Security

Advanced Operating Systems
(263-3800-00)

Timothy Roscoe

Thursday 2 December 2010

Systems@ETH Zürich

Overview

Systems@ETH Zürich

- Security is a huge field in its own right
- This lecture covers a selection of topics from the OS designer's perspective
  - There is a focus on security mechanisms
- ***Information Security*** course as background

Outline

Systems@ETH Zürich

- Security introduction (from an OS perspective)
  - What does it mean for a system to be secure?
  - Trusted computing base
- Security mechanisms
  - Access control matrix
  - Access control lists (review)
  - Capabilities (in depth)
- Non-discretionary access control
  - Mandatory access control
- Decentralised information flow control (labels)

Outline

- Security introduction (from an OS perspective)
  - What does it mean for a system to be secure?
  - Trusted computing base
- Security mechanisms
  - Access control matrix
  - Access control lists (review)
  - Capabilities (in depth)
- Non-discretionary access control
  - Mandatory access control
- Decentralised information flow control (labels)

What does it mean for a system to be secure?

Systems@ETH Zürich

- We all have different ideas about what security means
- Saying that "a system is secure" is meaningless without specifying the ***security policy***

Definition:

A set of rules and practices that specify or regulate how a system or organization provides security services to protect sensitive and critical system resources. [RFC 2828]

- Policy specifies allowed states of the system
- Security mechanisms are used to enforce the policy

Trusted computing base (TCB)

Definition

The totality of protection mechanisms within a computer system, including hardware, firmware, and software, the combination of which is responsible for enforcing a security policy. [RFC 2828]

- That part of the system that must be relied upon to enforce a security policy

. . . or that can circumvent the security policy

- Trusted by definition,

but that doesn't make it trustworthy

How can we make the TCB more trustworthy?

Systems@ETH Zürich

- Testing
- Source code inspection
  - . . . but how many bugs do you think are left in Linux?
- Assurance standards: orange book, common criteria
  - Various levels
  - Windows, Linux, and many others have been certified
- Formal verification
  - seL4 / L4.verified projects [Klein et al., 2009]
- Make it smaller!
  - Less code to trust
  - Less code to inspect/verify

Outline

- Security introduction (from an OS perspective)
  - What does it mean for a system to be secure?
  - Trusted computing base
- Security mechanisms
  - Access control matrix
  - Access control lists (review)
  - Capabilities (in depth)
- Non-discretionary access control
  - Mandatory access control
- Decentralised information flow control (labels)

Access control matrix

Representation/definition of permissible operations in a system:

	Objects				
Subjects	User1	User2	User3	File1	File2
User1		Send msg		RW	R
User2	Send msg				RW
User3	Set passwd	Set passwd	Set passwd		R

Subjects: users, processes, groups, etc.

Objects: other users/processes, files, memory objects, etc.

Privileges/rights: depends on object

(for file: read, write, execute, etc.)

Access control matrix properties

- Dynamic data structure, frequent changes
- Very sparse with many repeated entries
- Impractical to store explicitly

- Most common discretionary mechanisms:
  - Access control list: stores a column (who can access this)
  - Capabilities: store a row (what can access)

Issues for discretionary access control

- Propagation: Can a subject grant access to another?
- Restriction: Can a subject propagate a subset of its own rights?
- Revocation: Can access, once granted, be revoked?
- Amplification:
  Can an unprivileged subject perform restricted operations?
- Determination of object accessibility:
  Which subjects have access to a particular object?
  - Is an object accessible by any subject?
    (garbage collection)
- Determination of subject's protection domain:
  Which objects are accessible to a particular subject?

Access control lists

- Implemented by most commodity systems
- ACL associated with object
  - Propagation: meta right (eg. owner may chmod)
  - Restriction: meta right
  - Revocation: meta right
  - Amplification: protected invocation right (eg. setuid)
  - Accessibility: explicit in ACL
  - Protection domain: hard (if not impossible) to determine
- Usually condensed via groups / classes
- Can have negative rights
- Sometimes implicit (eg. UNIX process hierarchy)

UNIX ACLs

Systems@ETH Zürich

- Despite modern terminology, classic UNIX privileges are a (restricted) ACL representation:

```
drwx--x--x 2 andrewb andrewb 4.0K Mar 9 22:28 dir1
-rwxr-xr-x 1 andrewb andrewb 1.3K Mar 9 22:26 file1
-rw-r--r-- 1 andrewb andrewb  13K Mar 9 22:26 file2
-rw------- 1 andrewb andrewb  30K Mar 9 22:26 file3
-rw-rw---- 1 andrewb group1   92K Mar 9 22:27 file4
```

rights

subject (owner)

subject (group)

object

- Permissions for *other* are an implicit group of subjects

Capabilities: introduction

- Capability list associated with subject
- Each capability confers a certain right to its holder
  - Propagation: copy/transfer capabilities between subjects
  - Restriction: requires creation of new (derived) caps
  - Revocation: requires invalidation of caps from all subjects
  - (may be difficult)
  - Amplification: special invocation capability
  - Accessibility: requires inspection of all capability lists
  - (hard if not impossible to determine)
  - Protection domain: explicit in capability list
- Few successful commercial systems:
  - IBM System/38 (AKA AS/400, iSeries, System i, . . .), KeyKOS

Capabilities

Systems@ETH Zürich

- Main advantage of capabilities is fine-grained access control
  - ☑ Easy to provide access to specific subjects
  - ☑ Easy to delegate permissions to others
  - – See *The Confused Deputy*, Norm Hardy
- A cap presents prima facie evidence of the right to access
  - – Think of it as a key
- Consists of object identifier and a set of access rights
  - – Implies object naming
  - – Any representation must protect capabilities against forgery
- How are caps implemented and protected?

  Tagged: protected by hardware

  Partitioned: protected by software

  Sparse: protected by sparsity
  (probabilistically secure, like encryption)

Tagged capabilities, aka Hardware capabilities

Systems@ETH Zürich

Extra tag bit with every memory word (or group thereof)

- Tag identifies capabilities
- Capabilities may be used and copied like "normal" pointers
- Hardware checks permissions when dereferencing capability
- Modifications turn the tag off (reverting caps to plain data)
- Only the kernel can turn a tag bit on

☑ Propagation easy

- Restriction requires kernel to create new (weaker) capability

☒ Revocation virtually impossible (requires memory scan)

☒ Accessibility virtually impossible to determine

Tagged capabilities outside RAM

- Disk has no tags
- AS/400 simulates them by restricting physical I/O to the low-level OS
  - Extra bit stored for every word on disk
  - On page-out, page must be scanned and tags collected
  - On page-in, tags are reconstructed
  - Significant processing overhead on all disk I/O

Tagged capabilities: summary

- Secure through hardware protection
- Convenient for applications (appear as normal pointers)
- Checked by hardware $\Rightarrow$ fast validation
- Capability hardware is complex (hence slow)
- Separate mechanisms required for I/O and distribution

Partitioned capabilities

System maintains capabilities for each process, eg. as a capability list (clist)

- User code uses only handles (indirect references) to caps
- System validates access when performing any privileged operation (eg. mapping a page)
  - Validation is explicit at syscall time
  - Propagation: system call to copy a cap between clients
  - Restriction: invoke kernel to create new cap
  - Revocation: invoke kernel to remove cap from clist
  - Accessibility: requires scanning all clists
  - Protection domain: explicitly represented in clist
- Used in Hydra, Mach, KeyKOS, EROS, seL4, Barrelfish, many others

Partitioned capabilities

- Secure through kernel protection
  - Real caps live only in kernel space
- Validation at mapping/invocation time
  - Apps use “normal” pointers

☑ Fast validation possible

- (for memory objects, validation is cached byMMU)
- Propagation requires kernel intervention

☑ Reference counting and revocation possible with kernel support

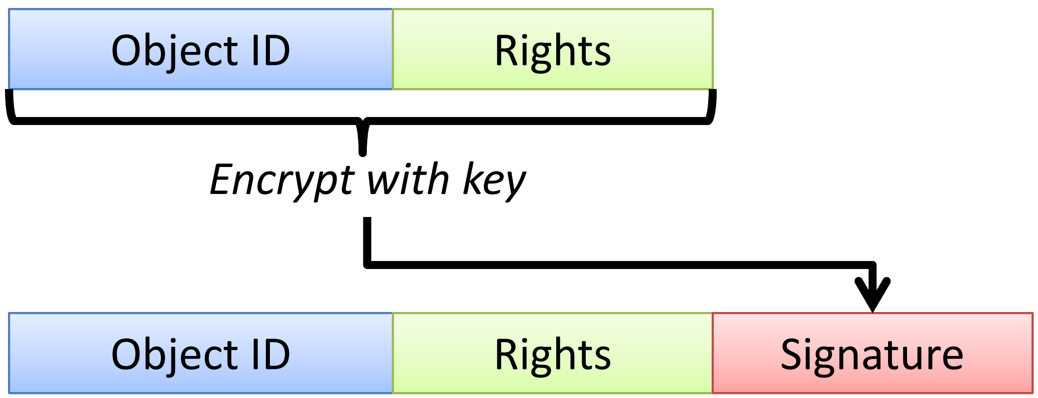
☑ No special hardware requirements

Sparse capabilities

Systems@ETH Zürich

- Basic idea similar to encryption
- Add bit string to capability
  - Makes valid capabilities a tiny subset of the capability space
  - Secure by infeasibility of exhaustive search of cap space

Encrypted sparse capabilities



Signature: bit string is encrypted object info

- Cap consists of object ID, rights, and signature
- Signature = object ID and rights encrypted with a private key
- Validated by checking signature

Password sparse capabilities

Object ID	Password

Capability

Global object table

OID	Password	Rights

Password: bit string is random data

- Cap consists of object ID, and password
- Rights determined by looking up password in a global object table

Sparse capabilities

- Sparse caps are regular user-level objects
- Can be passed like other data
  - Similar to tagged caps, but without hardware support
  - Validated at invocation time (either explicitly or implicitly)

Issues:

  - Full mediation requires extra support
    - See Mungi
  - High amplification of leaked data
    - Problem with covert channels

Aside: Physical memory management with caps

[Elkaduwe et al., 2008, Klein et al., 2009]

- Problem: allocation and management of physical memory
- Most kernels use dynamic allocation (malloc) for metadata
  - What do you do when it runs out?
- seL4 model (also used in Barrelfish):
  - All physical memory manipulated as partitioned capabilities
  - Memory not used for bootstrap is initially *untyped*
  - User controls allocation by *retyping* capabilities to:

    Frame may be mapped into a page table
    Thread kernel TCB and metadata for a new thread
    Endpoint targets for IPC
    VNode hardware page tables
    CNode tables used to store further capabilities
  - No dynamic allocation necessary in kernel!

Outline

- Security introduction (from an OS perspective)
  - What does it mean for a system to be secure?
  - Trusted computing base
- Security mechanisms
  - Access control matrix
  - Access control lists (review)
  - Capabilities (in depth)
- Non-discretionary access control
  - Mandatory access control
- Decentralised information flow control (labels)

Non-discretionary access control

- Both ACLs and capabilities are *discretionary access control*
  - Discretionary, a user with access to data can pass it on

Alternatives:

- Mandatory (system-controlled) policies
  - eg. certain users never access certain objects
  - No user can change these
  - Focus on restricting information flow
- Role-based policies
  - Subjects can take on specific pre-defined roles
  - Access rights depend on role

Mandatory access control (MAC)

Example: Bell-LaPadula model

- Every object o has a security classification L(o)
- Every subject s has a security clearance L(s)
- Classifications & clearances form hierarchical security levels
  - eg. top secret > secret > confidential > unclassified
- Rule 1 (*no read up*)
  - s can read o only if L(s) ≥ L(o)
  - standard confidentiality
- Rule 2 (** property*)
  - s can write o only if L(s) ≤ L(o)
  - prevents leakage (accidental or intentional)
  - problems: logging, command chain
  - need way to declassify data

Outline

- Security introduction (from an OS perspective)
  - What does it mean for a system to be secure?
  - Trusted computing base
- Security mechanisms
  - Access control matrix
  - Access control lists (review)
  - Capabilities (in depth)
- Non-discretionary access control
  - Mandatory access control
- Decentralised information flow control (labels)

Problems with MAC

- ☑ Under MAC, you don't need to trust those who access data not to leak it
- ☒ However, only the system administrator may declassify data
  - Not very flexible; leads to data becoming more classified
- ☒ Single system-wide policy
  - Small pre-defined number of categories
- Decentralized information flow control:
  - Allows a large (essentially unlimited) number of categories
  - Extends MAC with the notion of an owner for each category
  - The owner may declassify data within that category
  - Data may belong to multiple categories

Motivation

Why do I care about information flow control?

- Asbestos: you don't want to trust a dynamic web service implementation not to leak private data [Efstathopoulos et al., 2005]
- HiStar: you don't want to trust your virus scanner, login daemon, VPN client, … [Zeldovich et al., 2006]
- Both systems implement essentially the same model
  - We'll use the HiStar terminology/notation

Labels

- Every object has a label
- Label specifies how tainted the object is *for every category*
  - Categories are 61-bit identifiers, may be freely created
- The label of a subject acquires taint based on its activities (eg. reading data)
- MAC checks apply to every operation for every category
  - An action is disallowed if it would convey information from more to less tainted objects in any category
- There are 5 taint levels:

  * has untainting privileges in this category

  0 cannot be written/modified by default

  1 default level – no restrictions

  2 cannot be untainted/exported by default

  3 cannot be read/observed by default

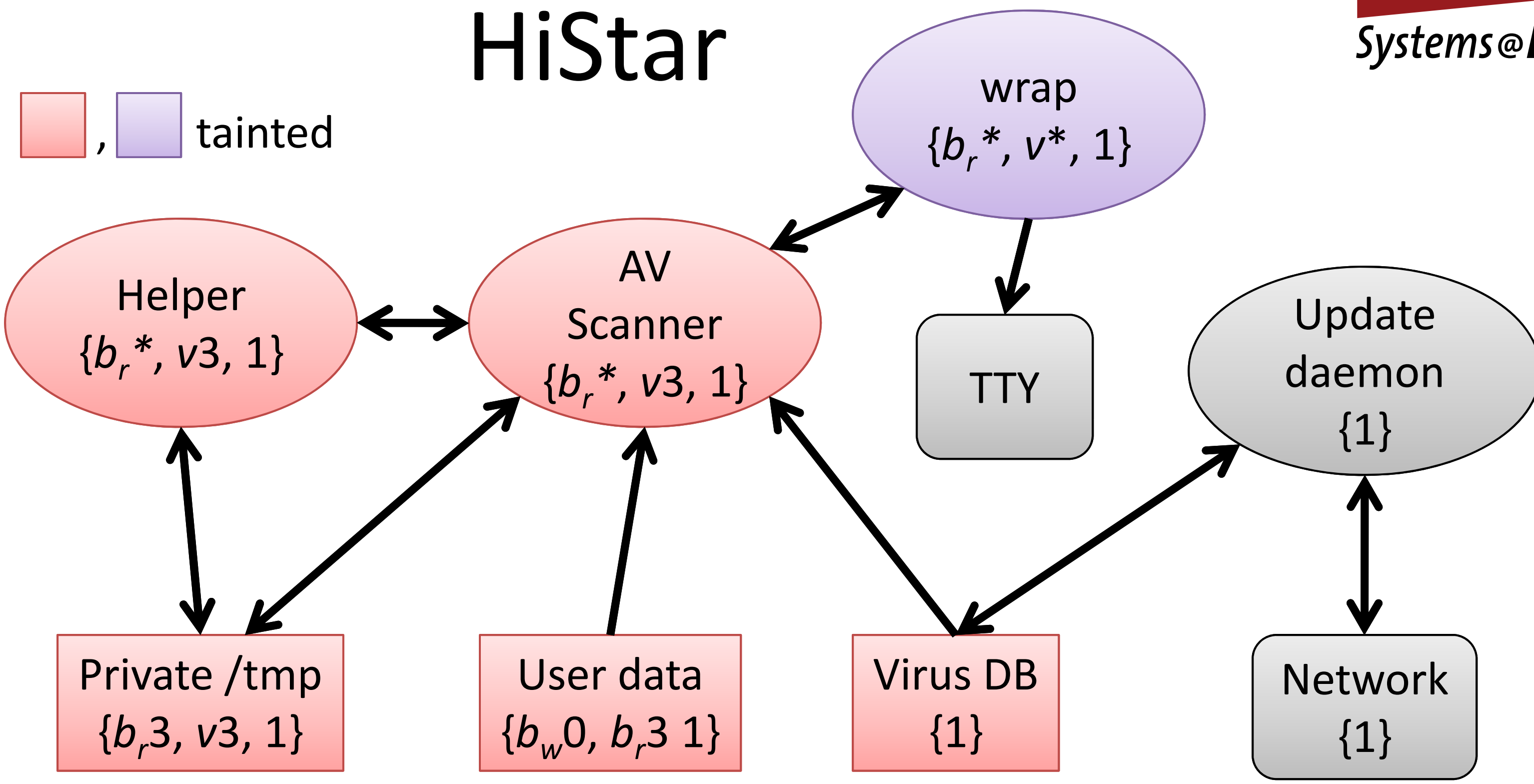
Star level

Systems@ETH Zürich

- Subjects (threads and gates in HiStar) with the star (*) level in a category are trusted for that category
  - Can disregard MAC rules (i.e. declassify) *but only within that category*
  - Called the **owner** of a category

- Essential difference from classic (military) MAC systems

Untrusted virus scanner on HiStar



- wrap trusted to untaint scan results and display them
- {b_r*;v3; 1}: helper owns the b_r category, has level 3 in the v category, and level 1 otherwise
- b_r = user's read category, b_w = user's write category

Challenges for DIFC

- Efficient implementation of labels and label comparison
  - Time and memory
- Usability issues
  - How do you pick the labels?
  - How do you debug taints?
    Once an operation fails it's too late!

Summary

- Access control mechanisms are provided by the OS to enforce a security policy
- Common discretionary mechanisms: ACLs and capabilities
  - Capabilities allow fine-grained access control
  - Caps enable simple and controlled delegation of privilege
- MAC provides information flow control, but is unsuited to a general-purpose system
- Decentralised information flow control (Asbestos, HiStar) tries to address this

Summary

Other interesting topics we haven’t covered:

- Role-based access control
- Principle of least authority/privilege
- Trusted computing / TPM
- Covert channels
- Formal verification

References

- Efstathopoulos, P., Krohn, M., VanDeBogart, S., Frey, C., Ziegler, D., Kohler, E., Mazières, D., Kaashoek, F., and Morris, R. (2005). **Labels and event processes in the Asbestos operating system**. In 20th SOSP, pages 17–30.
- Elkaduwe, D., Derrin, P., and Elphinstone, K. (2008). **Kernel design for isolation and assurance of physical memory**. In IIES '08: Proc. 1st workshop on isolation and integration in embedded systems, pages 35–40.
- Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., andWinwood, S. (2009). **seL4: Formal verification of an OS kernel**. In Proceedings of the 22nd ACM Symposium on Operating Systems Principles, pages 207–220.
- Zeldovich, N., Boyd-Wickizer, S., Kohler, E., and Mazières, D. (2006). **Making information flow explicit in HiStar**. In 7th OSDI.