Multilevel Security

Petr Ročkai

May 7, 2019

Overview

1. Access Control
2. Isolation
3. Covert Channels

Part 1: Access Control

Access Control

- there are 3 pieces of information
  - the subject (user)
  - the verb (what is to be done)
  - the object (the file or other resource)
- there are many ways to encode this information

Subjects

- typically, those are (possibly virtual) users
  - sub-user units are possible (e.g. programs)
  - roles and groups could also be subjects
- the subject must be named (names, identifiers)
- processes actually carry out the actions

Objects

- anything that can be manipulated by programs
  - although not everything is subject to access control
- could be files, directories, sockets, shared memory, ...
- object names depend on their type
  - file paths, i-node numbers, IP addresses, ...

Verbs

- the available “verbs” (actions) depend on object type
- a typical object would be a file
  - files can be read, written, executed
  - directories can be searched or listed or changed
- network connections can be established &c.

Access Control Policy

- decides which actions are allowed
- site- or institution-specific
- dynamic – objects and subjects come and go
- many ways to encode & maintain the policy

Security Labels

- an alternative to naming subjects & objects
  - we attach labels to them instead
- the security policy refers to labels
- how are labels assigned to objects?
  - labelling each object manually is impractical

Labelling Policy

- attach labels based on rules
  - applies both to subjects and objects
- label transitions for subjects
  - subjects are active participants
  - their actions can cause their labels to change

Label-Based Access Policies

- based on rules which refer to labels
  - a small 'programming language'
  - writing rules requires expert knowledge
- does not name subjects or objects directly
- but the overall policy includes the labelling

Ownership

- subjects can own objects
  - often by virtue of creating the objects
  - but ownership can be transferred
- special privileges & responsibilities
  - owned objects count towards resource limits

Mandatory vs Discretionary AC

- discretionary is the ‘traditional’ model
  - ownership implies control over access rights
- mandatory access control disconnects the two
  - owners cannot control access rights
  - management of the policy is a separate role

Mandatory + Discretionary

- those types of policies can coexist
  - e.g. some discretionary control is allowed
  - but the mandatory policy takes precedence
- purely mandatory access control is impractical
  - too much communication overhead

Policy Management

- centralised – one authority makes policy decisions
  - usually associated with mandatory systems
  - inflexible, high latency
- decentralised – multiple parties make decisions
  - less secure, typical for discretionary systems
  - more flexible, lower latency

Enforcement: Hardware

- all enforcement begins with the hardware
  - the CPU provides a privileged mode for the kernel
  - DMA memory and IO instructions are protected
- the MMU allows the kernel to isolate processes
  - and protect its own integrity

Enforcement: Kernel

- kernel uses hardware facilities to implement security
  - it stands between resources and processes
  - access is mediated through system calls
- file systems are part of the kernel
- user abstractions are part of the kernel

Enforcement: System Calls

- the kernel acts as an arbitrator
- a process is trapped in its own address space
- processes use system calls to access resources
  - kernel can decide what to allow
  - based on its access control model and policy

API-Level Access Control

- access control for user-level resources
  - things like contact lists, calendars, bookmarks
  - objects not provided by the operating system
- enforcement e.g. via a virtual machine
  - not applicable to execution of native code

Programs as Objects and Subjects

- program: passive (file) vs active (process)
  - only a process can be a subject
  - but program identity is attached to the file
- rights of a process may depend on its program
- a process exercises rights on the behalf of a user

Trusted vs Untrusted Code

- users perform actions on a computer
    - but they are always actually done by a program
    - the user is not directly in control
- the program should do what the user told it to
    - but how do we ensure this is so?
    - trust = belief that programs do what they should

Trojan Horse

- program designed to abuse misplaced trust
- presents some desirable functionality
- but also performs undesirable hidden actions
  - usually concealed from the user (see above)
- trojans present a major security risks

Security Objectives

- integrity
  - data must not be tampered with
  - crucial for programs, communication
- secrecy (confidentiality)
  - data must not be revealed

Metapolicies

- policies about policies
  - dictates what an access control policy can do
- how to write a secure access policy?
  - enforce a known secure meta-policy
  - conformance can be checked automatically

Multi-Level Security

- a meta-policy designed for hierarchical institutions
  - system of user ranks / security clearances
  - data is stratified too (e.g. by confidentiality)
- two basic types
  - secrecy-preserving (Bell-LaPadula)
  - integrity-preserving (Biba)

Confidentiality Objectives

- non-interference (stronger)
  - confidential actions cannot be observed at all
- non-deducibility (weaker)
  - confidential actions cannot be reliably inferred
  - only gives a probabilistic guarantee

Bell-LaPadula

- MLS meta-policy for confidentiality
- enforces 2 basic security properties
  - no read up: clearance is required for access
  - no write down: prevent information leaks
- special rights required for declassification

Biba

- MLS meta-policy for integrity
- inverse of Bell-LaPadula:
  - no write up: integrity is preserved
  - no read down: prevent confusion

Part 2: Isolation

Integrity

- isolated units must not influence each other
- prerequisite to all other guarantees
- example integrity violations:
  - a process overwriting memory of another process
  - a website in one tab changing text in another tab

Secrecy

- units must not observe other units
  - especially applies to obtaining data
- often much harder than integrity
  - information leaks are ubiquitous
  - often due to innocent-looking details

Resource Sharing

- resources are costly → sharing
- shared resources weaken isolation
  - units can indirectly influence each other
  - or at least learn something

Communication

- a completely isolated system is useless
- but communication channels weaken isolation
  - both isolation and communication are desirable
  - there is a trade-off to be found

Memory Management Unit

- is a subsystem of the processor
- takes care of address translation
  - user software uses virtual addresses
  - the MMU translates them to physical addresses
- the mappings can be managed by the OS kernel

Paging

- physical memory is split into frames
- virtual memory is split into pages
- pages and frames have the same size (usually 4KiB)
- frames are places, pages are the content
- page tables map between pages and frames

Processes

- process is primarily defined by its address space
  - address space meaning the valid virtual addresses
- this is implemented via the MMU
- when changing processes, a different page table is loaded
  - this is called a context switch
- the page table defines what the process can see

Memory Maps

- different view of the same principles
- the OS maps physical memory into the process
- multiple processes can have the same RAM area mapped
  - this is called shared memory
- often, a piece of RAM is only mapped in a single process

Page Tables

- the MMU is programmed using translation tables
  - those tables are stored in RAM
  - they are usually called page tables
- and they are fully in the management of the kernel
- the kernel can ask the MMU to replace the page table
  - this is how processes are isolated from each other

Kernel Protection

- kernel memory is usually mapped into all processes
  - this improves performance on many CPUs
  - (until meltdown hit us, anyway)
- kernel pages have a special 'supervisor' flag set
  - code executing in user mode cannot touch them
  - else, user code could tamper with kernel memory

Inter-Process Communication

- punches controlled gaps into process isolation
- different types, different risks
  - message passing, event handlers (safest)
  - streams of bytes
  - shared memory (most risky)

File Systems

- those are typically shared between all processes
- easily turned into an IPC mechanism
- usually very good access control coverage
  - but not perfect (e.g. free space, free i-nodes)
  - and also easily defeated if discretionary

BSD Jails

- a multi-process isolation mechanism
  - an entire process subtree is isolated as a unit
  - resource sharing is unrestricted within the group
- restricted view of file systems
  - but does not cover free space either
- restricted IPC, network capabilities

Linux Namespaces

- another resource isolation mechanism
- similar capabilities but finer-grained control
    - can isolate each subsystem individually
- many different resources
    - networking, filesystem, IPC
    - process tables, user tables

Virtualisation

- isolation of multiple operating systems on a singe host
- coarse-grained: block devices, network interfaces
  - access control policy becomes much simpler
  - simple policy → fewer bugs and mishaps
- high overhead (multiple operating system copies)

Sandboxing Overview

- artificial restriction of program capabilities
  - e.g. by giving up access rights
  - done for security reasons
- designed to limit damage in case of compromise
- voluntary (defensive programming), involuntary

Language-Based Sandboxing

- isolation at the level of a programming language
- type-based: static isolation guarantees
  - Safe Haskell, Modula 3, ...
- runtime-based: dynamic enforcement
  - JVM, JavaScript

OS-Level Sandboxing

- file system restrictions (`chroot`, `unveil`)
- system call restrictions
  - `systrace` – fine-grained, involuntary
  - `pledge` – coarse-grained, voluntary
  - targeted SELinux policies (involuntary)
  - AppArmor, TOMOYO Linux (also involuntary)

Google Native Client

- sandboxing based on dynamic recompilation
- similar to language-level sandboxing
  - but for native machine code
  - with a minimal performance penalty
- deprecated in favour of WebAssembly

Part 3: Covert Channels

Definition

- a mechanism which allows communication
  - even though it was not designed for that
  - and hence is not regulated by access control
- can be used for malicious exfiltration of data
  - the bad actor controls both endpoints

Motivation

- covert channels threaten properties of MLS
  - i.e. they may violate the Bell-LaPadula axioms
  - not applicable in the integrity (Biba) picture
- can be used to exfiltrate confidential data
  - using a trojan or some other attack vector

Anatomy

- a covert channel has 2 ends: writer & reader
- the writer, which runs with a security clearance
  - this would be the trojan or other exploit
- the reader, which runs without a clearance
  - can freely create unclassified files
  - or even directly send data across the network

Comparison to Side Channels

- side channels are information leaks
  - they work without compromising the target
  - rely on passive observation alone
- a covert channel relies on cooperation
  - both ends must be under the control of the attacker

Example

- (lack of) free space in the file system
  - not subject to traditional access control
- the writer can fill up / free up space
- the reader checks if writing files is possible

Synchronisation

- covert channels are usually unidirectional
- need opposite channel for synchronisation
  - may be a regular, open channel, if available
  - a sufficiently precise clock works too

Covert Channel Properties

- bandwidth – amount of data per time unit
  - varies wildly depending on specific channels
- noise – percentage of bits that get flipped
  - covert channels are usually not reliable
  - noise reduces effective bandwidth

Shared Resources

- each shared resource is a potential covert channel
- CPU, RAM, filesystem, network, …
- multiple reasons for sharing
  - conserve resources (avoidable)
  - facilitating communication (mostly unavoidable)

Further Examples (writer → reader)

- busy-loop → detection of slow CPU
- file locks → unable to open a file
- memory pressure → page faults (swapping)
- firehose data to disk → slow disk access

Discovery

- covert channels are a property of the system
- basic strategy: manual review / inspection
- better: system modelling and formal analysis
  - either semi-manual (covert tree flows)
  - automated – theorem provers / solvers

Mitigation / Defence

- reducing sharing
  - fewer shared resources = fewer channels
  - increases price
- reduce bandwidth – e.g. query rate limiting
- increase / inject noise