CS 591: Introduction to Computer Security

Lecture 2: Access Control

James Hook

10/20/07 14:32

Objectives

- Introduce the mechanism of Access Control
- Relate mechanism to Confidentiality, Integrity and Availability
- Introduce the Access Control Matrix Model and Protection State Transitions

10/20/07 14:32

Alice and Bob

- Standard names for "agents" in a security or crypto scenario
- Also known as "A" and "B"

10/20/07 14:32

An Access Control Scenario

- Alice:
  1. New Secret foo
- Bob:
  2. If (cp foo afoo)
  3. then echo "success"
  4. else echo "fail"

Intent:

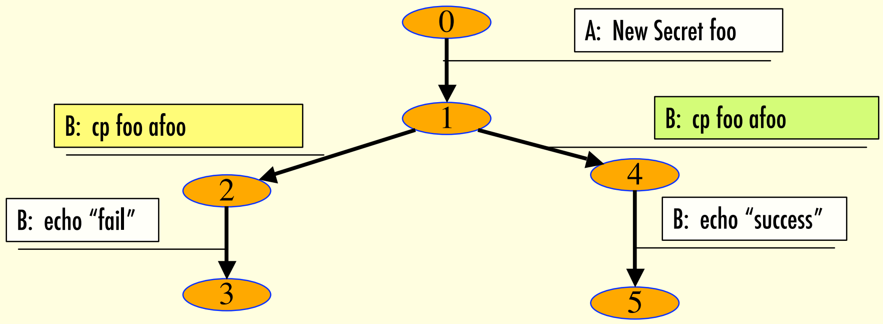
•Bob's **cp** is attempting to violate Alice's expected access policy

•If **cp** succeeds then the principle of **confidentiality** is not satisfied

10/20/07 14:32

Q: Revise scenario to violate **availability**

Characterizing the Violation



Basic Abstraction: States and Transitions

Q: What are the States?

Q: What determines if we reach State 2 or 4 from State 1?

10/20/07 14:32

Q: If we reach State 5 was State 1 good?

Secure and non-Secure States

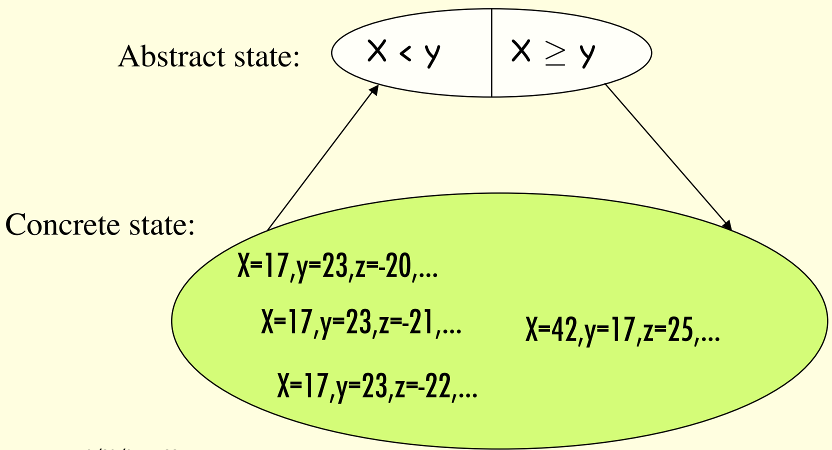
Characterize states in a system as "Secure" and "non-Secure"

A system is **Secure** if every transition maps Secure states to Secure states

Consequence: In the scenario, security is compromised if Alice's "New secret foo" yields a state in which Bob can access foo.

10/20/07 14:32

Abstraction



10/20/07 14:32

Protection States

An abstraction that focuses on security properties

Primarily interested in characterizing Safe states

Goal is to prove that all operations in the system preserve "security" of the protection state

Access Control Matrix is our first Protection State model

10/20/07 14:32

Access Control Matrix Model

- Lampson '71, refined by Graham and Denning ('71, '72)
- Concepts
  - **Objects**, the protected entities, O
  - **Subjects**, the active entities acting on the objects, S
  - **Rights**, the controlled operations subjects can perform on objects, R
  - **Access Control Matrix**, A, maps Objects and Subjects to sets of Rights
  - State: (S, O, A)

10/20/07 14:32

Confidentiality Scenario

Initial State

Subjects $S_0 = \{A,B\}$

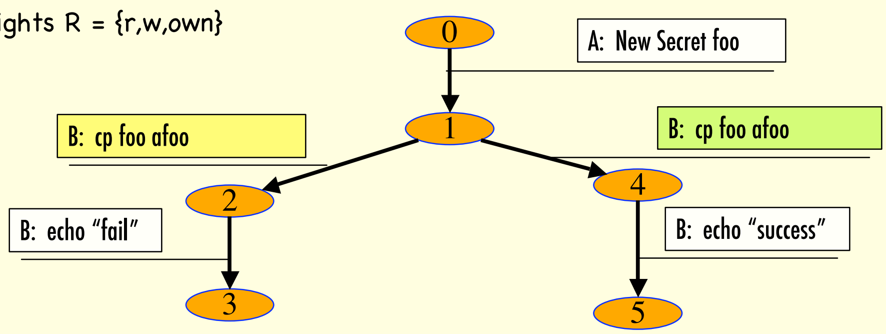
Objects $O_0 = \{\}$

AC Matrix $A_0 = \{\}$

Rights R = {r,w,own}

$(S_0, O_0, A_0) \vdash_{\text{A: New Secret foo}} (S_1, O_1, A_1)$

$(S_1, O_1, A_1) \vdash_{\text{B: cp foo afoo}} (S_1, O_1, A_1)$



Intended State 1

Subjects $S_1 = \{A,B\}$

Objects $O_1 = \{foo\}$

AC Matrix $A_1 = \{(A,foo,[r,w,own]), (B,foo,[])\}$

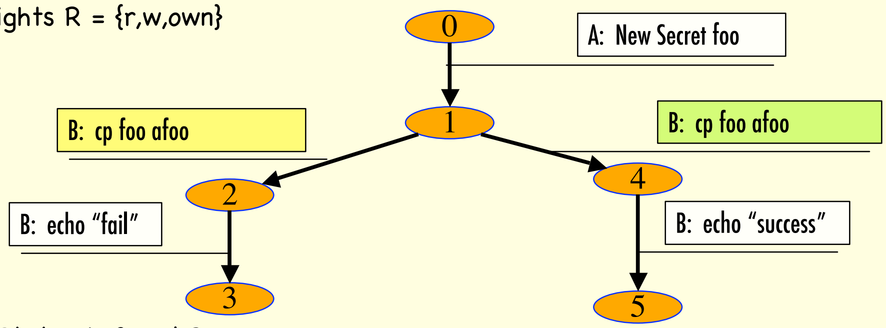
10/20/07 14:32

Confidentiality Scenario

Initial State
- Subjects $S_0 = \{A,B\}$
- Objects $O_0 = \{\}$
- AC Matrix $A_0 = \{\}$

Rights R = {r,w,own}

Is there a representation for Protection States 4 and 5?



States 1, 2 and 3
- Subjects $S_1 = \{A,B\}$
- Objects $O_1 = \{foo\}$
- AC Matrix $A_1 = \{(A,foo,[r,w,own]), (B,foo,[])\}$

Critical issue is definition of $\vdash_{cp}$ … …

10/20/07 14:32

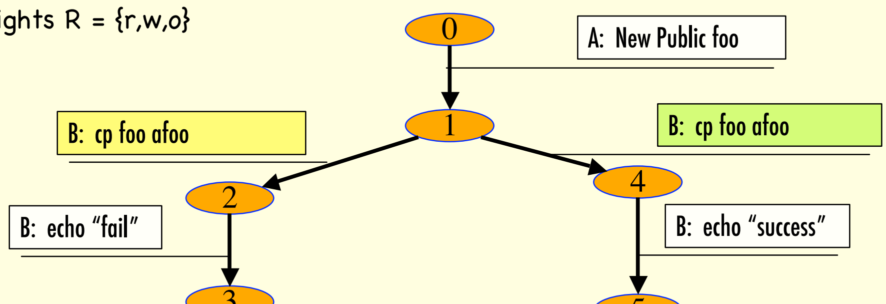
Availability Scenario

Initial State
- Subjects $S_0 = \{A,B\}$
- Objects $O_0 = \{\}$
- AC Matrix $A_0 = \{\}$

Rights R = {r,w,o}

$(S_0, O_0, A_0) \vdash_{A:\ New\ Public\ foo} (S_1, O_1, A_1)$

$(S_1, O_1, A_1) \vdash_{B:\ cp\ foo\ afoo} (S_4, O_4, A_4)$



State 1
- Subjects $S_1 = \{A,B\}$
- Objects $O_1 = \{foo\}$
- AC Matrix $A_1 = \{(A,foo,[r,w,o]), (B,foo,[r])\}$

State 4
- Subjects $S_4 = S_1$
- Objects $O_4 = O_1 \cup \{afoo\}$
- AC Matrix $A_4 = \{(A,foo,[r,w,o]), (B,foo,[r]), (A,afoo,[]), (B,afoo,[r,w,o])\}$

10/20/07 14:32

Voting Machine

- How can a voting machine be modeled with subjects, objects, and rights?
- In what ways do the rights change dynamically?

A Domain-Specific Language for Access Control

- Harrison, Ruzzo, and Ullman defined a set of primitive commands
  - Create subject s
  - Create object o
  - Enter r into a[s,o]
  - Delete r from a[s,o]
  - Destroy subject s
  - Destroy object o
- We will use this DSL of primitive commands to model the system in our example

Heads up:
We have 2 languages: HRU
primitives and
the example!

HRU Semantics

$(S, O, A) \vdash_{\text{Create subject } s} (S \cup \{s\}, O, A)$

$(S, O, A) \vdash_{\text{Create object } o} (S, O \cup \{o\}, A)$

$(S, O, A) \vdash_{\text{Enter } r \text{ into } a[s,o]} (S, O, A')$
where $A'[s,o] = A[s,o] \cup \{r\}$

$(S, O, A) \vdash_{\text{Delete } r \text{ from } a[s,o]} (S, O, A')$
where $A'[s,o] = A[s,o] - \{r\}$

$(S, O, A) \vdash_{\text{Destroy subject } s} (S - \{s\}, O, A\lfloor)$

$(S, O, A) \vdash_{\text{Destroy object } o} (S, O - \{o\}, A\lfloor)$
where $A\lfloor$ is the appropriate restriction of A

10/20/07 14:32

Molecules from Atoms

- This DSL gives us atomic transitions
- To model a system we combine these atomic operations into commands
- A system model in this framework is the set of commands that implement the system primitives

10/20/07 14:32

Modeling the Example

- Interface
  - X: New Secret <f>
  - X: New Public <f>
  - X: Cp <f> <f>
  - X: If <command> then <command> else <command>
- Assumptions
  - X ranges over {A,B}

10/20/07 14:32

Example

```
Initialize ()
   create subject A
   create subject B
end
New.Secret (x,f)
   create object f
   enter own into a[x,f]
   enter r into a[x,f]
   enter w into a[x,f]
end
```

```
New.Public (x,f)
   create object f
   enter own into a[x,f]
   enter r into a[A,f]
   enter r into a[B,f]
   enter w into a[x,f]
End
```

10/20/07 14:32

Example (cont)

```
Cp(x,src,dest)
   if r ∈ a[x,src]
   then
      create object dest
      enter own into a[x,dest]
      enter w into a[x,dest]
      ?
End
```

Conditional command

Modeling helps us be precise: Is the new file "public" or "secret"?

10/20/07 14:32

Modeling if

- How do we model the **if** statement in our scenario?
- We assumed Unix like "exit status"
- Could enrich model to have statements have value
- Does that add value?

10/20/07 14:32

Modeling if (cont)

- To establish system security we must model all sequences of commands
- What matters is that **cp** won't reveal Alice's secret
- Since we are considering **all** sequences of non-conditional commands we don't need to model

  If c1 then c2 else c3

  since we model both

  c1; c2

  c1; c3
- Why doesn't this argument apply to primitive commands?

10/20/07 14:32

Conditional Commands

- To obtain results in Chapter 3 we place technical restrictions on HRU conditional commands
- Condition must be "positive"
  - $r \in a[s,o]$
  - Cf. negative: $r \notin a[s,o]$
- Conjunctions of conditions are allowed
  - $r \in a[s,o] \wedge r' \in a[s',o']$
- Disjunctions are unnecessary
  - All atomic actions are idempotent
  - if $\phi \vee \psi$ then C $\equiv$ if ϕ then C; if ψ then C

10/20/07 14:32

Access Control Matrix

- Very high fidelity model
- Every user and process can be modeled as a subject
- Every file and process can be modeled as an object
- Does it scale?
- Is it useful?

10/20/07 14:32

Access Control Matrix

- The access control matrix model is a critical reference point
  - most systems can be modeled within the framework
  - most mechanisms are an imperfect approximation of the Access Control Matrix

10/20/07 14:32

Foundational Results

- Can we use an algorithm to test if a system is secure?
  - What do we mean by "system"?
  - What do we mean by "secure"?

10/20/07 14:32

Aside: Safety and Liveness

- Safety property: A bad thing does not happen
  - E.g. A memory safe program will not dereference a "bad" pointer
- Liveness property: A good thing will happen eventually
  - E.g. Every runnable process will eventually be scheduled

10/20/07 14:32

Security: safe or live?

- Availability is often a liveness property
- Confidentiality is often cast as a safety property
- Integrity can be both
  - The processor will execute the instruction stream is a liveness property
  - All memory will be accessed consistent with the protection state is a safety property

10/20/07 14:32

Bounding the Problem

- "Mono-operational" commands
  - If each system level command in the modeled system is implemented by a single HRU primitive the system is "mono-operational"
- General case
  - In the general case the commands of the system being modeled are implemented by arbitrary combinations of HRU primitives
- Cast Problem as Safety Property
  - *Bad things don't happen*

10/20/07 14:32

What is secure?

- Must designate a "bad thing" and then prove it doesn't happen
- Definition: A right *r* is leaked if it is added to an element of the access control matrix that does not already contain it
  - In our example "new secret foo" leaks rights "own, r and w" if foo did not already exist
- Definition: A system is safe with respect to right *r* if it does not leak the right *r*

10/20/07 14:32

Follow Bishop

If time permits in this lecture jump to Bishop's slide #03-04

10/20/07 14:32

Conclusion

- Modeling is the process of abstracting to the essence of the property of concern
- Security Modeling exploits "protection state" abstractions
- Access Control Matrix is a "best" model for file and process granularity modeling
- With virtually any realistic system the general security question will be undecidable

10/20/07 14:32

Looking Forward

- Next Week
  - Jim Binkley will lecture on Crypto
  - Bishop: 8, 9, 10
  - Anderson: 2, 5
- Following Week
  - Bishop: [1, 2, 3,] 4, 5, 7
  - Anderson: [1,] 7

10/20/07 14:32

Backup Materials

10/20/07 14:32

A scenario from the text

- Bishop models a language with interface:
  - Create.file(p,f)
  - Spawn.process(p,q)
  - Make.owner(p,f)
  - Grant.read.file.1(p,f,q)
  - Grant.read.file.2(p,f,q)
  - Grant.write.file.1(p,f,q)
  - Grant.write.file.2(p,f,q)
- Some of his examples follow

10/20/07 14:32

Commands

```
Command create.file (p,f)
  create object f;
  enter own into a[p,f];
  enter r into a[p,f];
  enter w into a[p,f];
end
```

10/20/07 14:32

Commands (cont)

```
Command spawn.process(p,q)
  create subject q;
  enter own into a[p,q];
  enter r into a[p,q];
  enter w into a[p,q];
  enter r into a[q,p];
  enter w into a[q,p];
End
```

10/20/07 14:32

Conditional Commands

```
Command grant.read.file.1(p,f,q)
  if own in a[p,f]
  then
      enter r into a[q,f]
End
```

10/20/07 14:32

Root Agent

Create subjects voter, tallyAgent, reporter

Create objects vote, state, tally, voterCard

Initialize tally=0

Enter

10/20/07 14:32

Voter Agent

```
Repeat Indefinitely:
  Present credential;
  If credential accepted then
    Prepare ballot;
    Confirm vote;
  Withdraw credential
```

10/20/07 14:32

Tally Agent

```
While (mode = election) do
  On credential presented do
    If credential valid then
        Enable voting;
        On vote commit do atomic
            add vote to tally
            invalidate credential
```

10/20/07 14:32