



Vulnerability Analysis



Vulnerability

- A defect in one system component or in the way the component is used
- By exploiting the bug, a threat agent can fire an unexpected behavior of the component
- The behavior allows the agent to violate the security policy
- This bug is a vulnerability



Vulnerability vs bug

- A bug may not result in a behavior that violates the security policy
- A bug that results in such a behavior is a vulnerability

=

any vulnerability is a bug but not the other way around



Taxonomies

- Several vulnerability taxonomies have been defined and may be adopted
- Each taxonomy has a goal (location discovery, evaluate the effects ...)
- Before applying a taxonomy we need to understand whether it satisfies with our goals



Location of the vulnerability

- Actions that are executed
 - Procedural
- People executing the action
 - Organization
- Hardware or software tools
 - ICT tools that are used



Some examples

- Action
 - A password communicated in an envelope that is not sealed
- People
 - Several administrators for the same machine
 - Task assigned to people that are not trained
- Tool
 - A password transmitted in clear on a network
 - No bound controls on a vector index



Taxonomy on tool vulns

A further classification, useful but not very rigorous

- Specification

- A tool that is more general than required (more functions, more parameters ...)

- Implementation

- A coding error in the program of the tool

- Structural

- The anomalous behavior arises when several components are integrated



Examples

- Specification = programming-in-the-large
 - A library is used that include more functions than those that are required
 - If someone succeeds in invoking some of the “useless” functions, anomalous behaviors may arise
 - Code reuse may introduce in a system some vulns in the code that is reused



Examples

- Implementation =
 - Well behaved input
 - No control on input parameters
 - Data and program confusion = jump into a data structure = stack /buffer/heap overflow
- These vulnerabilities are strongly dependent upon the native control in the language type system and in the language run time system
= no overflow with strong data types



Examples

- Structural: due to the composition of several components that are
 - Correct in isolation
 - Uncorrect when component
- Problems in the TCP/IP stack
- Some components delegate security checks to other ones, their correctness depends upon checks in other components



Another classification

- It considers an attack that exploit the vulnerability
 - Who can implement the attack
 - Those who own a local account
 - Those who can interact with the machine
 - ...
 - What can be achieved by the attack



Searching for vulns

- Any system can be described as the composition of standard and specialized (not standard) components
- Vulns and exploits for standard components are well known
- The search should focus on
 - Not standard components
 - Structural vulns due to the composition of standard components with not standard ones



Vulns and vulnerability scanning

- A vulnerability scanner is a tool that returns a set of vulns for each computer node in a network
- The scanner identifies the OS and the applications running on the node through a fingerprinting algorithms
- Then it access a database that maps each OS and application into a set of of public vulns
- Vulnerability scanning is a proper subset of a vulnerability analysis, the easiest one



Fingerprint

- The main mechanism to identify the OS and the application is the transmission of IP packets that violates the specifications
- All the applications and the OS reply in a standard way to a standard packet
- Each OS and application has its own reaction to a wrong packet that is not defined in the TCP/IP specification
- Several packets may be required to solve any ambiguity



Active and Passive Fingerprint

- An active scanner
 - sends malformed packet to some port of a node
 - analyzes the results to build the fingerprinting
- A passive scanner analyzes the traffic flowing in the network without disturbing the nodes
- Passive scanning/fingerprinting should be preferred in highly critical systems



False and true positive

- A scanner will signal a vulnerability even if the component has been patched
- This is what is called a false positive
- The only strategy to distinguish false and true positive is to actually implement an attack that exploits the vulnerability
- Not always possible on production systems even worse than active scanning



Not a boolean world

		Existence (gold standard)	
		Y	N
Test outcome	Y	True positive	False Positive
	N	False negative	True Negative

The problem arises anytime we can only deduce the existence of an object from some symptoms and do not have a direct access to it

Not a boolean world

		Condition (as determined by "Gold standard")		
		Condition positive	Condition negative	
Test outcome	Test outcome positive	True positive	False positive (Type I error)	Precision = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Test outcome positive}}$
	Test outcome negative	False negative (Type II error)	True negative	Negative predictive value = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Test outcome negative}}$
		Sensitivity = $\frac{\Sigma \text{ True positive}}{\Sigma \text{ Condition positive}}$	Specificity = $\frac{\Sigma \text{ True negative}}{\Sigma \text{ Condition negative}}$	Accuracy

$$\text{accuracy} = \frac{\text{number of true positives} + \text{number of true negatives}}{\text{number of true positives} + \text{false positives} + \text{false negatives} + \text{true negatives}}$$



Useful vulnerability

- It is worth exploiting a vulnerability and implementing an attack if this returns a large set of rights
- Most attacks may be implemented provided that there is a input value that is copied into a parameter of the procedure without checking its length
- This is not satisfied the procedure cannot be attacked



Tainting analysis

- A static analysis that returns the set of program variables that may receive an input variable and so be overflowed
- It returns a larger set than the actual one, worst case
- It can be improved by taking into account the behavior of the procedure used to copy the input value



Tainting analysis

- if x (y=input)
else (y=z);
w=y
- A tainting analysis tell us that w may have been tainted with an input value
- if x (y=input)
else (y=z);
copy (w, y)
- If copy checks the length of y before copying it into w, tainting but less danger



Non standard vulns in general

- To discover other vulns in a component, we consider that the vulns in a component defines a systemic property, the **robustness** of the component
- Systemic = it depends upon a component and upon its relations with other components
- There is a relation among
 - Search of vulns
 - A systemic property = Robustness



Robustness in ICT

- Robustness of a component =
The ability of the component of avoiding damage to the overall system when the component specifications are violated
- Violation of the specifications =
 - Inputs differs from the specified one
 - Available resources differs ...
 - ... (enumerating badness)



Robustness in biology

- Redundancy
- Feedback
 - Monitoring of the behavior
 - Tuning of the behavior
- Modularity
- Confinement of anomalous behavior
- Uncorrect components are removed and replaced
- No single point of failure

If any of these features is not satisfied a vuln arises



Robustness vs Vulnerability

- Any set of rules that defines how to build a robust component also defines a set of rules to discover vulnerabilities
- If the rules are violated, then the component is not robust, then there are some vulnerabilities



Robustness

- It differs from performance, efficiency, ease of use,
- It can be increased only by decreasing performance, efficiency, ease of use, ...



No Free Lunch Theorem



Robustness

- ♦ Let us consider a program that given the name of a worker returns the worker's salary
- ♦ The program is
 - correct if the salary is correct for any worker
 - high performance if the salary is computed in a very short time
 - easy to use if you learn to use it in a short time
 - robust ????



Robust what happens if

- Wrong record format in the file
- No worker with the name
- The name is 457 characters
- The allocated memory is smaller than expected
- No file with the worker names
- No file with info to compute the salary
- No disk

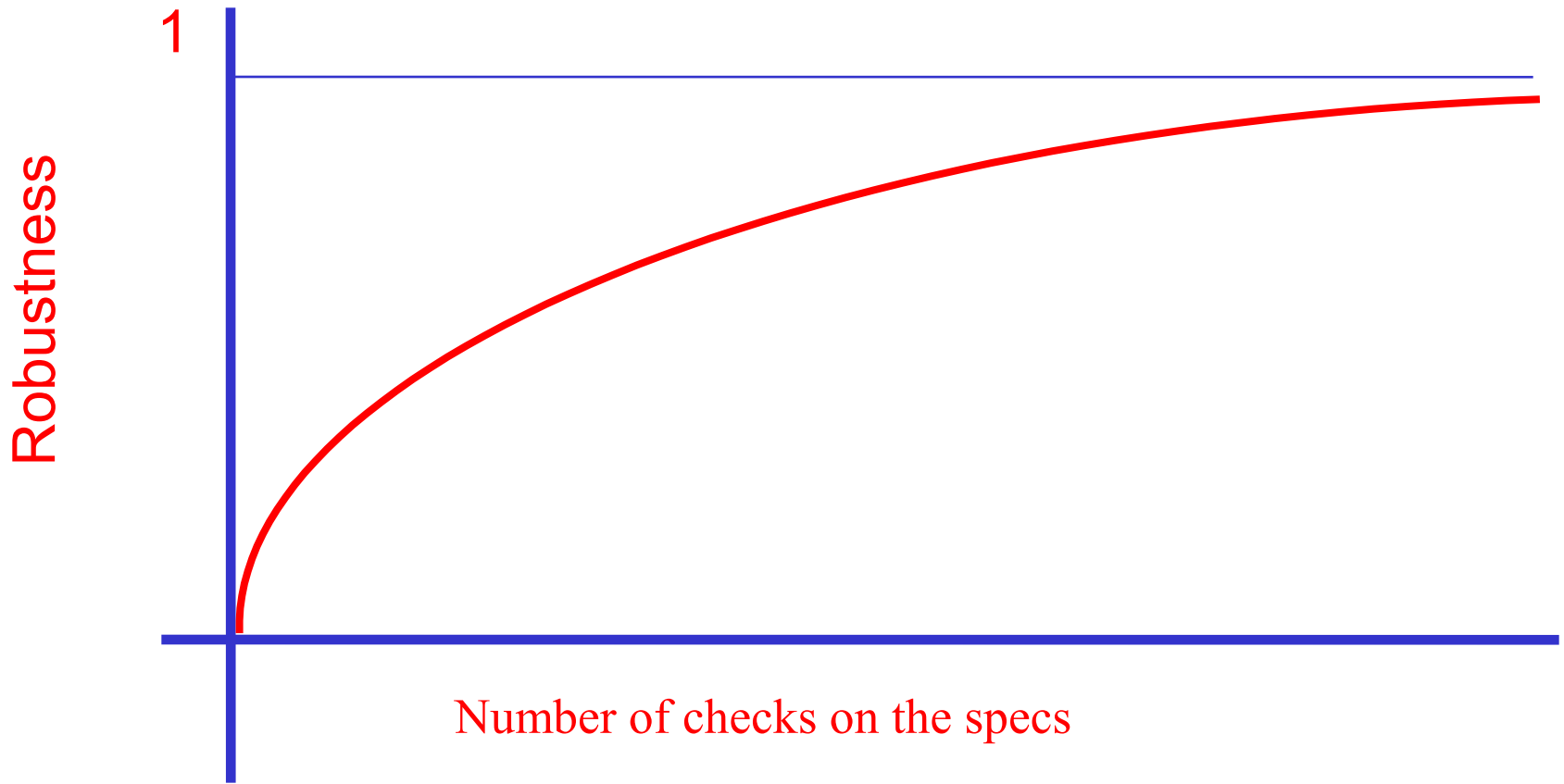


How much robustness

- It is almost impossible to define in advance any violation (this is a case of enumerating badness)
- Robustness is not a 0/1 property
- A robustness measure lies in the range 0..1
- 1 is an asymptotic value
- The value depends upon the number of checks in the program of the component to discover whether the specs are satisfied or not before using a given input or a given resource



How much robust?





How much robust?

- The value depends upon the number of checks
- Robustness $\rightarrow 1$ if the number of checks $\rightarrow \infty$
- Usually the checks are useless because the probability of violating the specs is very, very low, provided that the specs are corrects
- A compromise is required because the number of checks reduces the component performance
 - \rightarrow they slow downs the component because they are implemented through instructions as any other function of the components



Robustness

- It has been experimentally confirmed that even trivial checks can improve the component robustness
- This implies that complex checks should be adopted only after trivial ones
- Most efficient checks are those related to data types



Robustness vs Vulns

- We can define an ideal system as one where the components implement any control
- The ideal system is the asymptote of those that apply more and more checks
- Any difference between the ideal system and the current one may be a vulnerability
- If it is a vulnerability depends upon the context and the cost of the control
- Any set of guidelines to build a system also defines the potential vulns of the system

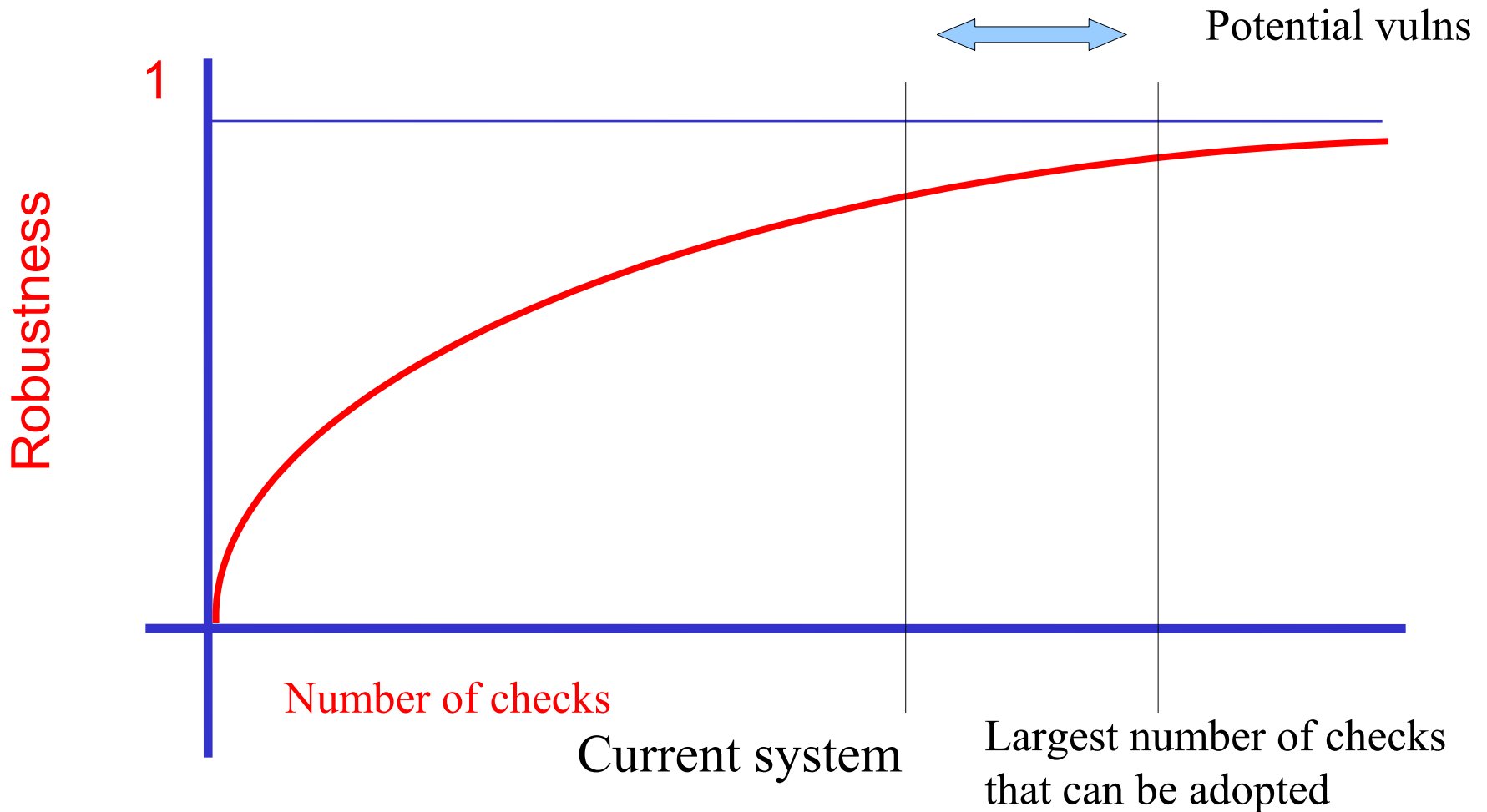


Robustness vs Vulns

- Some differences between the ideal system and the current one cannot be avoided if some controls have not been adopted to satisfy some performance requirements
- Other differences may be unrelated to performance and, hence, controls should be introduced
- The key strategy to discover vulnerabilities is to evaluate the cost of missing control and contrast it against the required efficiency



How much robust?





Safety vs Security

- Robustness may also be adopted to evaluate the safety of a system
- Security differs because we are interested in robustness with respect to intelligent attacks rather than to random failures



Safety vs Security

- Safety is proportional to the ratio of anomalous behaviors vs the overall number of behaviours
- A fault results in an anomalous behavior but, if faults are not related with one another, then the ratio shows the cases where faults are not controlled and confined
- In security, the attacker tries to force the system to behave in an anomalous way by attacking those components that influence the behavior of interest
- Safety = random faults / Security = intelligent faults



Safety vs Security

- Both applies the notion of probability and of risk
- Safety is focused on independent probability distribution
- Security is focused on conditional probability
 - There are some vulns, hence
 - There are some attackers, hence
 - The attacker can implement the attacks ...

Design principles for robustness (Saltzer&Schroder) or rules to discover vulnerabilities



- Economy of mechanisms
- Fail safe default (Default deny)
- Complete mediation
- Open design
- Separation of privilege
- Least Privilege
- Least common mechanism
- Psychological Acceptability
- **Work factor**
- **Compromise recording**



8 or 10 principles?

- After introducing the first 8 principles, S&S say:

Analysts of traditional physical security systems have suggested two further design principles which, unfortunately, apply only imperfectly to computer systems

- The principles applies to both a system and the mechanisms we introduce to secure the system



P1-Economy of mechanisms

Keep the design as simple and small as possible

- Simple implies that less things can go wrong and when errors occur, they are easier to find, understand and fix
- Vulns are proportional to the complexity of a mechanisms and to the code to implement it
cyclomatic number to find software bug
- Complexity can be achieved by composition
- SO Hardening = remove useless OS functionalities for applications of interest



P1- Economy of mechanism

- Esokernel and microkernel=
Avoid the implementation of complex functions in the kernel
- A strong integration between the OS kernel and the applications not only violates modularity principles but helps the spreading of errors (cascade failures)



P1- Economy of mechanisms

- Simplify the interface
- Complex operations should be implemented by composing simple operations
- If the operations are rather complex (and hence powerful), we may be forced to allow a user to invoke a powerful operation even to implement simple operations and this increases the user rights (related to the least privilege principle)



P2-Fail safe default (Default deny)

Base access decisions on permission rather than exclusion

- Burden of proof is on the principal seeking permission
- If the protection system fails, then legitimate access is denied but this also denies illegitimate access
- The initial state of the system is correct



P3-Complete Mediation

Every access to every object must be checked for authority

- Usually it is done once, on first access, but if permissions change after, unauthorized accesses are possible
- Performance gains achieved by caching the result of an authority check should be examined skeptically
- Each operation that is not controlled is a potential vuln as it may be invoked without authority



Access control matrix

	object		Which object operations the subject is entitled to invoke
subject	rights		←
			Condition 1. necessary 2. not sufficient For a secure system



Access control matrix

- Security requires this matrix exists for each system layer
- Furthermore, there is also a matrix for each application or virtual machine at the application layer
- Coherency among these matrices
- A matrix may be so large that it has to be stored on a secondary storage



Rights in $acm[i,j]$ -I

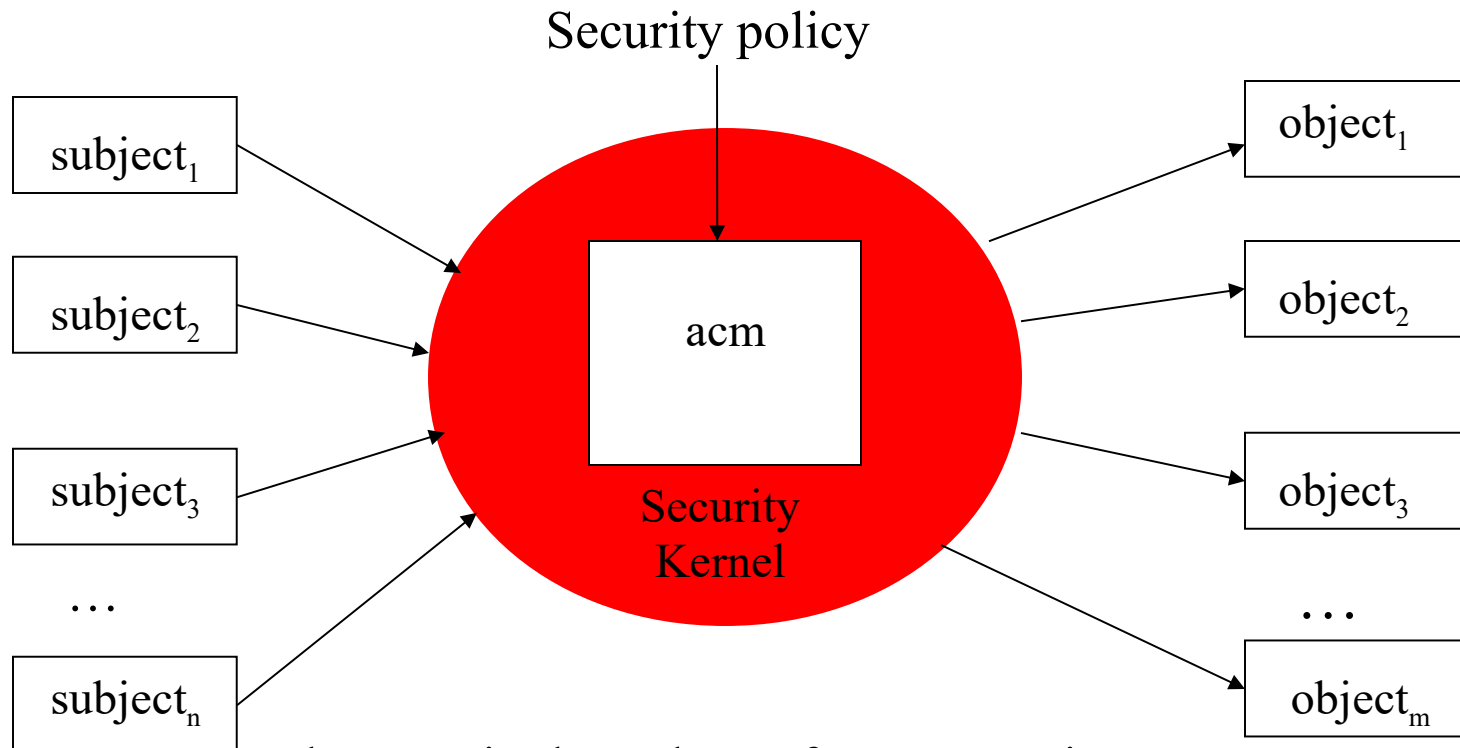
- DAC security policy = assigned by the owner of the j-th object
- MAC security policy = they also depends on the levels of the i-th subject and the j-th object
- In both cases further constraints may have to be satisfied before the subject can actually exploit the rights that have been assigned



Rights in $acm[i,j]$ -II

- The access control or protection matrix is a highly dynamic data structure
- Dynamicity is due to
 - Dynamic creation and destruction of subjects and objects
 - Some security policies dynamically updates the rights of each subject according to the operations the considered subject has invoked

Acm: a typical (centralized) implementation



The security kernel or reference monitor (TCB) mediate the subject attempts to invoke the operations defined by the objects



Access Control Matrix

- This is a logical data structure for which a large number of concrete implementations is possible
- Sometime the acm is not implemented by a matrix
- Problems arises when no all the subjects are known in advance (network services)
 - In this case, a row of the acm is paired with a class of subjects
 - Rules to map each subject into a class have to be defined



Security Kernel or Reference Monitor

- It belongs to the Trusted Computing Base (TCB)
= its correctness is a necessary condition for the correct implementation of the security policy
- As small as possible to apply formal techniques to prove its correctness
- A basis for induction proof of security properties
- In some systems it is stored in a tamper proof memory to prevent illegal updates



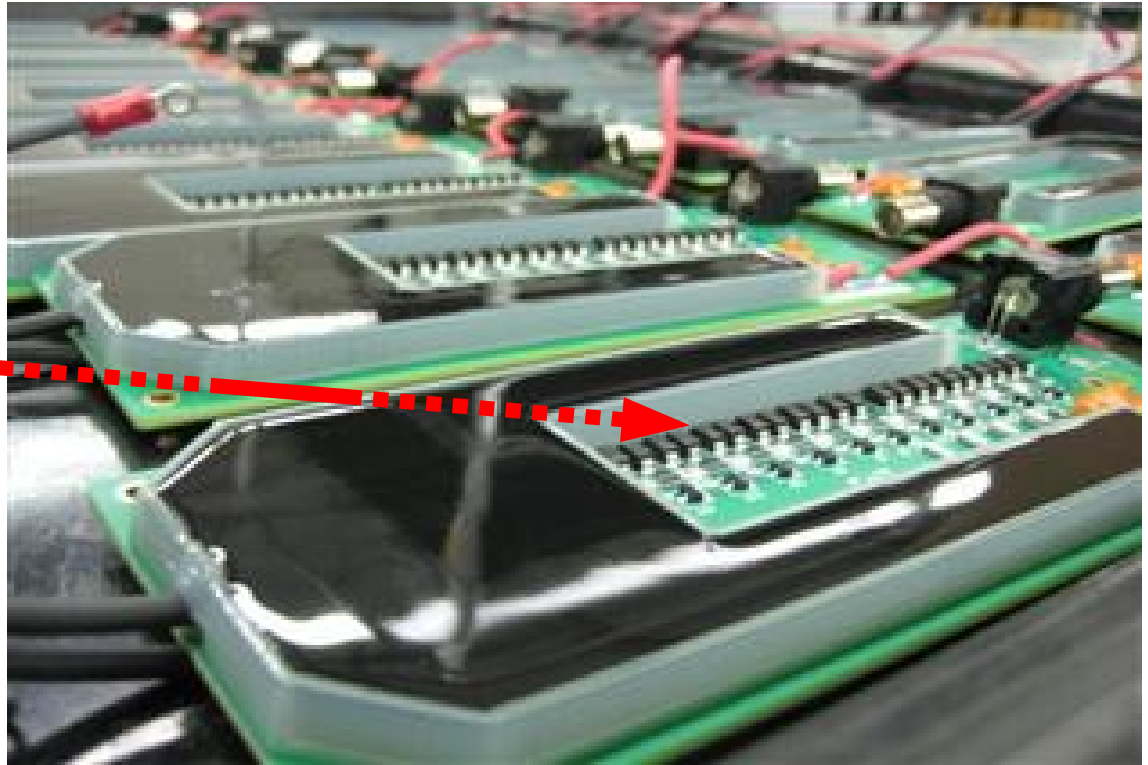
Tamper proof

- A component where any physical attack is
 - Prevented or at least
 - Detected
- All the components are glued with silicone
- Memory chipes are protected by an electrified grid that cancel any information as soon as an attack is attempted

Silicone tamper proof



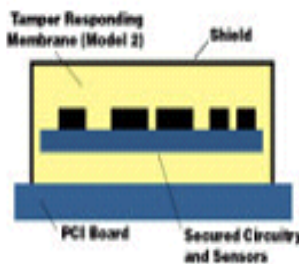
Silicone



Secure Coprocessor

IBM 4758 hardware

The IBM PCI Cryptographic Coprocessors are state-of-the-art secure subsystems that you can install in server systems to perform DES and public-key cryptography. You can also load software for highly sensitive processing, such as the minting of electronic postage, which must perform its intended function even when under the physical control of a motivated adversary.



The secure Coprocessor module is mounted on a two-thirds length PCI version 2.1 board and fits in a single slot that provides +5.0VDC +/- 15% and +12VDC +/- 15% power and meets other environmental requirements as listed in chapter 2 of the [General Information Manual](#). (3.3VDC variations of the Models 002 and 023 are also available to OEM customers and used in IBM @server pSeries, and IBM @server iSeries and IBM @server zSeries servers.) The sealed Coprocessor module incorporates physical penetration, power sequencing, temperature, and radiation sensors to detect physical attacks against the encapsulated subsystem. Batteries provide backup power that is active from the time of factory certification until the end of the product's useful life. Any detected tamper event results in loss of power which immediately causes the zeroization of internal secrets and the destruction of the factory certification.

Complete mediation + fail safe default



- If both principles are applied
 - The system starts in a secure state
 - Provided that the security kernel is correct, only secure transitions are enabled
- Induction proofs on reachable states
- If fail safe default does not hold no induction basis exists



Complete mediation+ fail safe default

- Let us assume that to grant a right R on an operation op the object $Ob(op)$ has to be updated
- In the initial state no subject owns the right of updating $Ob(op)$
 - No subject can grant this right
 - Hence no subject can be granted this right



Access control matrix

- An implicit assumption is that the identity of the subject is checked before accessing the matrix
 - ⇒ how can we control that a subject that claims of being A is A
 - Explicit check in the security kernel
 - Password
 - One-time password
 - Challenge response
 - Electronic signature



One time password

- A function F with at least two parameters
 - S a secret value
 - N the number of received requests (defined in an implicit or explicit way)
- The subject to be authenticated computes and transmits $F(S, N)$
- The receiver computes again $F(S, N)$ and checks
- Synchronization on the value of N



Challenge - response

- Partners agree on a function F and keep it secret
- F has an input parameter x
- One of the partners sends y (challenge)
- The receiver computes $F(y)$ and sends back the result
- Also the challenger computes $F(y)$ to check whether the response is correct



Complete mediation: problems

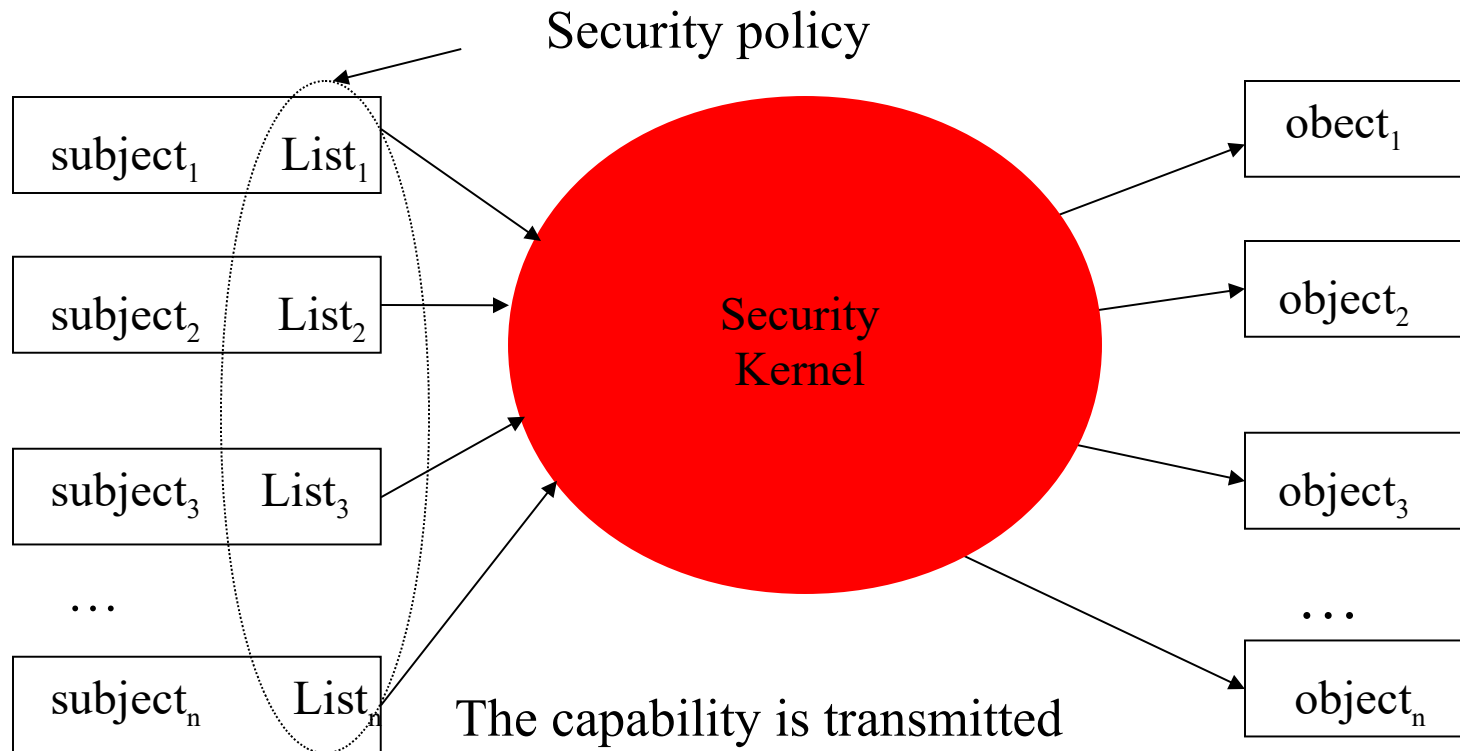
- High performance in the access to acm is required due to the huge number of checks
- An implementation where a centralized data structure is shared among all the subjects and the objects usually cannot achieve an acceptable performance
- A distributed solution is to be preferred so that the overhead is independent of the number of objects



Parallel Solutions - 1

- Capability list= a row based organization of the matrix
 - A capability = $\langle \text{object address} , \text{rights} \rangle$ = a generalization of pointer = a protected pointer
 - When invoking an operation, the subject specifies which of its capability has to be used for the operation
 - The specification is implicit using a pointer

Acm as capability lists



The capability is transmitted to the security kernel that checks whether it enables the operation
The SK does not manage the ACM



Capability -I

- Invocation $op_i(obj_j, par, n)$ = execute the i-th operation of the j-th object as enabled by the n-th capability in the subject list
- If S transmits a capability (a pointer) to another subject S' then S can delegate S' to invoke an operation S' is not entitled to
- Capability = ticket for an object
- Delegation increases the number of instances of a given rights that, in turn, increases the complexity of right revocation



Capability - II

- Capabilities are generated by the security kernel that distributes them to the subject
- A subject should only be able
 - to store
 - to read (use)
 - to copy (delegation)
 - but not to update a capability
- Only the security kernel can update a capability
- The probability of a successful attack against the security policy increases since rights are stored in the subject (the worst place to store rights)



Capability -III

- In some cases the MMU may implement an efficient hw/fw support for capabilities at the OS levels
- The capability list is stored in the MMU
- The MMU
 - checks the rights in parallel with the address translation
 - prevent a subject from updating its list



Capability -IV

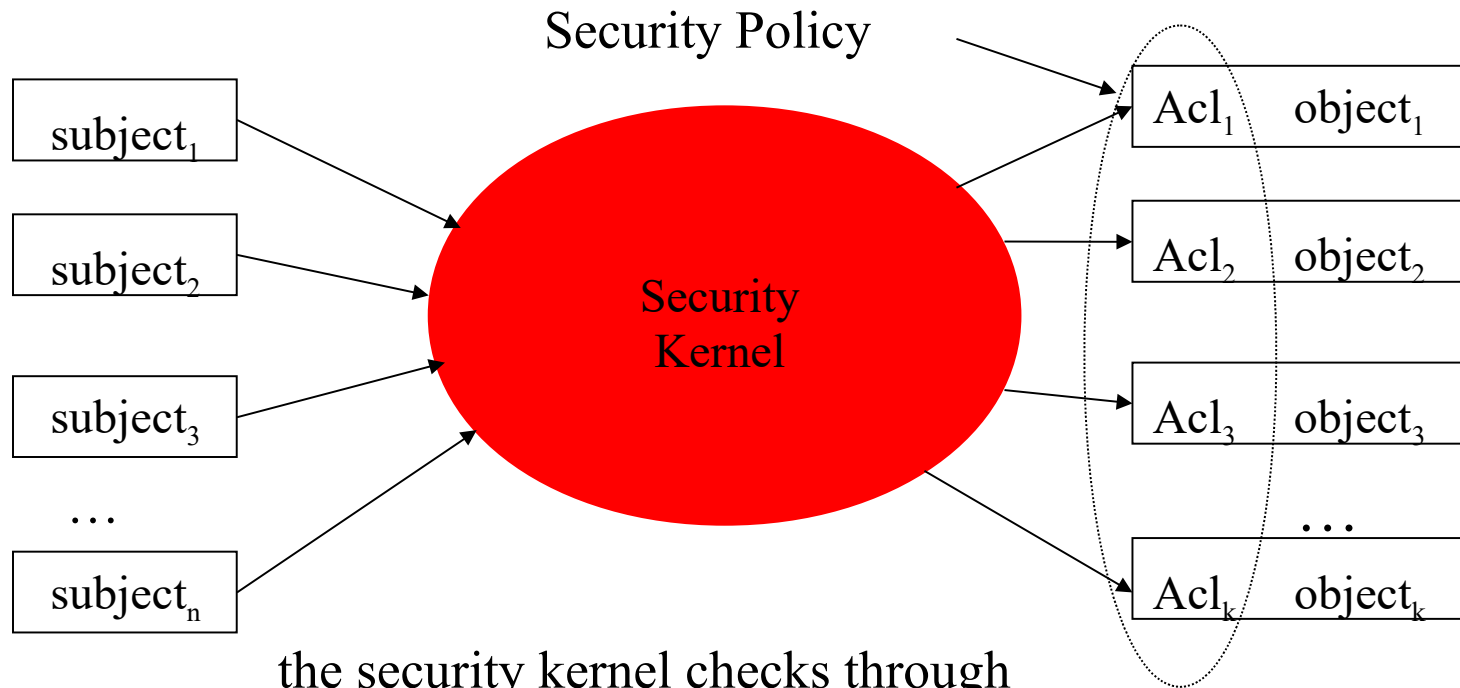
- Address translation exploits a segment/page table that store the physical address
- For each segment/page some operations are defined among a predefined set
(read, write, fetch)
- Some processors do not check the rights if the segment/page is already stored in the cache or if the address has already been translated



Complete Mediation - 2

- Access control list = a column based organization of the acm
- One list for each object
- Each list element stores the rights of a distinct object
- Even in this case the control may be implemented by the Security Kernel
- A centralized structure for each object

ACM: ACL



the security kernel checks through the object ACL that the security policy is satisfied

The checks may also be implemented by the object



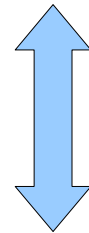
Access control list

- A more flexible solution may be achieved through
 - Partition of the subjects
 - The sequential scanning of the list (no direct access is possible because the subject does not know its position)

If subject \in Set1 then {op1, op2}

else If subject \in Set2 then {op3, op4}

else {op5}



this is an ACL!

- the subjects are partitioned into three sets
- in this way we can grant rights even to **subjects not known in advance**. This is not possible for capabilities and makes it possible to define acs for **web services**



HW/FW support for ACL

- Associative memory where the key may be
 - Subject → set of rights
 - Subject, operation → boolean
- FPGA that implements a function that is a chain of if conditions about
 - Sets of users
 - Priority among sets



ACL vs Unix files

The bit array paired with each file and that defines

- Owner rights
- Group owner rights
- Other users rights

Is an implementation of the file ACL



ACL and file descriptor

```
struct stat {  
    mode_t st_mode; // File type & mode           access control list + set uid bit  
    ino_t st_ino; // i-node number  
    dev_t st_dev; // device number (file system)  
    dev_t st_rdev; // device n. for special files  
    nlink_t st_nlink; // number of links  
    uid_t st_uid; // user ID of owner  
    gid_t st_gid; // group ID of owner  
    off_t st_size; // size in bytes, for reg. files  
    time_t st_atime; // time of last access  
    time_t st_mtime; // time of last modif.  
    time_t st_ctime; // time of last status change  
    long st_blksize; // best I/O block size  
    long st_blocks; // number of 512-byte blocks  
}
```



Unix/Linux -I

- ACL are defined in terms of process identifier
 - Real user ID → owner
 - Effective user ID
 - Saved user ID
- in Linux we also have
- File system ID



ACL vs Router

- Router ACL are built by composing two cases
 - IP Range₁ → route
 - messages from these nodes are routed
 - IP Range₂ → drop
 - messages from these nodes are dropped
- A list for each input/output connection can be built that specifies the IP addressed paired with the traffic that can cross any router connection
- The address have not to be known in advance



ACL & Router

- ACL of input 1

- 131.114.*.* → route
- 131.4.5.6 → route
- 131.4.*.* → drop

Only traffic from 131.4.*.* is dropped but that from 131.4.5.6

- ACL of output 1

- 131.114.*.* → drop
- 131.4.*.* → drop

No address in 131.4.*.* can send traffic to the network connected to output 1



Routing in Linux: iptables

- Input chain: rules for the packets addressed to the node
- Output chain: rules for the packets produced by the node
- Forward chain: rules for the packets that cross the node
- Default allow



Nat table

- Prerouting chain= any input packet
- Postrouting chain = any output packet
- Output chain = any produced packet



Routing in Linux

- Drop
- Route
- Return – return to the invoking chain
- Queue – transmit to user space
- Log
- Reject
- Dnat/Snat/Masquerade



Examples

- iptables -A INPUT -p UDP drop

Insert a rule in the input chain to drop UDP packet

- A = add a rule
- INPUT = in the input chain
- p UDP = the protocol

- iptables -A INPUT -p TCP -dport 156 drop

Drop any TCP packet addressed to port 156

- iptables -N newcontrol

Create a new chain where new controls can be later inserted



An important point

- Anyone is aware and agrees of the importance of controlling the network traffic that enters a network
- Hence these controls are critical in the border router that connects a network to a public one
- Are there any reasons to check the traffic leaving a network?



Controlling the output traffic

- The control of output traffic is an important mechanism to discover successful attacks against the network (egress filtering)
- If someone is controlling a node and stealing information in the node we can discover illegal connection of the node to some outside network
- A planned DDOS attack can be discovered before it is actually executed



ACMatrix, subjects and objects

- As the number of subjects and objects increases, the complexity of
 - defining the ac matrix
 - checking its correctness
 - achieving full mediationstrongly increases
- Some solutions have been proposed to simplify the definition of the matrix



Role vs subject

- When (subject = a final user) we can apply the notion of role
- Role =
 - A professional profile and the corresponding rights
 - Strongly depends upon the applicative environment
- Any role is paired with
 - A set of users that can be assigned that role
 - A set of rights
- Role Based Access Control
- Rights are not assigned to users but to roles
- A user U acquires the rights when U is assigned a given role
- When U leaves the role, the rights are lost



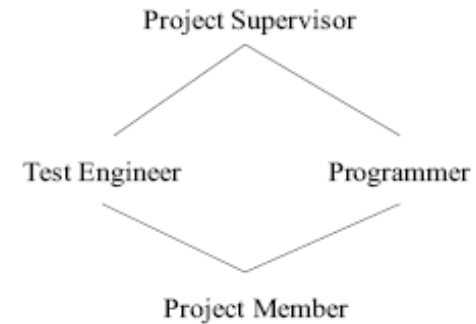
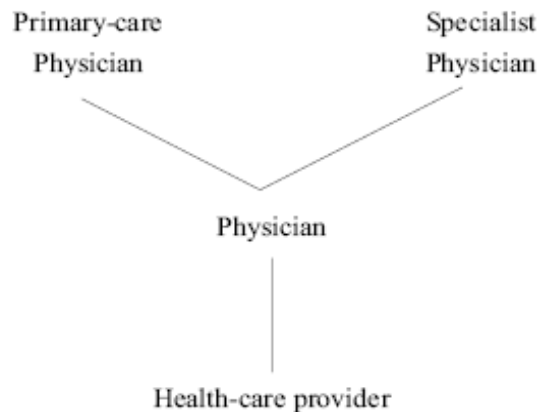
Role- II

- Rules are defined to specify when a role may be assigned and when it is lost
- The rules may take into account previous operations executed by the users
- Any role change may require a password to identify the user



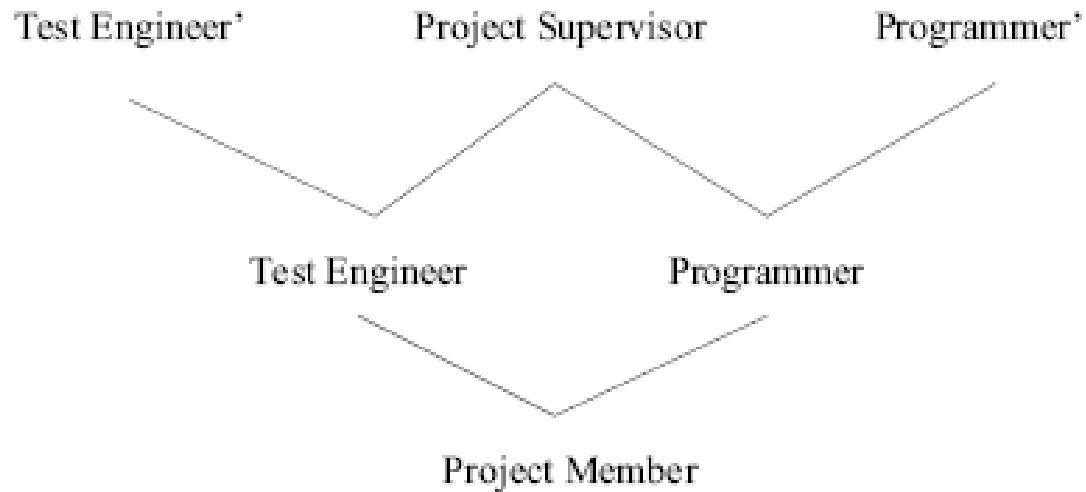
Role hierarchy - I

- Role may be partially order
- If a role is larger than another one then it includes all its rights



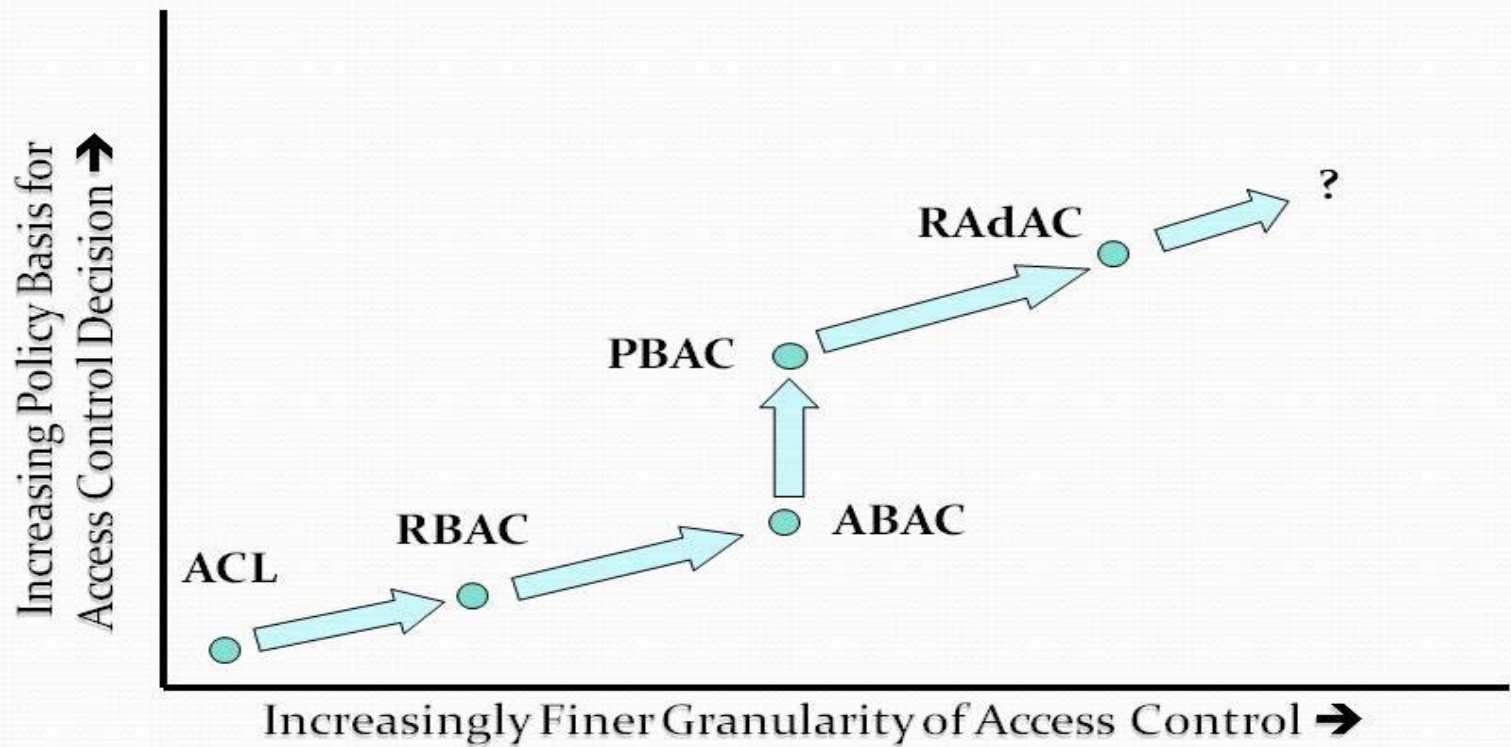


Hierarchy II



Other models (defined in the following)

Access Control Models





Attribute Based Access Control

- Each subject is paired with a set of attributes
- The right of invoking an operation is a function of the current values of the attributes
- Not supported by OS but only at the application level
- To support it at the OS level a standard set of attributes for all the user has to be defined



ABAC

- Attributes =
 - Role
 - Security level
 - IP address of the user system
- As an example the operation can be executed if
 - Role= system manager
 - (Role= system manager) AND (ip = local)
 - (Level > confidential) AND (ip = local) AND (8 <local time <16)



Risk Based Access Control

- The risk posed to the system because of the operation is evaluated
- The evaluation takes into account attributes of the system, of the user etc to decide whether the rights should be granted
- No reasonable implementation



P4-Open Design - I

The design should not be secret

or

The security should not depend on the secrecy of the design or of the implementation

- Popularly misunderstood to mean that source code should be public

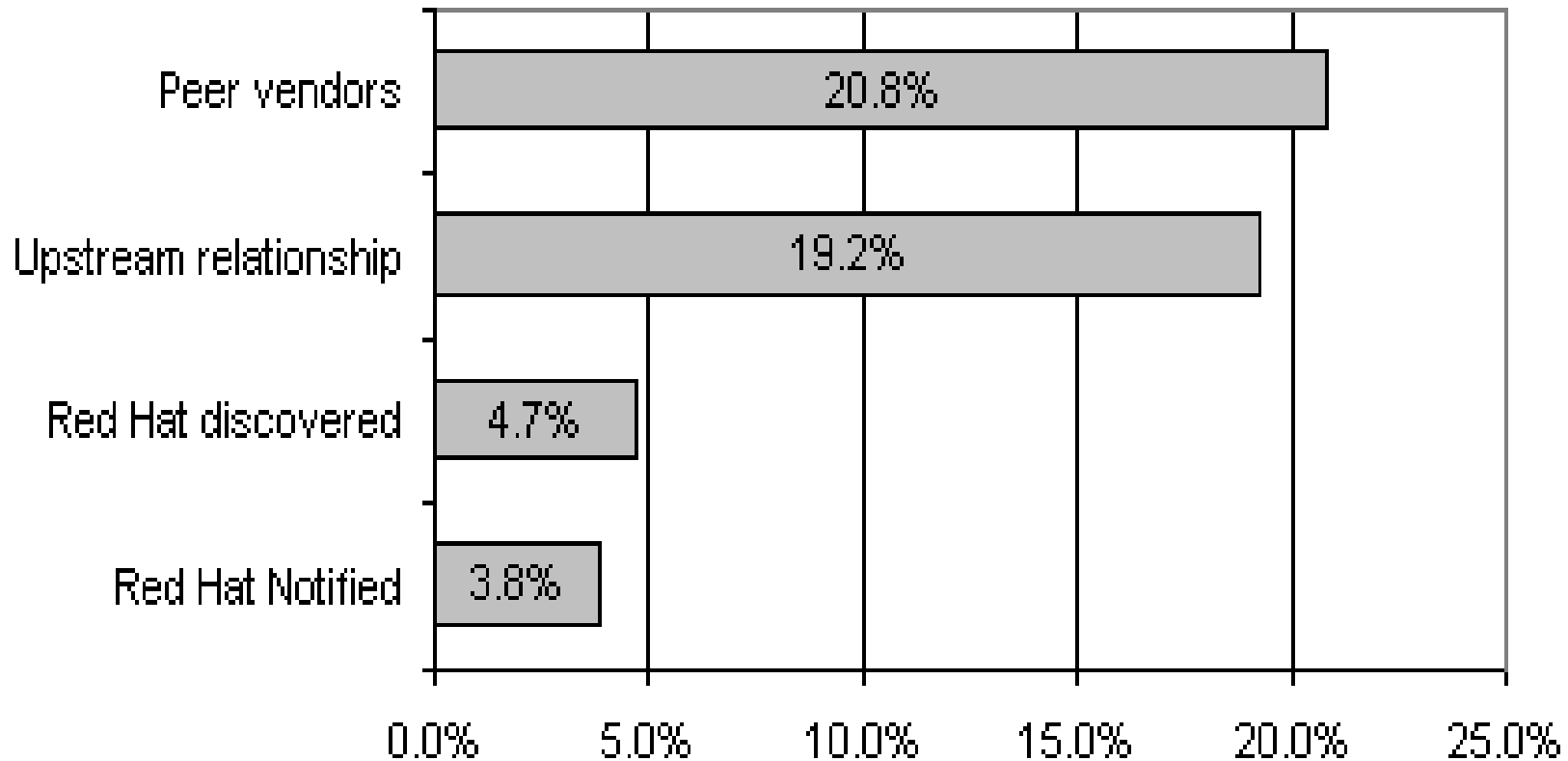


P4-Open Design - II

- A system peer review is fundamental to discover vulns in the design and/on in the implementation
- This implies that the disclosure of the design and of the implementation is useful only if
 - it results in a peer review
 - any peer that discovers a vulnerability communicates it to the owner
- If there is not a peer review (no peer) or if the vulns are not reported to the owner then the open design is useless
- Strength and weakness of open source



Vulns vs open design





P5 - Separation of privilege

Where feasible, a protection mechanism that requires two keys to unlock it is more robust and flexible than one that allows access to the presenter of only a single key

or

Require multiple conditions to grant privilege

- Separation of duty
- Defence in depth



P5 – Separation of privilege

- A complex operation should be decomposed into simpler operations
- Each simple operation is enabled by a proper rights
- We can control that the subject owns both
 - The right of invoking the complex op
 - The right of invoking each simple op



Example

- Op = transfer some money from account1 to account2
- 5 rights
 - Transfer money
 - Read account1
 - Update account1
 - Read account2
 - Update account2
- Someone can transfer money but not from account1 or to account2



P6 – Least privilege - I

Every subject should operate using the least set of privileges necessary to complete its job

or

*A subject should be given only those privileges it needs to complete its task and only **for the time** to complete it*

- A useless right is a vuln
- Rights granted as needed, revoked after used
- The ac matrix is a dynamic data structure
- Rights are assigned and revoked as the computation evolves



P6 – Least privilege - II

- This principle should be applied even if the security policy is static as it defines **how rights should be managed** rather than how they are assigned to each subject
- If, in a given time interval, a subject does not need a right then the right should be revoked and the acm should be updated to prevent the subject from using the right in the interval
- The right is granted at the end of the interval
- Extreme version of can know/need to know



Least Privilege - III

- Protection Domain Switching = the same subject is executed but the rights in the proper positions of the acm are updated
- Protection Domain Switching = update of an acm row
- We can have a PD switching even without a context switching
- The corresponding overhead is a function of the implementation level and the adopted implementation of the acm (capability vs acl)
- Revoking a right is not simple with capabilities



Least Privilege - IV

- An alternative definition is focused on the small protection domains
- As the size of the protection domain decreases, it also decreases the risk due to an attack against the considered subject
- If the protection domains are not small then revoke grants when not needed and grant when needed



Least Privilege - V

- The system designer has to choose the proper compromise because a full application of this principle may result in low performances
 - ⇔ for each command that is executed, the acm should to be updated
 - ⇔ the asymptotic system is too slow



Least privilege – In principle

When/how the domain switching is fired

- 1) Through further, proper instructions
- 2) Some language constructs also fire the domain switching



Least Privilege – Common solution

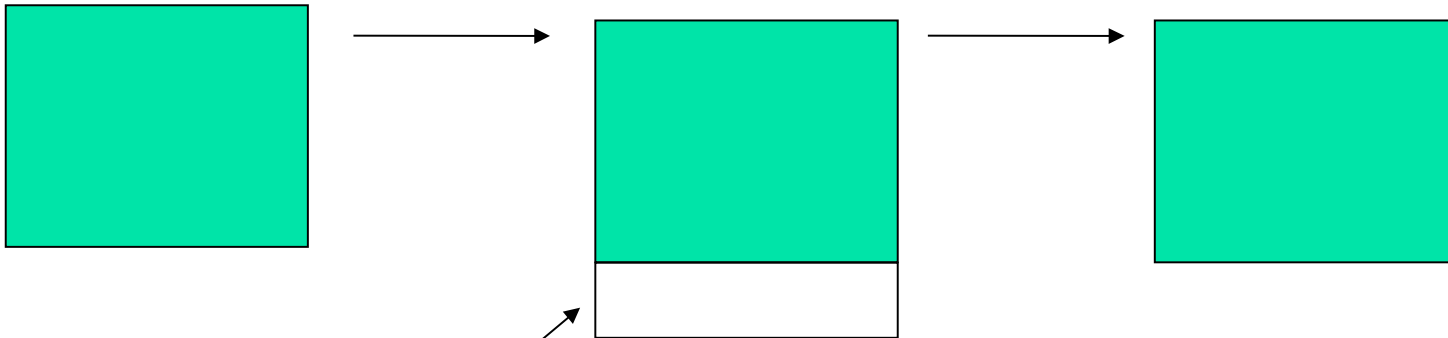
- In the classical solution pairs a domain switching occurs when
 - A procedure (method) is called
 - A procedure (method) returns
- A new row is created (call) and destroyed (return) rather than updating a row
 - When the procedure is invoked, a new row that defines its rights is created
 - The row is destroyed when the procedure returns
 - Rights are paired with the instance of a procedure executed by (or on behalf of) a subject rather than with the procedure code or with the subject



History of the ac matrix

Procedure called

Procedure ends



Row paired
with the new instance
subject=new instance

Rows created and destroyed
Rather than updated



Least Privilege – Common solution

- The rights in the new row are a function of
 - The private variables of the method (they depends upon the variable types),
 - Input parameters (type of the parameters and the kind of access to the parameters)
- The structure of the program into classes/ methods defines the strategy to manage the rights granted to the subjects on the program data structures
- The programmer can choose the size of each protection domain
- Domain switching is handled in an automatic way



Example

Op(x, y)

a :

- If two subjects (programs) invoke this op, each program has its own row, we have two local copies of a or one copy if shared variables are supported (depends upon the type of a)
- Each row enables the program to access its own parameters and to non shared local variables
- If a static acm is adopted, the management of rights is rather more complex and access of a program to the parameters of the other program is simplified



Least Privilege - Amplification

- It may be useful if the set of rights of the invoked procedure differs from that of the invoker
As an example, consider the case where the procedure knows the implementation of the object
- Rights are amplified: provided that some rights are owned, other may be granted
- Term is misleading because the set of rights that is granted may differ from the original one rather than including it



Least privilege vs objects

- The domain decomposition principle is fully coherent with an object oriented methodology
- A simple object defines a small protection domain (a few internal variables) that can be managed in a simple way
- Even if a simple object is successfully attacked, the attack has a low impact and can be easily detected
- Sharing among objects should always be minimized



Least privilege – message passing

- In the case of message passing, subjects are processes/threads interacting through ports or channels
- To satisfy the principle
 - Distinct interactions are implemented through distinct ports,
 - Ports can be opened/closed (created/destroyed)
 - If an interaction may occur, then the corresponding port is open/created
 - The port is closed/destroyed as soon as the interaction is no longer possible



Least privilege – message passing

- Closed port \neq (open port + mechanism to discard messages)
 - The overhead to discard messages is much lower if the port is closed or if does not exist
 - Messages can be discarded as they are routed
- In the most dangerous case, the subject can do nothing because it is always busy to discard messages (Denial of Service)



Least Privilege – Unix - I

- The principle is violated because root has any right (and it is the target of any attack)
- This strongly simplifies attacks, any procedure executed by root is a target
- Management countermeasures such as having distinct administrators for a system
- Further countermeasures are technological such as recording (logging) any operation that has been invoked by root (where???)



Least Privilege – Unix - II

- Chroot constrains the access to the file system by defining a new root
- Jail (BSD) makes it possible to constrain other operation such as network connection
- These are implementation of sandbox = a minimal environment for untrusted application



Sandbox

- Definitely a bad idea
- Any sandbox implementation has been violated
- When the subject succeeds in leaving the sandbox, no other countermeasure exist



P7- Least common mechanism

Minimize the amount of mechanisms common to more than one user and depended on by all users

- Mechanisms should not be shared
 - Information can flow along shared channels
 - Covert channels
- Isolation
 - Virtual machines
 - Sandboxes



P7- Least common mechanism

- A powerful mechanism, if useful, should be decomposed into simpler ones
- If just one mechanism is used to implement several operations
 - Several subjects will be granted the rights of invoking the mechanism
 - This hides the fact that there are several distinct operations and several distinct rights
 - The least privilege cannot be satisfied



P7 – Least common mechanism

- By decomposing operations into simpler ones we can better satisfy separation of privilege and least privilege
- Simpler operations makes it possible to assign to each subject only the rights it needs and it is entitled to



P8 - Psychological Acceptability

The human interface should be designed for ease of use so that users routinely and automatically accept the protection mechanisms correctly

or

Do not adopt policies users will surely violate

- Security mechanisms should not add to difficulty of accessing resource
 - Hide complexity introduced by security mechanisms
 - Ease of installation, configuration, use
 - Human factors critical here



Last two principles

- Recall they have been introduced because even if the other are satisfied a vulnerability is possible
- They are useful if some attacks are successful



P9 – Work factor

Compare the cost of circumventing the mechanism with the resources of a potential attacker

- The probability of a successful attack increases with the resources the attacker can access
- The cost of circumventing a mechanism is the attacker work factor
- “Ceteris paribus” a mechanism is better than another if it can be defeated only through a larger amount of work
- Several mechanisms can be defeated only by indirect strategies, such as waiting for an hardware failure
- Reliable estimates of the work are very complex anytime several attacks are required to violate a system



P10 – Compromise recording

Mechanisms that reliably record a compromise of information may replace more elaborate ones that completely prevent loss

- If they produce a tamperproof record that is reported to the owner, they support the discover of unauthorized use.
- In computer systems it is difficult to guarantee discovery after the system has been attacked.
- Logical damage (and internally stored records of tampering) can be undone by a clever attacker



P10 – Compromise recording

- Useful to collect information about attacks, goals and threat
- Any collected information can be used to evaluate the robustness that a system may offer as well as to improve the accuracy of the various analysis in a risk assessment



Compromise recording

- A log file that records, at least, any of
 - Login attempt
 - Failed login
 - Access to critical resources
- Protection of log file
 - write once memory (e.g. paper)
 - insert a sequence number to discover log manipulation
 - insertion in a record of a value that is a function of all the previous records
- Forensics = the file should be structured so that it can be used to prosecute the attacker and as a legal source of evidence in an investigation



Logging policies

What happen when a file is full?

- Throw away – all the data are destroyed
- Reset – rotation within a file
- Rotate – rotation among several files
- Compress and archive – stored in a low cost memory (there are some laws that require that some data are preserved)



Throwing away log files

- The worst solution
 - The files are a source of evidence and of information about security
 - They may also be useful for safety
- Even if the law entitles us to destroy the logs shortly after they are collected, it is better to preserve them for some months
 - This is the interval of time that is required to discover any intrusion



Rotating log files

- N distinct files
 - logfile.1 , logfile.2, ... logfile.n
- Each day a distinct file is used



Compress and archive

- Better solution that takes into account
 - Forensics investigation
 - Commercial problems with clients, suppliers
- Log are copied onto low cost, removable memory devices

The logo consists of several overlapping squares in yellow, red, and blue, with a vertical black line passing through them.

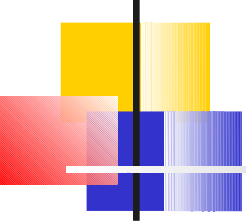
Syslog

- A logging system to store information produced by the kernel and by system utilities
- It enables a classification of log messages according to the source and the critical level of the event
- Messages can be addressed to several destinations

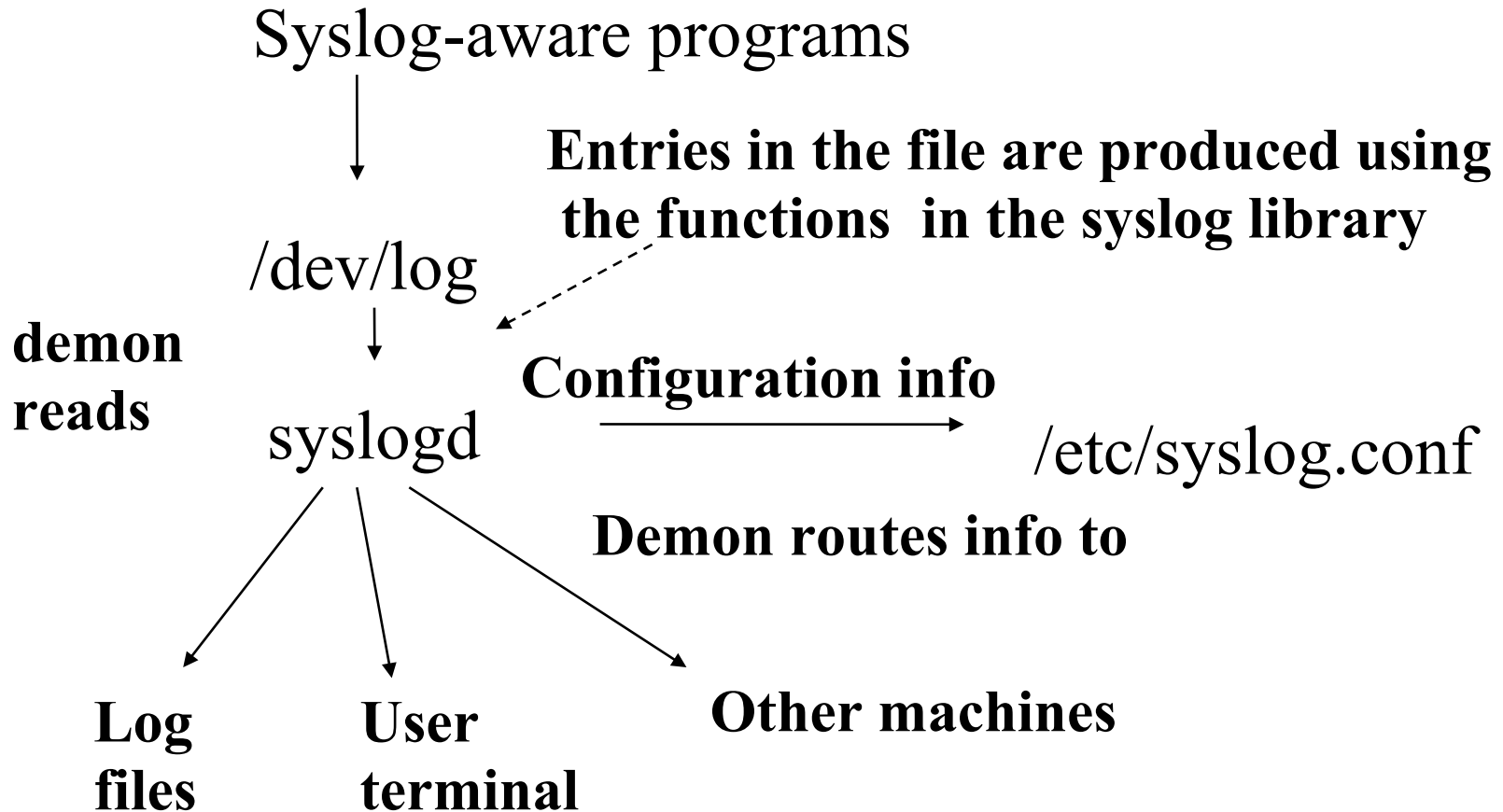


Syslog: 3 elements

- Syslogd /etc/syslog.conf
 - A demon that implement the logging
 - It is programmed through a configuration file
- openlog, syslog, closelog
 - Procedures to produce event to be logged
- logger
 - User command to produce a log



Syslog





Syslogd: configuration

- Configuration info in `/etc/syslog.conf`
- A text file
 - White lines and those beginning with `#` are ignored
 - ***Selector*** <TAB> ***action***
 - `mail.info` `/var/log/maillog`



Selector

- Identifies
 - The source – the program (`facility`) that is transmitting the message
 - The message `severity level`
- Syntax
 - `facility.level`
 - facility names and severity levels have to be selected in a predefined set



Facility names

Facility

Used by

kern

kernel

user

user process, default

mail

mail system

daemon

System daemons

auth

Security and authorization
related commands

lpr

printer spooling system

news

Usenet news system



Facility names

Facility

Used by

uucp

UUCP

cron

cron daemon

mark

Timestamps produced with a fixed frequency

local0-7

local message

syslog

syslog internal messages

authpriv

Private or system messages

ftp

ftp daemon, ftpd

*

further facilities



Severity level

Level

That means approx.



emerg (panic)

Panic situation

alert

Urgent situation

crit

Critical condition

err

other errors

warning

warning

notice

worth an analysis

info

info

debug

debugging info



Selector

- Several facilities separated by `' , '`
 - `daemon,auth,mail.level` action
- The composition of several selectors by `' ; '`
 - `daemon.level1; mail.level2` action
- The OR composition of selectors is expressed through `' | '` – **un** a message matches if it matches at least one selector.
- `' * '` or `' none '`, (all or none) can be used



Selector

- The level defines the lowest level of a logged message
 - mail.warning, matches any message from the mail system with a level that is, at least, warning
- 'none' is used to neglect some facilities .
 - *.level1;mail.none action

Any facility, a level not smaller than level1 but neglect the mail facility



Action: message handling

Action

filename

@hostname

@ipaddress

user1, user2,...

*

That means

Append the message to a local file

send the message to hostname

send the message to the node with the specified IP address

write the message on the screen of any of these users if the user is logged

write the message on any screen

The logo for syslog, featuring a stylized 's' composed of overlapping yellow, red, and blue squares, followed by the text 'yslog' in a blue sans-serif font.

syslog

Program	Facility	Levels	Description
amd	auth	err-info	NFS automounter
date	auth	notice	Display and set date
ftpd	daemon	err-debug	ftp daemon
gated	daemon	alert-info	Routing daemon
gopher	daemon	err	Internet info server
halt/reboot	auth	crit	Shutdown programs
login/rlogind	auth	crit-info	Login programs
lpd	lpr	err-info	BSD line printer daemon



syslog

Program	Facility	Levels	Description
named	daemon	err-info	Name sever (DNS)
passwd	auth	err	Password setting programs
sendmail	mail	debug-alert	Mail transport system
rwho	daemon	err-notice	remote who daemon
su	auth	crit, notice	substitute UID prog.
sudo	local2	notice, alert	Limited su program
syslogd	syslog, mark	err-info	internet errors, timestamps



syslog

- `openlog (ident, logopt, facility);`
 - Messages are logged as specified by `logopt`
 - They all begin with `ident`
- `Syslog (priority, message, parameters...);`
 - `message` is sent to syslog, that logs it according to `priority level`
- `close ();`



Logopt

- `LOG_CONS`
Write directly to system console if there is an error while sending to system logger.
- `LOG_NDELAY`
Open the connection immediately (normally, the connection is opened when the first message is logged).
- `LOG_NOWAIT`
Don't wait for child processes that may have been created while logging the message.
- `LOG_ODELAY`
The converse of `LOG_NDELAY`; opening of the connection is delayed until `syslog()` is called. (This is the default, and need not be specified.)
- `LOG_PERROR`
(Not in POSIX.1-2001.) Print to `stderr` as well.
- `LOG_PID`
Include PID with each message.



Blockchain: A probabilistic approach

- It forces information traveling over a network of computers to become more transparent and verifiable using mathematical problems that require significant computational power to solve.
- A potential attacker can corrupt a shared database with false information only if it owns a majority of the computational power of the entire network.
- Blockchain protocols ensure that transactions on a blockchain are valid and never recorded more than once, enabling people to coordinate transactions in a decentralized way without the need to rely on a trusted authority to verify and clear all transactions.



Blockchain: A probabilistic approach -2

- A blockchain is simply a chronological database of transactions recorded by a network of computers
- Each blockchain is encrypted and organized into smaller datasets referred to as “blocks.”
- Every block contains information about some transactions, a reference to the preceding block in the blockchain, as well as an answer to a complex mathematical puzzle, to validate the data associated with that block.
- Every computer in the network stores a copy of the blockchain and it periodically synchronizes with the other ones to make sure that all of them have the same shared database.



Blockchain: A probabilistic approach -3

- To ensure a blockchain only records legitimate transactions the network confirms that new transactions are valid and do not invalidate former transactions.
- A block is appended to the blockchain only if the network computers reach consensus on validity of its transaction
- Consensus is achieved through different voting mechanisms, the most common is Proof of Work, which depends on the amount of processing power donated to the network
- A block added to the blockchain, can no longer be deleted and its transactions can be verified by everyone on the network. It becomes a permanent record that the computers on the network use to coordinate an action or verify an event.



Blockchain: A probabilistic approach -4

- The Proof of Work consensus mechanism requires that certain computers on the network (the “miners”) solve computationally-intensive mathematical puzzles, while others verify that the solution to that puzzle does not correspond to a previous transaction.
- To incentivize miners to invest computational power, the first miner to solve the mathematical problem is rewarded either through the issuance of currency or through transaction fees

Blockchain: A probabilistic approach - 5

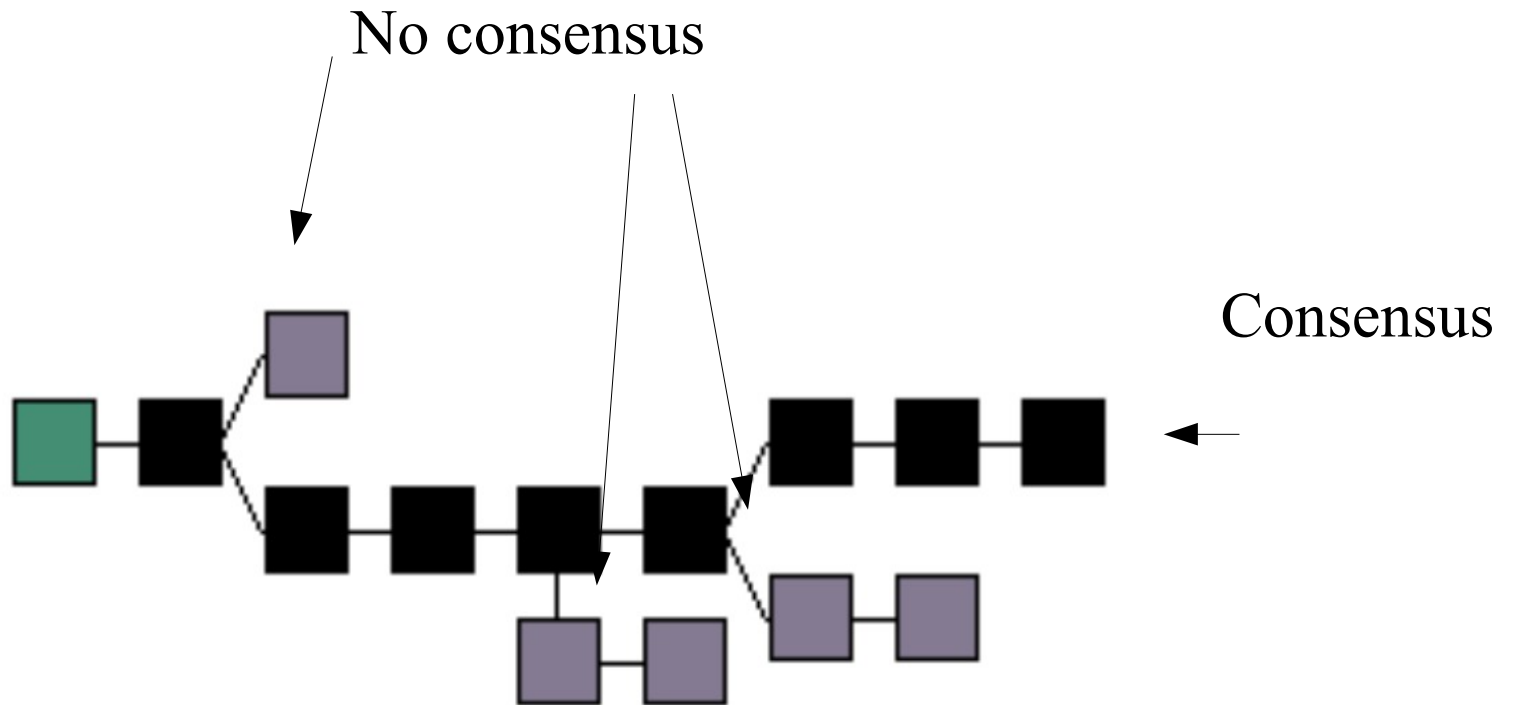


Fig. 1. A graphical representation of the blockchain



Security vs ICT security

- All the principles previously discussed do not fully characterize ICT security
- The two peculiar features of ICT security are
 - Automatic attack
 - The virtual machine hierarchy



Virtual machine hierarchy

- Any ICT system is a hierarchy of virtual machines
- Each virtual machine
 - Defines a set of mechanisms = a programming language
 - The defined mechanism abstracts and hides those of the underlying machine
 - Any machine can be a standard one, with all the consequent implications on vulns



Why ICT security is difficult?

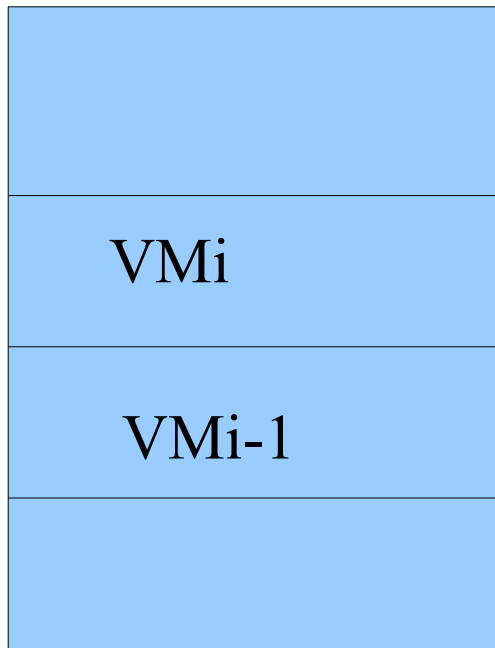
- Vulns may be discovered in the specs and in the implementation of a virtual machine VM
- Vulns cannot be abstracted because a vulnerability in VM results in attacks against any machine of the stack that runs on top of VM
 - ⇔ a vuln in the hardware architecture makes it possible to attack any VM running on it
 - ⇔ a vuln in the OS makes it possible to attack any application it supports



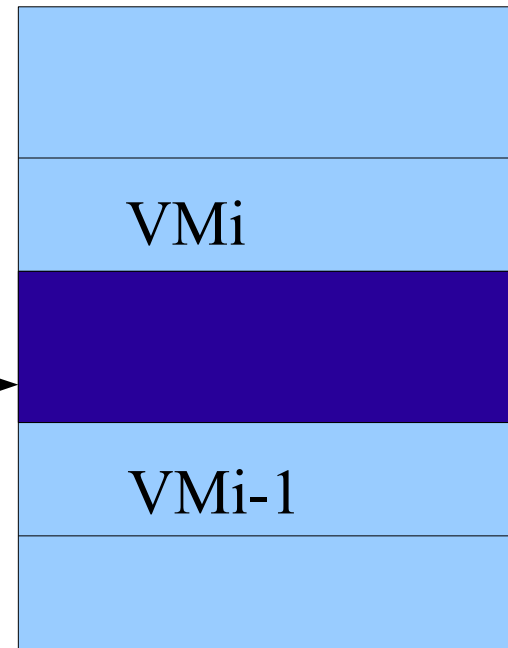
Going down

- A trend in attack is attacking low level virtual machine
- By controlling a low level of the hierarchy any higher level can be attacked
- An interesting attack is the one that inserts a further virtual machine in the hierarchy
 - Difficult to be detected
 - High impact from a security perspective

Blue Pill Attack



New
Virtual
Machine





Blue Pill Attack

The new machine can

- return fake information about the system states to upper layer virtual machines
- transmit to the underlying machines commands that differ from those received by higher VMs
- *Machine in the middle*, a generalization of man in the middle



Hierarchy and robustness - I

- Robustness at any level
 - Each VM should include the checks on the subjects and the objects of the corresponding level
 - The distribution of checks at the various VMs is the simplest way to minimize the overall overhead
 - This also guarantees that the checks of a VM cannot be violated by working at a lower level

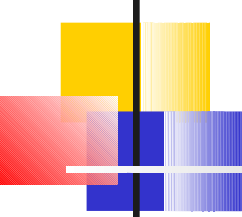
If this strategy is not applied then either

- A VM does not execute any checks

or

- The checks of a VM are delegated to another one but this increases the overall complexity

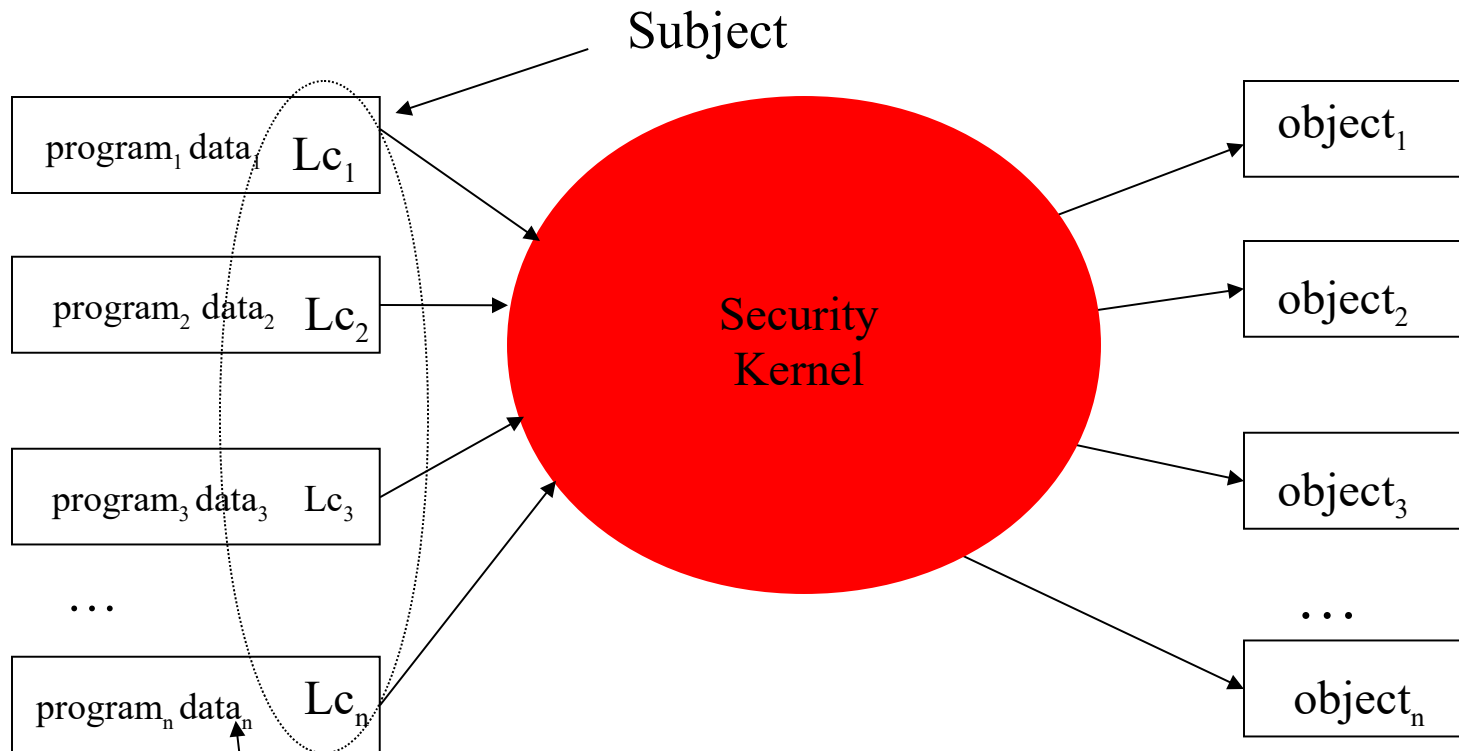
- Redundancy = checks are repeated in distinct VMs



Example - Capability

- VM(L), the machine at level L adopts a capability based solution to manage the rights of a subject
- VM(L-1), the underlying machine at level L-1
 - Implements the subjects and the objects of VM(L)
 - Manages some further objects that implement the capabilities of VM(L)
- The acm of VM(L-1) should guarantee that the subjects of VM(L) cannot manipulate their capabilities

Capability



The implementation of a subject



Hierarchy and robustness - II

- Security policy and mechanism modularity in a hierarchy of VMs:
 - any VM defines a set of mechanisms that may be freely composed by the user of the VM to implement a security policy
 - ⇔
 - Each VM exploits some assumptions on the security of these mechanisms that has to be guaranteed by at least one of the underlying VMs
 - Example: to prevent the manipulation of a capability we can apply
 - Encryption
 - Protection of a memory segment
 - Protection of a data structure
 -
- A distributed implementation of the TCB by several VMs



Hierarchy and robustness - III

- The robustness of a VM is a function of the robustness of the underlying VMs
- Even functionally equivalent machines have a very different robustness because of
 - The implementation of the machine
 - The implementation of the underlying machines

Robustness does not agree with abstraction \Rightarrow

Robustness can be evaluated only in terms of the implementation



A common problem: example

- A memory area, in some memory of VMi is shared among several applications by distinct users of a VMi+k
- The applications that share the area may be not know in advance because they depend upon the users that are sharing VMi
- An application that can access an area can read in it some values left by another application or by another user
- This shows why cloud security is a big problem



A common problem: example

- A memory area, in some memory of VMi is shared among several applications by distinct users of a VMi+k
- The applications that share the area may be not know in advance because they depend upon the users that are sharing VMi
- An application that can access an area can read in it some values left by another application or by another user
- This shows why cloud security is a big problem



Solution

- Any memory area that is either
 - released by an application or
 - garbage collected

has to be reinitialed to avoid any illegal information flow between two applications
(covert channel)

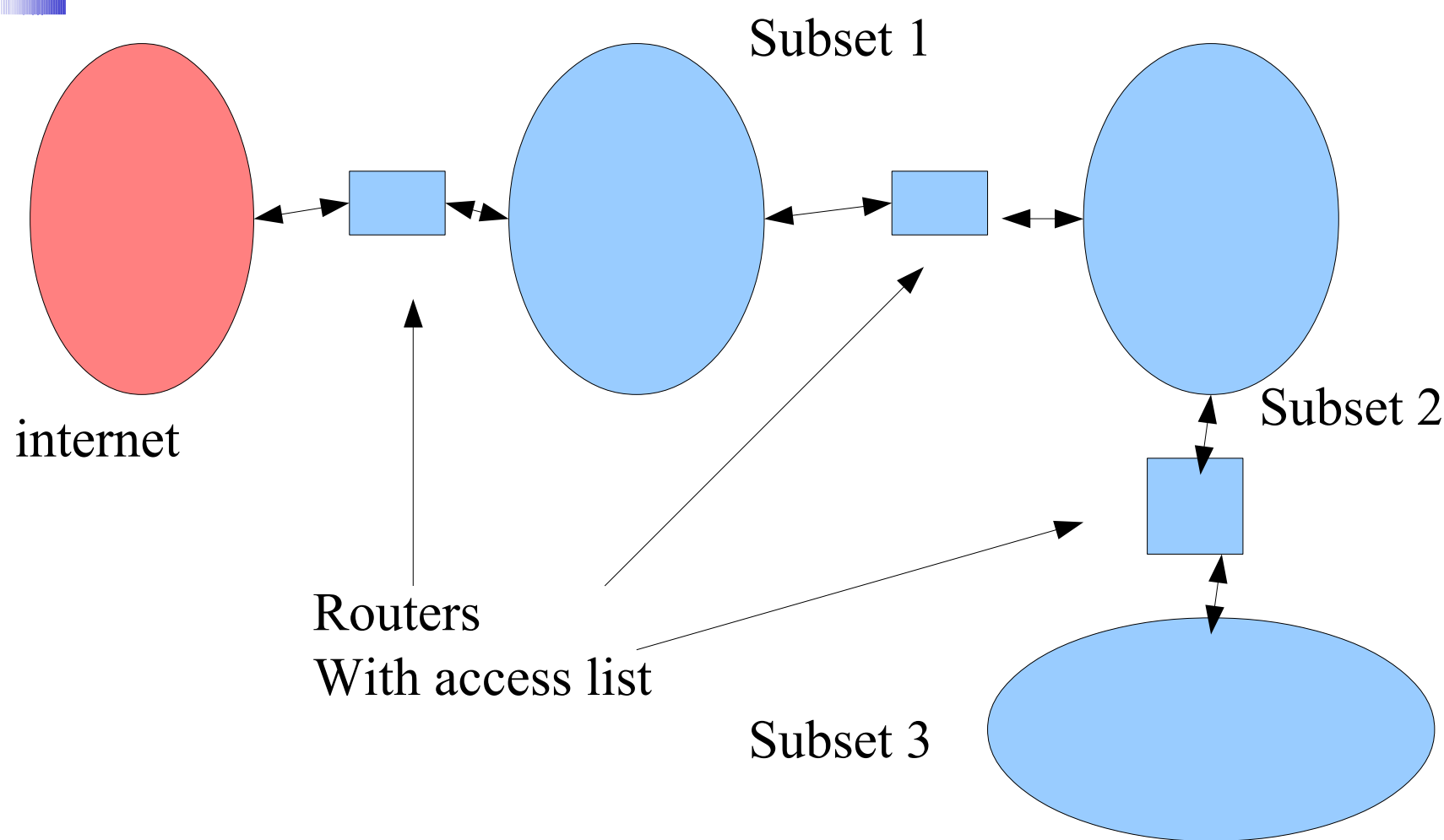
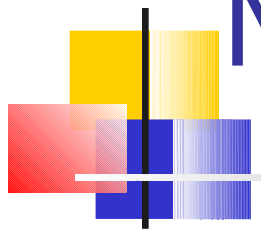
- This holds for any memory area
 - cache,
 - main memory,
 - secondary storage



Solution

- In a system with severe security requirements, all the resources are partitioned into pools each with a distinct level
- The resource in a pool with a given level are shared only among applications run by users with the same security level
- Sharing is constrained to minimize unanticipated flow of information among applications with distinct security levels

Network separation





Network Separation

- Each subnetwork include a subset of the node and it is flat = any node in a subset can interact with another node in the same subnet
- The router access list determines the visibility and interaction rules among subset = which nodes in a subset can interact with nodes in a neighbour subset
- Usually direct interaction among nodes in disjoint subset is forbidden



A general principle ...

- The previous example shows that sharing should be avoided or at least minimized to improve the security of a system
- A secure system
 - is as simple as possible
 - avoids sharing as much as possible
- This explains why a secure system is more expensive of a less secure one



Examples

- Memory segments are partitioned into subsets, each paired with a security level
- Traffic segregation = network channels are partitioned into subset, security critical information is transmitted only along some lines
 - Switches with routing rules and ACLs rather than hubs
 - Partitioning of virtual lines created by tagging or by encryption
 - Distinct transmission frequencies but low security
- It is important to understand that any system manages at least two level of information



Two security levels

- User information
- Information to implement the security policy
- Distinct mechanisms have to be applied to protect the two kinds of information



Example

- A sniffer on a communication line reads any information transmitted along the line
- If a user information is transmitted the sniffer can read the information
- If a user password is transmitted and read by the sniffer then all the user information is lost



Sharing and Cloud

- We have already seen that cloud architectures result in large savings because they are based upon pools of resources shared among users
- Elasticity = when a resource is not used it can be granted to any user that requires it
- What happens when a resource passes from one user to another one?