software
 - Especially OS software
- Defined as a set of open standards

What Does TPM Hardware Do?

- Three basic core functionalities:
 - Secure storage and use of keys
 - Secure software attestations
 - Sealing data
- These functions can be used to build several useful security features

TPM Key Storage

- Keys are stored in a tamperproof area
- TPM hardware can generate RSA key pairs
 - Using true random number generator
- Each TPM chip has one permanent endorsement key
- Other keys generated as needed

The Endorsement Key

- Created when the chip was fabricated
- Used to sign attestations
 - To prove that this particular machine made the attestation
- A public/private key pair
 - Private part never leaves the trusted hardware

TPM Cryptography

- Some TPM hardware includes encryption and decryption functions
- To ensure keys are never outside a tamperproof perimeter

TPM Attestations

- Allows TPM to provide proof that a particular piece of software is running on the machine
 - An OS, a web browser, whatever
- Essentially, a signature on a hash of the software

An Example of an Attestation

- What version of Linux is running on this machine?
- TPM (with appropriate SW support) hashes the OS itself
- Signs the hash with its attestation key
- Sends the signature to whoever needs to know

Secure TPM Boot Facilities

- Use attestations to ensure that the boot loader is trusted code
- The trusted boot loader then checks the OS it intends to load
 - Trusted attestations can tell the boot loader if it's the right one
 - Bail out if it's not the right one
- Can prevent an attacker from getting you to boot a corrupted kernel

Sealing Data With TPM

- Encrypt the data with keys particular to one machine
 - Keys stored by TPM
- Data can only be decrypted successfully on that machine
- Can also seal storage such that only a particular application can access it

The TPM Controversy

- TPM can be used for many good security purposes
- But some believe it takes too much power from the user
 - E.g., can require user to prove he's running a particular browser before you give him a file
 - Or seal a file so only the owner's application can read it
- Many (but not all) critics worry especially about DRM uses
 - Also serious issues about companies using it to achieve anti-competitive effects
- Serious questions about practicality based on patching, various releases, etc.
 - Will you have to accept attestations for all of them?