Course Website:

https://kmicinski.com/cis700-f22

CIS700

Language-Based Security

Instructor: Kris Micinski

- Main Course Objective
- Logistics, Course Delivery
- Syllabus
- Grading
- Exams
- Course FAQs

Course Objective

Give you background knowledge of formal principles from CS (programming languages, formal methods, discrete math, …) to understand how to think precisely about the security of programs

Grading

- 50% — Problem sets
  - Between 4 and 6 of these, weighted equally
  - Coq exercises, written homeworks
  - **Collaboration / pair-programming allowed**
- 10% — Prep / lead a paper discussion at some point through semester
- 40% — Final project, a 2-to-5-week project relevant to class

- Week 1 — Security Policies Intro; Coq Intro
- Week 2 — Role-Based Access Control; Induction on lists
- Week 3 — More Induction, Polymorhism; Safety Properties
- Week 4 — Constructive Logic Intro; Memory safety
- Week 5 — Inductive propositions; Memory exploitation primer
- Week 6 — Curry-Howard correspondence, access control formalization
- Week 7 — Capabilities, Authorization logic

- Week 8 — Noninterference and Information Flow
- Week 9 — Security Type Systems
- Week 10 — Product Programs / Relational Verification (RHTT, …)
- Week 11 — Control/data/…-flow integrity
- Week 12 — Side channels; Probabilistic information flow
- Week 13 — Presentations 1, assorted topics
- Week 14 — Presentations 2, course conclusion

Paper Reading

- We will read **eight papers** through the semester.
- I realize papers can be hard to read—even if you are finding the technical details frustrating to wade through, commit to reading ~1-2 hours at a time
- Plan to spend roughly 4 to 8 hours for each paper we read
- We will have homework problems drawn from the papers, which you will collectively answer together (by yourself/in groups)

Paper Presenting

- I will do the first one to give an idea of content
- Present at least…
  - Key ideas
  - Identify novel contributions
  - How were the ideas evaluated? Did they prove theorems, build a prototype, evaluate against a currently-existing tool?
  - What do you think this paper contributes to the future of the field?

- First paper: next Thursday
  - Looking Back at the Bell-La Padula Model

Syllabus

Most up-to-date syllabus always available at:

https://kmicinski.com/cis700-f22/syllabus

Principals and Access Control Intro

CIS700 — Fall 2022

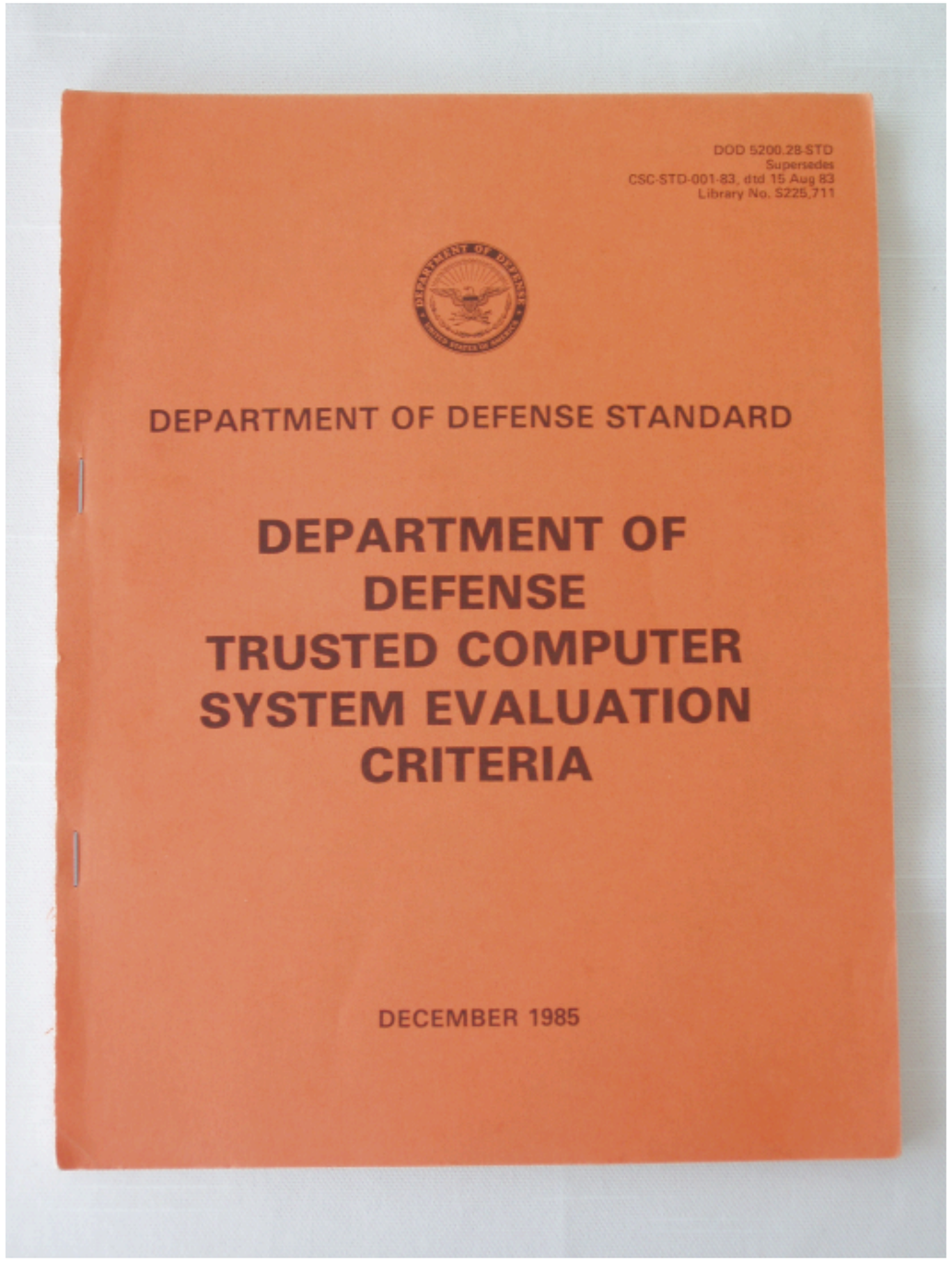
Kris Micinski

Early History

- We had computers; they were huge
- No security, but who owned all of them? The government
- Computers mostly run batch jobs to completion
- Only trusted users used computers at first
- Next: spent taxpayer $$ on computers; let's let the *public* use them—but <u>how</u> when we don't trust them?

Trusted Computer System Evaluation Criteria (TCSEC)

- Published by DOD 1983 (updated 1985)
  - "Rainbow series" published by DOD throughout 80s
- **Mandatory** security policy (top-down, users have levels, …)
- **Marking** data must be annotated with its security level
- **Discretionary** policy: need-to-know individuals authorized
- **Accountability** via identification, authentication,& auditing
- **Assurance** emphasis placed upon correctness of system


DOD 5200.28-STD
Supersedes
CSC-STD-001-83, dtd 15 Aug 83
Library No. S225,711
DEPARTMENT OF DEFENSE STANDARD
DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA
DECEMBER 1985


DEPARTMENT OF DEFENSE STANDARD
DEPARTMENT OF DEFENSE TRUSTED COMPUTER SYSTEM EVALUATION CRITERIA
DEPARTMENT OF DEFENSE
PASSWORD MANAGEMENT
GUIDELINE
COMPUTER SECURITY REQUIREMENTS
NATIONAL COMPUTER SECURITY CENTER
A GUIDE TO UNDERSTANDING AUDIT IN TRUSTED SYSTEMS
Guidelines for Formal Verification Systems

Rainbow books were prophetic

- Surprisingly, we see few systems that truly embody TCSEC *to this day*
- Linux / Windows / … support forms of discretionary access control (DAC)
- SELinux (NSA) supports MFA
- Even in early 80s, TCSEC goals still in the space of research and high-end industry
  - Almost no systems truly verified in practice

Principals

- Principals is often used vaguely, refers to a set of **users** in a system who take various actions
- Set of principals defined on a case-by-case basis
  - Owner of secret key?
  - Running process?
  - Alice / Bob / …
- Generally, we talk about principals because they *own or have control over* some object in a system
- Sometimes, principals may “act for” others
  - E.g., manager may act for employee (but not reverse)

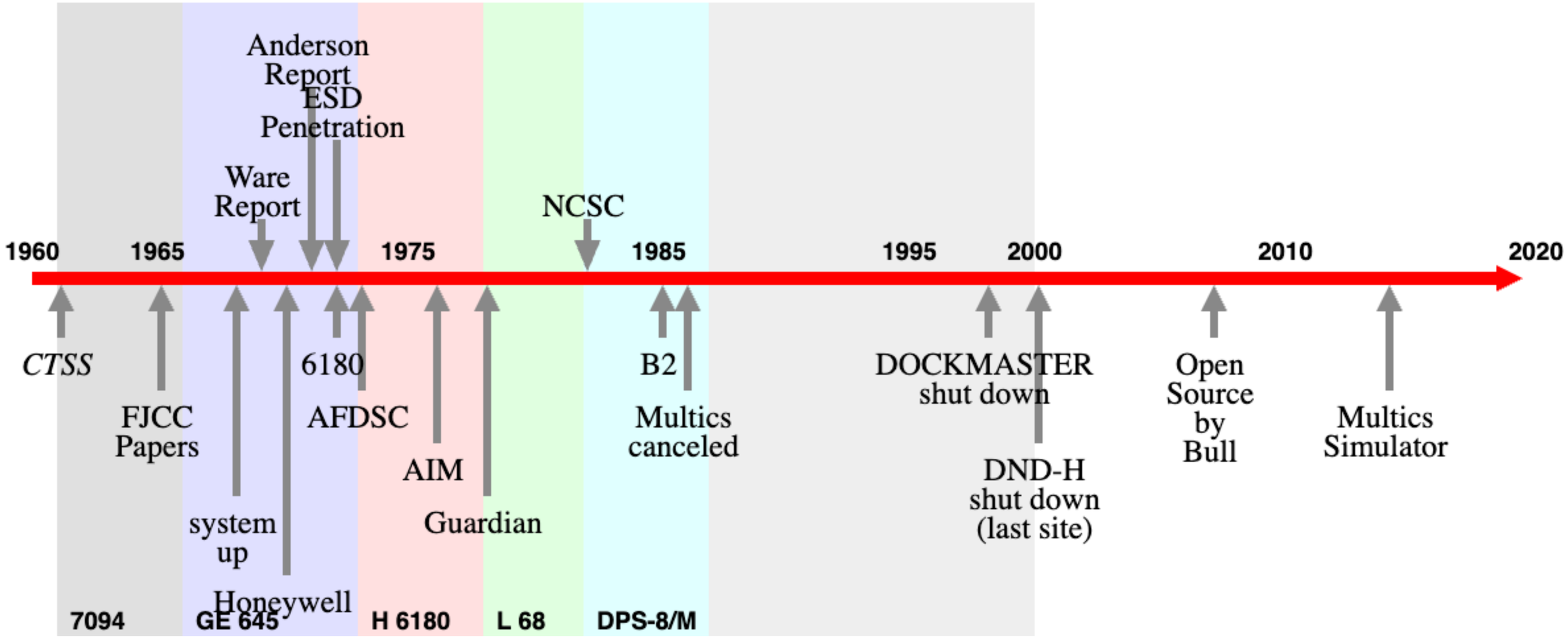
Alice and Bob talk to each other over unsecured HTTP; Eve eavesdrops

Who are the principals?

Access Control

- https://www.cs.cornell.edu/fbs/publications/chptr.MAC.pdf
- “Who can access / read / write / execute / … this file /…”
  - In UNIX, everything a file—terminals, directories, etc…
- Mandatory vs. discretionary
  - Discretionary access control (user controls label)
  - Mandatory: top-down security policy managed by OS

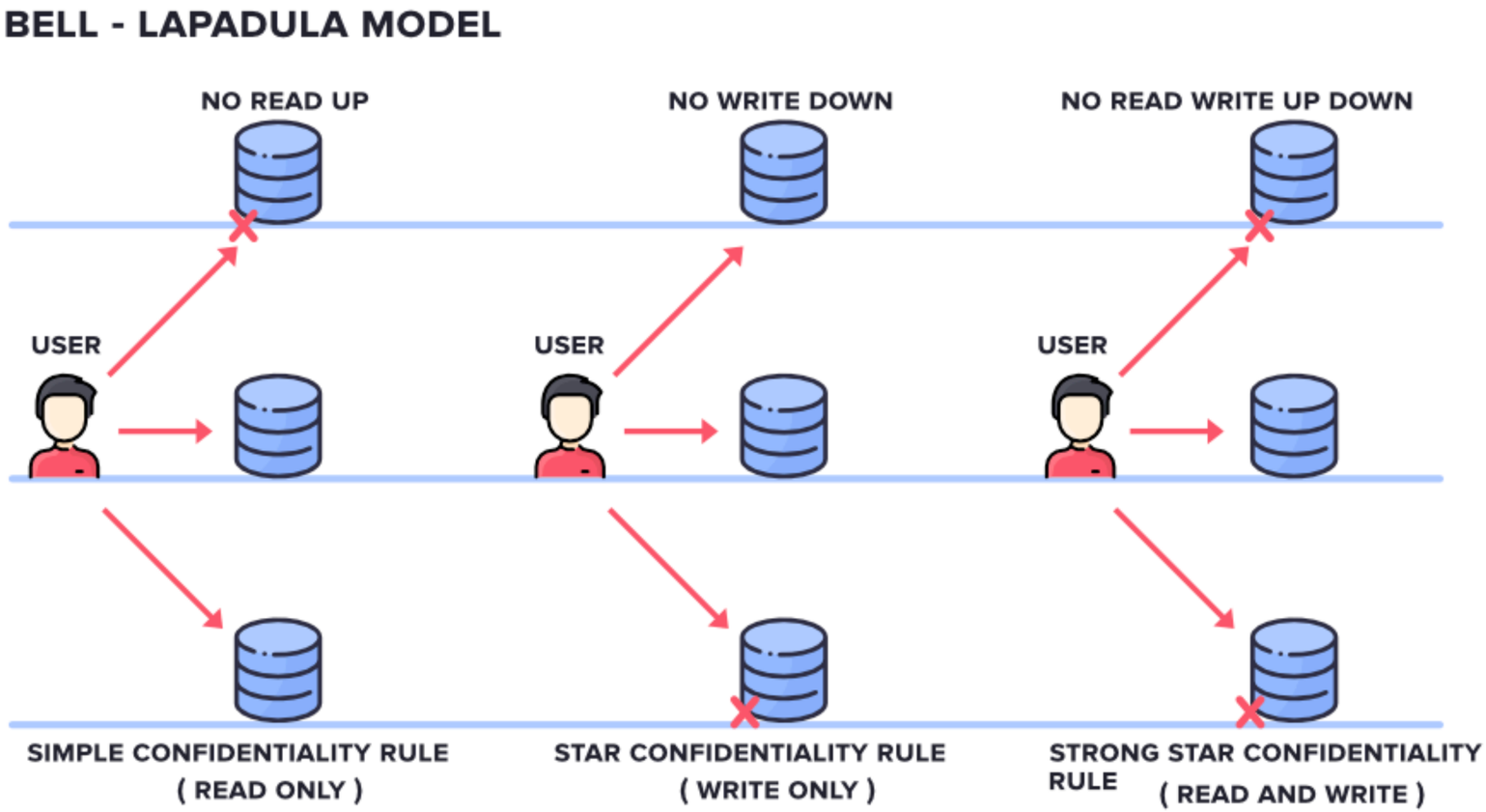
Multics History



Multics eventually met B2

- Precursor to UNIX, developed proprietarily
- Project Guardian—Honeywell+USAFRL produce auditable Multics (1975-80s)
- B2 is a US NCSC security classification that "indicates support for mandatory access control as well as a relatively high level of security assurance."
  - This requires accounting for **covert** channels (e.g., timing, power, …)
  - Implemented **access isolation** mechanism
- https://multicians.org/b2.html

Access Control



"If the objective of the Orange Book and the NCSC was to create a rich supply of high-assurance systems that incorporated mandatory security controls, it is hard to find that the result was anything but failure. ... If the objective of the Orange Book and NCSC was to raise the bar by motivating vendors to include security controls in their products, the case for success is stronger." — Steve Lipner

Direct Flows

```
fin = open(“file.txt”) // owned by Alice
text = read(fin)
fout = open(“file2.txt”)
write(fout,text)
```

Is this program bad? *It depends*

- Who *runs* the program? If it’s Alice, maybe it’s ok
  - But, maybe Alice is not permitted to write to disc (her share up)
- If Bob runs the program, is the new file owned by him?
  - If so, this is a **direct flow**
  - But maybe he is authorized to xfer data for Alice but can’t read

Direct Flows

```
fin = open(“file.txt”) // owned by Alice
text = read(fin)
fout = open(“file2.txt”)
write(fout,text)
```

On modern (POSIX) systems, “file.txt” has (a) owner and (b) read/write/execute permissions (bitwise: in that order—110 is read/write/noexecute), along with all its parent (recursively) directories. Directory listing guarded by “execute” permission

Permissions specified for **owner**, **group**, and **world**

```
chmod 777 file.txt # Everyone read/write/execute
chmod 760 foo/file.txt # Owner rwx, group rw, world none
```

```
kmicinski % mkdir foo
kmicinski % touch foo/bar
kmicinski % chmod 000 foo
kmicinski % ls foo
ls: foo: Permission denied
kmicinski % rm -rf foo
rm: foo: Permission denied
kmicinski % sudo rm -rf foo
```

Indirect Flows

```
fin = open(“file.txt”) // owned by Alice
text = read(fin)
fout = open(“file2.txt”)
if (text == “0”):
  write(fout,”1”)
else:
  write(fout,”0”)
```

Also leaks information—tracking direct flows is not enough! Thus, we always track the influence of the **program counter** (i.e., what I have branched on) at every point

Because `text` is Alice’s data, once we branch on it any writes become implicitly tainted by knowledge of (some property of) the data

Confidentiality

```
fin = open(“file.txt”) // owned by Alice
text = read(fin)
fout = open(“file2.txt”)
write(fout,text)
```

Direct flows represent the most basic kind of safety policy we will discuss in this class: **confidentiality**

Confidentiality says, **“data at some security level may not be observable to anyone below that level.”**

In other words: a (low-security) user may not **read up**

Integrity

```
fout = open(“file.txt”) // owned by Alice
write(fout,”Setec Astronomy”) // write corrupt’s Alice’s file!
```

If Bob executes this program, he will corrupt Alice’s data! This can be just as bad as reading data he shouldn’t be able to!

This is a violation of **integrity**, which occurs when a low-security user influences high-security data

Bell-LaPadula: “no read up, no write down.” (Strong star: you may only write at your level.)

Summary

Confidentiality — keep data secret (safety property)

Integrity — keep stuff free from public influence (liveness property)

No read up, no write down (Bell LaPadula)

Principals, labels (labels form order)