

# Access Control

Lecture 9 – CS442 – Spring 2008

Vinod Ganapathy

Lecture notes based upon:

- Chapters 2, 3 and 4 in your textbook.

# Plan for today

- Quick summary of OpenPGP
- Access Control Matrix (Chapter 2)
- Overview of formal results (Chapter 3)
- Access Control Policies (Chapter 4)

# PGP Chains

- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (not present in version 3)
  - Public key algorithm, associated parameters
  - Public key

# OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer's key identifier (identifies key to encipher hash)
  - Public key algorithm (used to encipher hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (enciphered hash)
- Version 4 packet more complex

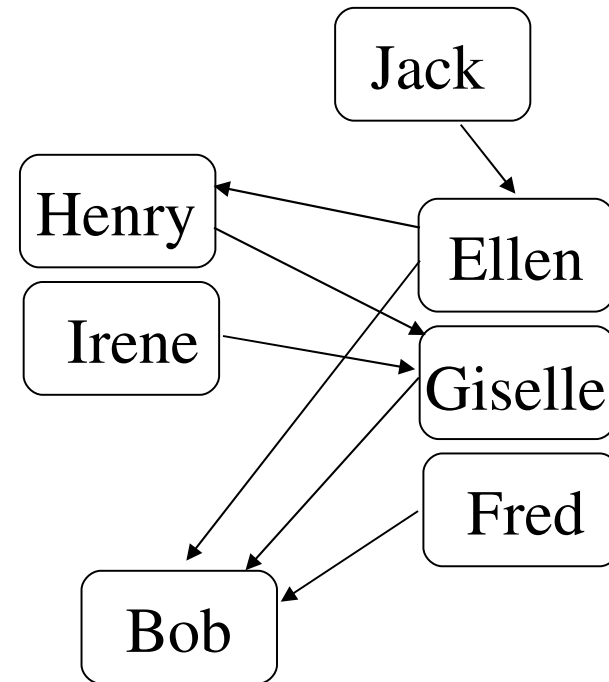
# Signing

- Single certificate may have multiple signatures
- Notion of “trust” embedded in each signature
  - Range from “untrusted” to “ultimate trust”
  - Signer defines meaning of trust level (no standards!)
- All version 4 keys signed by subject
  - Called “self-signing”

# Validating Certificates

- Alice needs to validate Bob's OpenPGP cert
  - Does not know Fred, Giselle, or Ellen
- Alice gets Giselle's cert
  - Knows Henry slightly, but his signature is at "casual" level of trust
- Alice gets Ellen's cert
  - Knows Jack, so uses his cert to validate Ellen's, then hers to validate Bob's

Arrows show signatures  
Self signatures not shown



# Access Control: Overview

- Protection state of system
  - Describes current settings, values of system relevant to protection
- Access control matrix
  - Describes protection state precisely
  - Matrix describing rights of subjects
  - State transitions change elements of matrix

# Description

objects (entities)

	$o_1$	...	$o_m$	$s_1$	...	$s_n$
$s_1$						
$s_2$						
...						
$s_n$						

subjects

- Subjects  $S = \{ s_1, \dots, s_n \}$
- Objects  $O = \{ o_1, \dots, o_m \}$
- Rights  $R = \{ r_1, \dots, r_k \}$
- Entries  $A[s_i, o_j] \subseteq R$
- $A[s_i, o_j] = \{ r_x, \dots, r_y \}$   
means subject  $s_i$  has  
rights  $r_x, \dots, r_y$  over  
object  $o_j$

# Example 1

- Processes  $p, q$
- Files  $f, g$
- Rights  $r, w, x, a, o$

	$f$	$g$	$p$	$q$
$p$	$rwo$	$r$	$rwXO$	$w$
$q$	$a$	$ro$	$r$	$rwXO$

# Example 2

- Procedures *inc\_ctr*, *dec\_ctr*, *manage*
- Variable *counter*
- Rights *+*, *-*, *call*

*counter inc\_ctr dec\_ctr manage*

*inc\_ctr*

*+*

*dec\_ctr*

*-*

*manage*

*call*

*call*

*call*

	<i>call</i>	<i>call</i>	<i>call</i>

# State Transitions

- Change the protection state of system
- $\vdash$  represents transition
  - $X_i \vdash_{\tau} X_{i+1}$ : command  $\tau$  moves system from state  $X_i$  to  $X_{i+1}$
  - $X_i \vdash^* X_{i+1}$ : a sequence of commands moves system from state  $X_i$  to  $X_{i+1}$
- Commands often called *transformation procedures*

# Primitive Operations

- **create subject  $s$ ; create object  $o$** 
  - Creates new row, column in ACM; creates new column in ACM
- **destroy subject  $s$ ; destroy object  $o$** 
  - Deletes row, column from ACM; deletes column from ACM
- **enter  $r$  into  $A[s, o]$** 
  - Adds  $r$  rights for subject  $s$  over object  $o$
- **delete  $r$  from  $A[s, o]$** 
  - Removes  $r$  rights from subject  $s$  over object  $o$

# Creating File

- Process  $p$  creates file  $f$  with  $r$  and  $w$  permission

```
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
```

# Mono-Operational Commands

- Make process  $p$  the owner of file  $g$   
command *make.owner(p, g)*  
    enter *own* into  $A[p, g]$ ;  
end
- Mono-operational command
  - Single primitive operation in this command

# Conditional Commands

- Let  $p$  give  $q$   $r$  rights over  $f$ , if  $p$  owns  $f$   
command `grant.read.file.1(p, f, q)`  
    if `own in A[p, f]`  
    then  
        enter  $r$  into `A[q, f]`;  
    end
- Mono-conditional command
  - Single condition in this command

# Multiple Conditions

- Let  $p$  give  $q$   $r$  and  $w$  rights over  $f$ , if  $p$  owns  $f$  and  $p$  has  $c$  rights over  $q$

```
command grant.read.file.2( $p, f, q$ )  
  if  $own$  in  $A[p, f]$  and  $c$  in  $A[p, q]$   
  then  
    enter  $r$  into  $A[q, f]$ ;  
    enter  $w$  into  $A[q, f]$ ;  
end
```

# Key Points

- Access control matrix simplest abstraction mechanism for representing protection state
- Transitions alter protection state
- 6 primitive operations alter matrix
  - Transitions can be expressed as commands composed of these operations and, possibly, conditions

# What Is “Secure”?

- Adding a generic right  $r$  where there was not one is “leaking”
- If a system  $S$ , beginning in initial state  $s_0$ , cannot leak right  $r$ , it is *safe with respect to the right  $r$* .

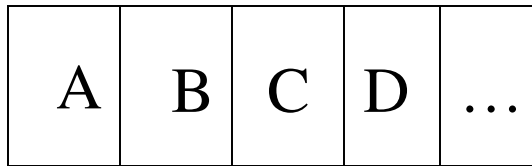
# Safety Question

- Does there exist an algorithm for determining whether a protection system  $S$  with initial state  $s_0$  is safe with respect to a generic right  $r$ ?
  - Here, “safe” = “secure” for an abstract model
- Answer: **No**. Seminal result due to Harrison, Ruzzo and Ullman (1976).

# General Case

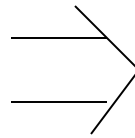
- Answer: *no*
- Sketch of proof:
  - Reduce halting problem to safety problem
  - Turing Machine review:
    - Infinite tape in one direction
    - States  $K$ , symbols  $M$ ; distinguished blank  $b$
    - Transition function  $\delta(k, m) = (k', m', L)$  means in state  $k$ , symbol  $m$  on tape location replaced by symbol  $m'$ , head moves to left one square, and enters state  $k'$
    - Halting state is  $q_f$ ; TM halts when it enters this state

# Mapping



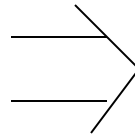
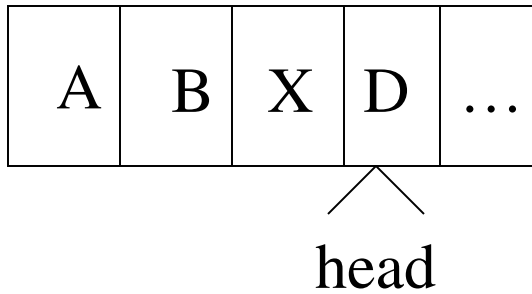
head

Current state is  $k$



	$s_1$	$s_2$	$s_3$	$s_4$	
$s_1$	A	<i>own</i>			
$s_2$		B	<i>own</i>		
$s_3$			C $k$	<i>own</i>	
$s_4$				D end	

# Mapping



	$s_1$	$s_2$	$s_3$	$s_4$	
$s_1$	A	<i>own</i>			
$s_2$		B	<i>own</i>		
$s_3$			X	<i>own</i>	
$s_4$				D $k_1$ end	

After  $\delta(k, C) = (k_1, X, R)$   
where  $k$  is the current  
state and  $k_1$  the next state

# Command Mapping

$\delta(k, C) = (k_1, X, R)$  at intermediate becomes

command  $c_{k,C}(s_3, s_4)$

if  $own$  in  $A[s_3, s_4]$  and  $k$  in  $A[s_3, s_3]$   
and  $C$  in  $A[s_3, s_3]$

then

delete  $k$  from  $A[s_3, s_3]$ ;

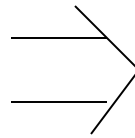
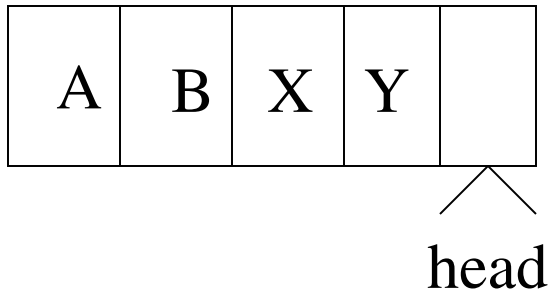
delete  $C$  from  $A[s_3, s_3]$ ;

enter  $X$  into  $A[s_3, s_3]$ ;

enter  $k_1$  into  $A[s_4, s_4]$ ;

end

# Mapping



	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$
$s_1$	A	<i>own</i>			
$s_2$		B	<i>own</i>		
$s_3$			X	<i>own</i>	
$s_4$				Y	<i>own</i>
$s_5$					<i>b k_2 end</i>

After  $\delta(k_1, D) = (k_2, Y, R)$   
where  $k_1$  is the current  
state and  $k_2$  the next state

# Command Mapping

$\delta(k_1, D) = (k_2, Y, R)$  at end becomes  
command  $\text{crighthmost}_{k,c}(s_4, s_5)$   
if *end* in  $A[s_4, s_4]$  and  $k_1$  in  $A[s_4, s_4]$   
    and  $D$  in  $A[s_4, s_4]$   
then  
    delete *end* from  $A[s_4, s_4]$ ;  
    create subject  $s_5$ ;  
    enter *own* into  $A[s_4, s_5]$ ;  
    enter *end* into  $A[s_5, s_5]$ ;  
    delete  $k_1$  from  $A[s_4, s_4]$ ;  
    delete  $D$  from  $A[s_4, s_4]$ ;  
    enter  $Y$  into  $A[s_4, s_4]$ ;  
    enter  $k_2$  into  $A[s_5, s_5]$ ;  
end

# Rest of Proof

- Protection system exactly simulates a TM
  - Exactly 1 *end* right in ACM
  - 1 right in entries corresponds to state
  - Thus, at most 1 applicable command
- If TM enters state  $q_f$ , then right has leaked
- If safety question decidable, then represent TM as above and determine if  $q_f$  leaks
  - Implies halting problem decidable
- Conclusion: safety question undecidable

# Chapter 4

- Overview
- Policies
- Trust
- Nature of Security Mechanisms
- Example Policy

# Security Policy

- Policy partitions system states into:
  - Authorized (secure)
    - These are states the system can enter
  - Unauthorized (nonsecure)
    - If the system enters any of these states, it's a security violation
- Secure system
  - Starts in authorized state
  - Never enters unauthorized state

# Confidentiality

- $X$  set of entities,  $I$  information
- $I$  has *confidentiality* property with respect to  $X$  if no  $x \in X$  can obtain information from  $I$
- $I$  can be disclosed to others
- Example:
  - $X$  set of students
  - $I$  final exam answer key
  - $I$  is confidential with respect to  $X$  if students cannot obtain final exam answer key

# Integrity

- $X$  set of entities,  $I$  information
- $I$  has *integrity* property with respect to  $X$  if all  $x \in X$  trust information in  $I$
- Types of integrity:
  - trust  $I$ , its conveyance and protection (data integrity)
  - $I$  information about origin of something or an identity (origin integrity, authentication)
  - $I$  resource: means resource functions as it should (assurance)

# Availability

- $X$  set of entities,  $I$  resource
- $I$  has *availability* property with respect to  $X$  if all  $x \in X$  can access  $I$
- Types of availability:
  - traditional:  $x$  gets access or not
  - quality of service: promised a level of access (for example, a specific level of bandwidth) and not meet it, even though some access is achieved

# Policy Models

- Abstract description of a policy or class of policies
- Focus on points of interest in policies
  - Security levels in multilevel security models
  - Separation of duty in Clark-Wilson model
  - Conflict of interest in Chinese Wall model

# Types of Security Policies

- Military (governmental) security policy
  - Policy primarily protecting confidentiality
- Commercial security policy
  - Policy primarily protecting integrity
- Confidentiality policy
  - Policy protecting only confidentiality
- Integrity policy
  - Policy protecting only integrity

# Integrity and Transactions

- Begin in consistent state
  - “Consistent” defined by specification
- Perform series of actions (*transaction*)
  - Actions cannot be interrupted
  - If actions complete, system in consistent state
  - If actions do not complete, system reverts to beginning (consistent) state

# Trust

Administrator installs patch

2. Trusts patch came from vendor, not tampered with in transit
3. Trusts vendor tested patch thoroughly
4. Trusts vendor's test environment corresponds to local environment
5. Trusts patch is installed correctly

# Trust in Formal Verification

- Gives formal mathematical proof that given input  $i$ , program  $P$  produces output  $o$  as specified
- Suppose a security-related program  $S$  formally verified to work with operating system  $O$
- What are the assumptions?

# Trust in Formal Methods

1. Proof has no errors
  - Bugs in automated theorem provers
2. Preconditions hold in environment in which  $S$  is to be used
3.  $S$  transformed into executable  $S'$  whose actions follow source code
  - Compiler bugs, linker/loader/library problems
4. Hardware executes  $S'$  as intended
  - Hardware bugs (Pentium £00£ bug, for example)

# Types of Access Control

- Discretionary Access Control (DAC, IBAC)
  - individual user sets access control mechanism to allow or deny access to an object
- Mandatory Access Control (MAC)
  - system mechanism controls access to object, and individual cannot alter that access
- Originator Controlled Access Control (ORCON)
  - originator (creator) of information controls who can access information

# Question

- Policy disallows cheating
  - Includes copying homework, with or without permission
- CS class has students do homework on computer
- Anne forgets to read-protect her homework file
- Bill copies it
- Who cheated?
  - Anne, Bill, or both?

# Answer Part 1

- Bill cheated
  - Policy forbids copying homework assignment
  - Bill did it
  - System entered unauthorized state (Bill having a copy of Anne's assignment)
- If not explicit in computer security policy, certainly implicit
  - Not credible that a unit of the university allows something that the university as a whole forbids, unless the unit explicitly says so

# Answer Part 2

- Anne didn't protect her homework
  - Not required by security policy
- She didn't breach security
- If policy said students had to read-protect homework files, then Anne did breach security
  - She didn't do this

# Mechanisms

- Entity or procedure that enforces some part of the security policy
  - Access controls (like bits to prevent someone from reading a homework file)
  - Disallowing people from bringing CDs and floppy disks into a computer facility to control what is placed on systems

# Key Points

- Policies describe *what* is allowed
- Mechanisms control *how* policies are enforced
- Trust underlies everything
- Next class:
  - Classic Confidentiality and Integrity policies