CSCI 331: Introduction to Computer Security

Lecture 20: Information Flow

Instructor: Dan Barowy

Williams

Topics

Reference monitors

Bell-LaPadula model

Denning's information flow model

Your to-dos

1. Final project, **due Sunday, Dec 10 at 10pm**.
2. Resubmissions due **Sunday, Dec 17**.
3. If you want to talk about your project (or anything else), I have office hours:
   - **Today**, from 4-5:30pm
   - **Tomorrow**, from 12:30-1:30pm

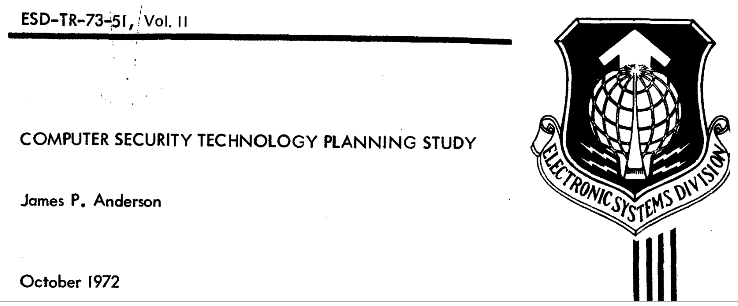
Paper discussion (Thompson)

Reference Monitor

Reference Monitor

Invented by **James P. Anderson** in 1972.

A principled mechanism for mitigating attacks by limiting access based on rules.



ESD-TR-73-51, Vol. II

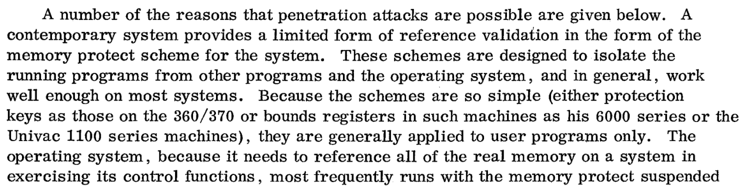
COMPUTER SECURITY TECHNOLOGY PLANNING STUDY

James P. Anderson

October 1972

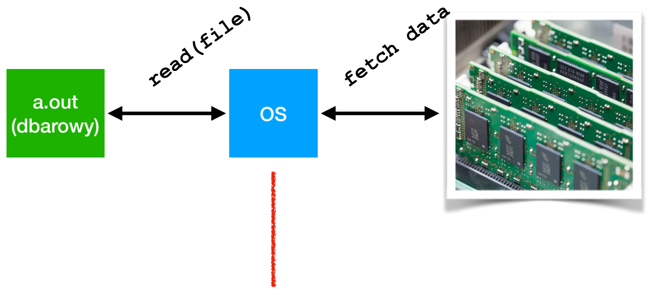
ELECTRONIC SYSTEMS DIVISION

Reference Monitor



A number of the reasons that penetration attacks are possible are given below. A contemporary system provides a limited form of reference validation in the form of the memory protect scheme for the system. These schemes are designed to isolate the running programs from other programs and the operating system, and in general, work well enough on most systems. Because the schemes are so simple (either protection keys as those on the 360/370 or bounds registers in such machines as his 6000 series or the Univac 1100 series machines), they are generally applied to user programs only. The operating system, because it needs to reference all of the real memory on a system in exercising its control functions, most frequently runs with the memory protect suspended

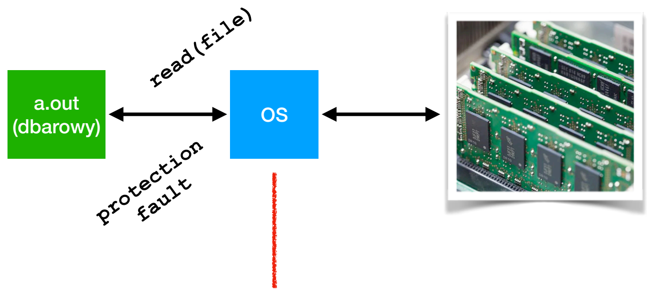
Traditional OS Design



"Does `dbarowy` **have access to** `file`**?"**

yes

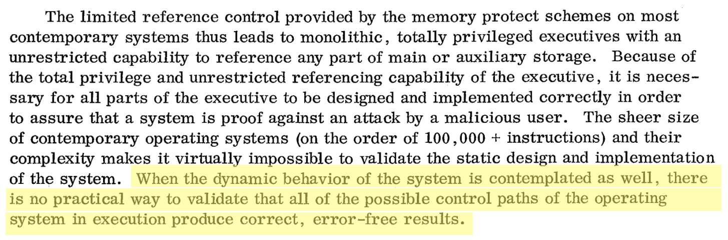
Traditional OS Design



"Does `dbarowy` **have access to** `file`**?"**

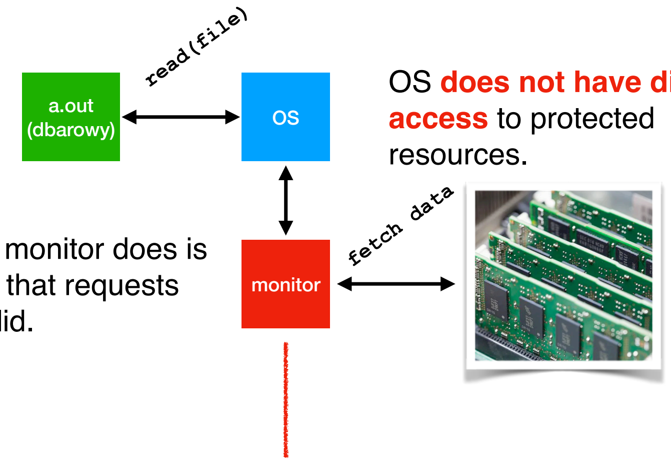
no

Reference Monitor



The limited reference control provided by the memory protect schemes on most contemporary systems thus leads to monolithic, totally privileged executives with an unrestricted capability to reference any part of main or auxiliary storage. Because of the total privilege and unrestricted referencing capability of the executive, it is necessary for all parts of the executive to be designed and implemented correctly in order to assure that a system is proof against an attack by a malicious user. The sheer size of contemporary operating systems (on the order of 100,000 + instructions) and their complexity makes it virtually impossible to validate the static design and implementation of the system. When the dynamic behavior of the system is contemplated as well, there is no practical way to validate that all of the possible control paths of the operating system in execution produce correct, error-free results.

Reference Monitor Design



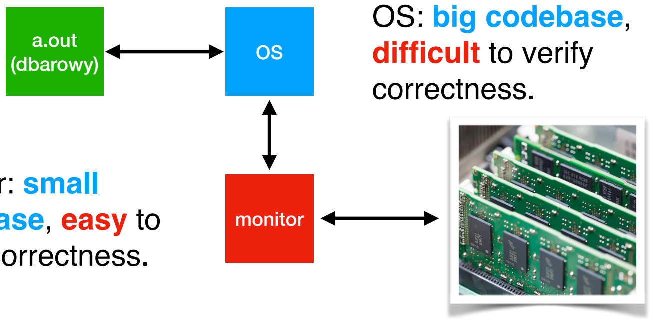
OS **does not have direct access** to protected resources.

All the monitor does is **check** that requests are valid.

"Does `dbarowy` **have access to** `file`**?"**

yes

Reference Monitor Design



OS: **big codebase**, **difficult** to verify correctness.

Monitor: **small codebase**, **easy** to verify correctness.

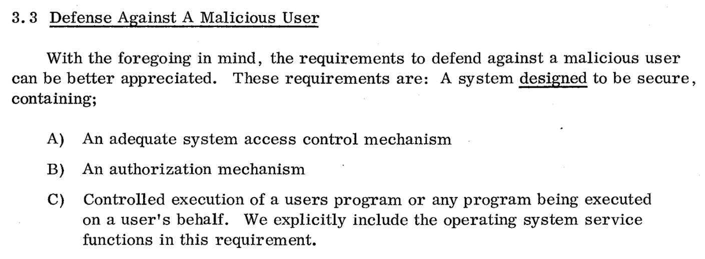
A reference monitor reduces the size of the **trusted compute base**.

Trusted Compute Base

A **trusted compute base** is the set of all **hardware** and **software** components such that **any bug** might jeopardize the enforcement of a given **security policy**.

Observe that this definition *depends on the given security policy*.

Reference Monitor



3.3 Defense Against A Malicious User

With the foregoing in mind, the requirements to defend against a malicious user can be better appreciated. These requirements are: A system designed to be secure, containing;

A) An adequate system access control mechanism

B) An authorization mechanism

C) Controlled execution of a users program or any program being executed on a user's behalf. We explicitly include the operating system service functions in this requirement.

With respect to unbounded resource access, a reference monitor removes even the operating system from the TCB.

How can we guarantee that a monitor “does the right thing”?

Information Flow

A formalism that deals with trust

Bell-LaPadula Model

Developed in 1973 by **David Bell** and **Leonard LaPadula** at MITRE for the Multics OS.

1. The **Simple Security Property** states that a subject at a given security level may not read an object at a higher security level.
2. The ***** (**star**) **Property** states that a subject at a given security level may not write to any object at a lower security level.
3. The **Discretionary Security Property** allows information to flow to lower levels (e.g., generals to soldiers).

These rules are a little vague.

Secure Information Flow Model

- Invented in 1975 by **Dorothy Denning**, then a PhD student at Purdue.
- "A Lattice Model of Secure Information Flow" (1976)
- A formal model that ensures that a computer will always "do the right thing" with respect to a security policy.
- A reference monitor can be proven secure provided that it faithfully (verifiably) implements the Denning model.

Secure Information Flow Model

- The Denning model is about **keeping secrets**.
- It **depends** on being able to reliably **authenticate** (i.e., "Identity" from CIAA) principals (subjects and objects).
- It **guarantees** that high-security information **cannot be leaked** to low-security principals.
- Is general enough to work in secure operating systems, secure compilers, military organizations, etc.

Mathematical Models

You have almost certainly seen a **mathematical model** (or "abstraction") used in CS before.

E.g., a **Turing machine**.

I like to think of **models** as **games**.

Like games, they tell you what the **rules** are.

Like games, we want to know whether we can "**win**" at some objective **while following the rules**.

Mathematical Models

The Denning model defines a **realistic** and **clear** **set of rules**, unlike the Bell-LaPadula model.

The Denning model is built on top of a **lattice**, which is a kind of **graph**.

Specifically, **vertices** denote classes of things, or **tags**, and **edges** denote how things can be accessed, or **flow relationships**.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **objects**.

A set of **principals**.

A set of **security tags**.

A set of **flow relations**.

A set of **join relations**.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **objects**.

These could be things like **files**, **memory locations**, etc.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **principals**.

These are **processing agents**, e.g., a computer processor, or a computer program, or people.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **security classes**.

These are **labels**, like “top secret,” “classified,” “sensitive,” and “public.”

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **flow relations**.

These are **functions** that **take two SCs** and return **true** or **false**; they say whether information **can flow** from one **SC** to another.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **join relations**.

These are **functions** that **take two SCs** and return an **SC**; they say **what security class is derived** by combining information from **SC**s.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **objects**.

A set of **principals**.

A set of **security classes**.

A set of **flow relations**.

A set of **join relations**.

Secure Information Flow Model

Helper function:

s(**x**) : **SC**

where **x** is either a **N** or an **P**.

In other words, the function **s** gives you the **security class** of an **object** or a **principal**.

Secure Information Flow Model

FM = <**N**, **P**, **SC**, {→}, {⊕}>

A set of **objects**.

A set of **principals**.

A set of **security tags**.

A set of **flow relations**.

A set of **join relations**.

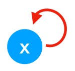
Secure Information Flow Model

→ means **can flow**

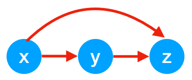
if **a** → **b** then data can flow from tag **a** to tag **b**

Flow relations are:

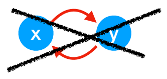
a) reflexive: **x** → **x**



b) transitive: **x** → **y** and **y** → **z** then **x** → **z**



c) anti-symmetric: **x** → **y** and **y** → **x** then **x = y**



Secure Information Flow Model

⊕ means **join**

a ⊕ **b** denotes the class obtained by combining **a** and **b**

i.e., what happens when you **staple things together**

Join relations are:

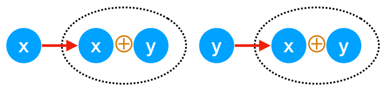
a) reflexive: **x** ⊕ **x** = **x**

b) commutative: **x** ⊕ **y** = **y** ⊕ **x**

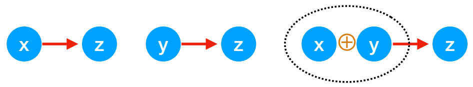
Secure Information Flow Model

The following relations are always true, to avoid absurdities:

a) $x \rightarrow (x \oplus y)$ and $y \rightarrow (x \oplus y)$



b) if $x \rightarrow z$ and $y \rightarrow z$ then $(x \oplus y) \rightarrow z$



Example

FM = <N, P, SC, {$\rightarrow$}, {$\oplus$}>

N = { petitions, holy scripture}	s(king) = kingly
P = { king, peon }	s(peon) = public
SC = { kingly, public }	s(petitions) = public
$\rightarrow$ = { public $\rightarrow$ kingly }	s(holy scripture) = kingly
$\oplus$ = { public $\oplus$ kingly = kingly }	

Can peons read petitions?

Can peons read scripture?

Can peons share petitions with the king?

Class Activity

FM = <N, P, SC, {$\rightarrow$}, {$\oplus$}>

N = { your diary, dinner plans, parents' diary}

P = { you, your little sister (YLS), parents }

SC = { ??? }

$\rightarrow$ = { ??? }

$\oplus$ = { ??? }

1. **Only parents** should be able to read **the parents' diary**.
2. **Only you** should be able to read **your diary**.
3. **Anyone** can read the **dinner plans**.

Fill in SC, $\rightarrow$, and $\oplus$ and provide tags s.

Class Activity

Your model should be able to answer these questions **mechanically**.

- Can you read your diary?
- Can you write about dinner in your diary?
- Can your parents copy dinner information from their diary into the dinner plans?
- What happens if a page from your diary and a page from your parents diary both just happen to fall out at the same time and stick together. Who can read those pages?

Practicality issues for access controls

Mandatory vs Discretionary Controls

Can parents tell the kids the dinner plans at all?

Not the way we formulated it.

Access controls are **discretionary** in the sense that a principal with a certain access permission is capable of performing that action **unless restrained** by a **mandatory access control**.

Implicit flow

Consider the following program.

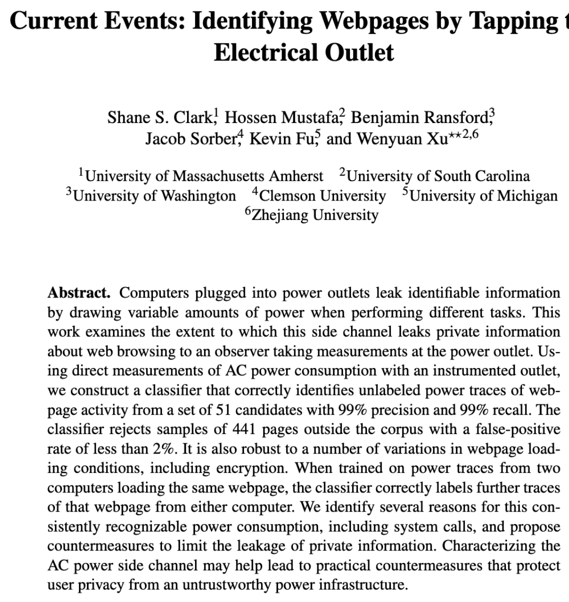
Suppose **s**(`l`) = public and **s**(`h`) = private.

```
int l;
bool h;
if (h) {
    l = 3
} else {
    l = 42
}
```

The value of `h` can be deduced because of a **side channel**.

Side Channel

A **side channel vulnerability** is any vulnerability that exists when **public information** observed during the correct operation of an implementation allows an attacker to **infer and exploit secret state**.



Current Events: Identifying Webpages by Tapping the Electrical Outlet

Shane S. Clark[1] Hossen Mustafa[2] Benjamin Ransford[3]
Jacob Sorber[4] Kevin Fu[5] and Wenyuan Xu**[2,6]

[1]University of Massachusetts Amherst [2]University of South Carolina
[3]University of Washington [4]Clemson University [5]University of Michigan
[6]Zhejiang University

Abstract. Computers plugged into power outlets leak identifiable information by drawing variable amounts of power when performing different tasks. This work examines the extent to which this side channel leaks private information about web browsing to an observer taking measurements at the power outlet. Using direct measurements of AC power consumption with an instrumented outlet, we construct a classifier that correctly identifies unlabeled power traces of webpage activity from a set of 51 candidates with 99% precision and 99% recall. The classifier rejects samples of 441 pages outside the corpus with a false-positive rate of less than 2%. It is also robust to a number of variations in webpage loading conditions, including encryption. When trained on power traces from two computers loading the same webpage, the classifier correctly labels further traces of that webpage from either computer. We identify several reasons for this consistently recognizable power consumption, including system calls, and propose countermeasures to limit the leakage of private information. Characterizing the AC power side channel may help lead to practical countermeasures that protect user privacy from an untrustworthy power infrastructure.

Side Channel Mitigation

Try to ensure that there is no relationship between observable and unobservable state.

input
program
output

An extremely difficult task, especially for programs that *do something*!

Side Channel Mitigation

Alternative: minimize relationship between observable and unobservable state.

```
int l;
bool h;
if (h) {
    l = 3
} else {
    l = 42
}
```

“How many bits of information about `h` does `l` leak?”

`l` leaks 1 bit; since `h` is a 1 bit variable, this is everything an attacker needs.

Further reading: **quantitative information flow**.

Recap & Next Class

Today we learned:

Reference monitors

Bell-LaPadula

Denning model

Next class:

How to give a good presentation

Short primer on networks