Protection

Amir H. Payberah
amir@sics.se

Amirkabir University of Technology
(Tehran Polytechnic)

Introduction

Protection

- In one protection model, a computer consists of a collection of objects, hardware or software.

Protection

- In one protection model, a computer consists of a collection of objects, hardware or software.

- Each object has a unique name and can be accessed through a well-defined set of operations.

Protection

- In one protection model, a computer consists of a collection of objects, hardware or software.
- Each object has a unique name and can be accessed through a well-defined set of operations.
- Protection problem: ensure that each object is accessed correctly and only by those processes that are allowed to do so.

Policy vs. Mechanism

- Policies decide what will be done.
  - Policies are likely to change from place to place or time to time.

Policy vs. Mechanism

- Policies decide what will be done.
  - Policies are likely to change from place to place or time to time.

- Mechanisms determine how something will be done.

Policy vs. Mechanism

- Policies decide what will be done.
  - Policies are likely to change from place to place or time to time.
- Mechanisms determine how something will be done.
- The separation of policy and mechanism: flexibility

Principles of Protection

- Principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks.

Principles of Protection

- Principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks.

- Limits damage if entity has a bug.

Principles of Protection

- Principle of least privilege
  - Programs, users and systems should be given just enough privileges to perform their tasks.
- Limits damage if entity has a bug.
- Fine-grained management
  - More complex
  - More overhead
  - More protective

Domain of Protection

Domain of Protection

- The need-to-know principle: at any time, a process should be able to access only those resources that it currently requires to complete its task.

Domain of Protection

- The need-to-know principle: at any time, a process should be able to access only those resources that it currently requires to complete its task.
- Limiting the amount of damage a faulty process can cause in the system.

Domain of Protection Example (1/2)

- Assume a process p invokes procedure $A()$.

Domain of Protection Example (1/2)

- Assume a process p invokes procedure $A()$.
- The procedure should be allowed to access only its own variables and the parameters passed to it.

Domain of Protection Example (1/2)

- Assume a process p invokes procedure $A()$.
- The procedure should be allowed to access only its own variables and the parameters passed to it.
- The procedure should not be able to access all the variables of process p.

Domain of Protection Example (2/2)

- Assume a process p invokes a compiler to compile a particular file.

Domain of Protection Example (2/2)

- Assume a process p invokes a compiler to compile a particular file.
- The compiler should not be able to access files arbitrarily.

Domain of Protection Example (2/2)

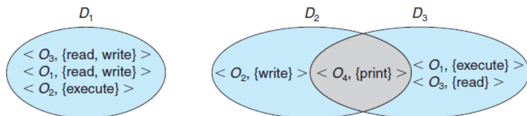
- Assume a process p invokes a compiler to compile a particular file.
- The compiler should not be able to access files arbitrarily.
- The compiler should have access only to a well-defined subset of files (such as the source file, listing file, and so on) related to the file to be compiled.

Domain of Protection Example (2/2)

- Assume a process p invokes a compiler to compile a particular file.
- The compiler should not be able to access files arbitrarily.
- The compiler should have access only to a well-defined subset of files (such as the source file, listing file, and so on) related to the file to be compiled.
- The compiler may have private files used for accounting or optimization purposes that process p should not be able to access.

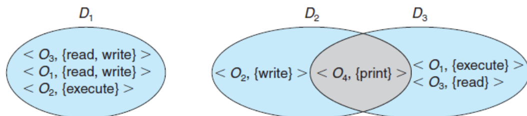
Domain Structure

- Access-right = ⟨object-name, rights-set⟩ where rights-set is a subset of all valid operations that can be performed on the object.
  - E.g., domain D_2 has the access right $\langle O_2, \{write\} \rangle$



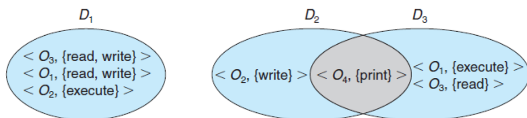
Domain Structure

- Access-right $= \langle$object-name, rights-set$\rangle$ where rights-set is a subset of all valid operations that can be performed on the object.
  - E.g., domain D_2 has the access right $\langle O_2, \{write\}\rangle$
- Domain $=$ set of access-rights



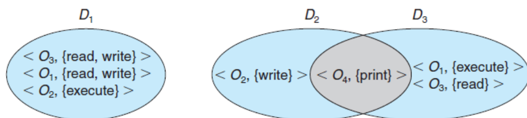
Domain Structure

- Access-right = ⟨object-name, rights-set⟩ where rights-set is a subset of all valid operations that can be performed on the object.
  - E.g., domain D_2 has the access right $\langle O_2, \{write\} \rangle$
- Domain = set of access-rights
- Can be static (during life of system, during life of process), or dynamic (changed by process as needed).



Domain Structure

- Access-right = ⟨object-name, rights-set⟩ where rights-set is a subset of all valid operations that can be performed on the object.
  - E.g., domain D_2 has the access right $\langle O_2, \{write\}\rangle$
- Domain = set of access-rights
- Can be static (during life of system, during life of process), or dynamic (changed by process as needed).
- Domain switching



Protection Domains

- Each user may be a domain

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.
- Each process may be a domain

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.
- Each process may be a domain
  - The set of accessible objects depends on the identity of the process.

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.
- Each process may be a domain
  - The set of accessible objects depends on the identity of the process.
  - Domain switching when one process sends a message to another process and then waits for a response.

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.
- Each process may be a domain
  - The set of accessible objects depends on the identity of the process.
  - Domain switching when one process sends a message to another process and then waits for a response.
- Each procedure may be a domain

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.
- Each process may be a domain
  - The set of accessible objects depends on the identity of the process.
  - Domain switching when one process sends a message to another process and then waits for a response.
- Each procedure may be a domain
  - The set of accessible objects corresponds to the local variables defined within the procedure.

Protection Domains

- Each user may be a domain
  - The set of accessible objects depends on the identity of the user.
  - Domain switching in users logout/login.
- Each process may be a domain
  - The set of accessible objects depends on the identity of the process.
  - Domain switching when one process sends a message to another process and then waits for a response.
- Each procedure may be a domain
  - The set of accessible objects corresponds to the local variables defined within the procedure.
  - Domain switching when a procedure call is made.

UNIX Domain Implementation (1/2)

- Domain = user-id

UNIX Domain Implementation (1/2)

- Domain = user-id
- Domain switching corresponds to changing the user id:
  - via file system
  - via passwords
  - via commands

UNIX Domain Implementation (1/2)

- Domain = user-id
- Domain switching corresponds to changing the user id:
  - via file system
  - via passwords
  - via commands
- Domain switching via file system:

UNIX Domain Implementation (1/2)

- Domain = user-id

- Domain switching corresponds to changing the user id:
  - via file system
  - via passwords
  - via commands

- Domain switching via file system:
  - Each file has associated with it a domain bit (`setuid` bit)

UNIX Domain Implementation (1/2)

- Domain = user-id

- Domain switching corresponds to changing the user id:
  - via file system
  - via passwords
  - via commands

- Domain switching via file system:
  - Each file has associated with it a domain bit (`setuid` bit)
  - If the `setuid` bit is on, and a user executes that file, the userID is set to that of the owner of the file.

UNIX Domain Implementation (1/2)

- Domain = user-id

- Domain switching corresponds to changing the user id:
  - via file system
  - via passwords
  - via commands

- Domain switching via file system:
  - Each file has associated with it a domain bit (`setuid` bit)
  - If the `setuid` bit is on, and a user executes that file, the userID is set to that of the owner of the file.
  - If the `setuid` bit is off, the userID does not change.

UNIX Domain Implementation (2/2)

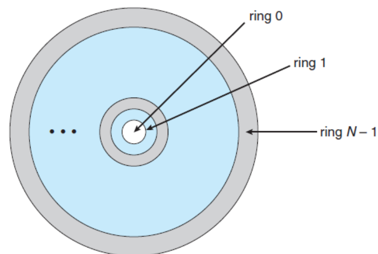
- Domain switching via passwords:
  - `su` command temporarily switches to another user's domain when other domain's password provided

UNIX Domain Implementation (2/2)

- Domain switching via passwords:
  - `su` command temporarily switches to another user's domain when other domain's password provided
- Domain switching via commands:
  - `sudo` command prefix executes specified command in another domain.

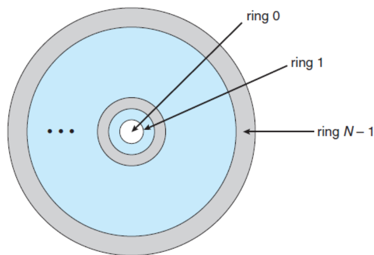
MULTICS Domain Implementation

- Let D_i and D_j be any two domain rings.



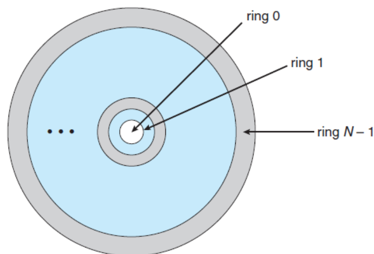
MULTICS Domain Implementation

- Let D_i and D_j be any two domain rings.
- A process in D_i can only access segments associated with domains j, where $(j \geq i)$.



MULTICS Domain Implementation

- Let D_i and D_j be any two domain rings.
- A process in D_i can only access segments associated with domains j, where ($j \geq i$).
- Domain switching when a process crosses from one ring to another by calling a procedure in a different ring.



MULTICS Limitations

- Fairly complex $\rightarrow$ more overhead
- It does not allow strict need-to-know
  - Object accessible in D_j but not in D_i, then j must be $< i$.
  - But then every segment accessible in D_i also accessible in D_j.

Access Matrix

Access Matrix

- View protection as a access matrix.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix

- View protection as a access matrix.
- Rows represent domains.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix

- View protection as a access matrix.
- Rows represent domains.
- Columns represent objects.

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix

- View protection as a access matrix.
- Rows represent domains.
- Columns represent objects.
- $access(i, j)$ is the set of operations that a process executing in $domain_i$ can invoke on $object_j$.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Use of Access Matrix

- If a process in domain D_i tries to do an operation on object O_j, then the operation must be in the access matrix.

- User who creates object can define access column for that object.

- For a new object O_j, the column O_j is added to the access matrix.

Access Matrix With Domains As Objects

- `switch` operation: switching a process from one domain to another.

Access Matrix With Domains As Objects

- `switch` operation: switching a process from one domain to another.
- Including domains among the objects of the access matrix.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix With Domains As Objects

- `switch` operation: switching a process from one domain to another.
- Including domains among the objects of the access matrix.
- Switching from domain D_i to domain D_j is allowed if and only if the access right `switch` $\in access(i, j)$.

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch
D_3		read	execute					
D_4	read write		read write		switch			

Access Matrix Operations

- Allowing controlled change in the contents of the access-matrix entries requires three additional operations:
  - `copy`: applicable to an object
  - `owner`: applicable to an object
  - `control`: applicable to domain object

Access Matrix copy Operation

- With the copy right, a domain can copy its access right to another domain.
- Denoted by an asterisk (*) appended to the access right.

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix owner Operation

- With the owner right, a process in domain D_i can add and remove any right in any entry in column j.

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix control Operation

- If $access(i, j)$ includes the control right, then a process in domain D_i can remove any access right from row j.

object / domain	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

Access Matrix Mechanism and Policy

- Access matrix design separates mechanism from policy.

Access Matrix Mechanism and Policy

- Access matrix design separates mechanism from policy.
- Mechanism
  - OS provides access-matrix + rules
  - It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.

Access Matrix Mechanism and Policy

- Access matrix design separates mechanism from policy.
- Mechanism
  - OS provides access-matrix + rules
  - It ensures that the matrix is only manipulated by authorized agents and that rules are strictly enforced.
- Policy
  - User dictates policy.
  - Who can access what object and in what mode.

Implementation of Access Matrix

Implementation of Access Matrix

- In general, the access matrix is sparse: most of the entries will be empty.

Implementation of Access Matrix

- In general, the access matrix is sparse: most of the entries will be empty.
- Option 1: Global table
- Option 2: Access lists for objects
- Option 3: Capability list for domains
- Option 4: Lock-key

Option 1 - Global Table

- Store ordered triples $\langle domain, object, rights_set \rangle$ in table.

Option 1 - Global Table

- Store ordered triples $\langle domain, object, rights_set \rangle$ in table.
- A requested operation M on object O_j within domain D_i $\rightarrow$ search table for $\langle D_i, O_j, R_k \rangle$.
  - with $M \in R_k$

Option 1 - Global Table

- Store ordered triples $\langle domain, object, rights_set \rangle$ in table.
- A requested operation M on object O_j within domain D_i $\rightarrow$ search table for $\langle D_i, O_j, R_k \rangle$.
  - with $M \in R_k$
- But table could be large $\rightarrow$ won't fit in main memory.

Option 1 - Global Table

- Store ordered triples $\langle domain, object, rights_set \rangle$ in table.
- A requested operation M on object O_j within domain D_i $\rightarrow$ search table for $\langle D_i, O_j, R_k \rangle$.
    - with $M \in R_k$
- But table could be large $\rightarrow$ won't fit in main memory.
- Difficult to group objects, e.g., consider an object that all domains can read.

Option 2 - Access Lists For Objects

- Each column implemented as an access list for one object.

Option 2 - Access Lists For Objects

- Each column implemented as an access list for one object.

- Per-object list consists of ordered pairs $\langle domain, rights_set \rangle$, defining all domains with non-empty set of access rights for the object.

Option 2 - Access Lists For Objects

- Each column implemented as an access list for one object.
- Per-object list consists of ordered pairs $\langle domain, rights_set \rangle$, defining all domains with non-empty set of access rights for the object.
- Easily extended to contain default set $\rightarrow$ if $M \in$ default set, also allow access.

Option 3 - Capability List For Domains (1/2)

- Instead of object-based, list is domain-based.

Option 3 - Capability List For Domains (1/2)

- Instead of object-based, list is domain-based.
- Capability list for domain is list of objects together with operations allows on them.

Option 3 - Capability List For Domains (1/2)

- Instead of object-based, list is domain-based.
- Capability list for domain is list of objects together with operations allows on them.
- Object represented by its name or address, called a capability.

Option 3 - Capability List For Domains (1/2)

- Instead of object-based, list is domain-based.
- Capability list for domain is list of objects together with operations allows on them.
- Object represented by its name or address, called a capability.
- To execute operation M on object O_j, a process requests operation and specifies capability as parameter.
  - Possession of capability means access is allowed

Option 3 - Capability List For Domains (2/2)

- Capability list associated with domain, but never directly accessible by domain.
  - Rather, protected object, maintained by OS and accessed indirectly
  - Like a secure pointer

Option 4 - Lock-Key

- Compromise between access lists and capability lists.

Option 4 - Lock-Key

- Compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called locks.

Option 4 - Lock-Key

- Compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called locks.
- Each domain has a list of unique bit patterns called keys.

Option 4 - Lock-Key

- Compromise between access lists and capability lists.
- Each object has a list of unique bit patterns, called locks.
- Each domain has a list of unique bit patterns called keys.
- Process in a domain can only access an object, if the domain has key that matches one of the locks.

Comparison of Implementations (1/3)

- Many trade-offs to consider.

Comparison of Implementations (1/3)

- Many trade-offs to consider.
- Global table is simple, but can be large.

Comparison of Implementations (1/3)

- Many trade-offs to consider.
- Global table is simple, but can be large.
- Access lists correspond to needs of users
  - Because access-right info for a domain is not localized, determining the set of access rights for each domain is difficult.
  - Every access to an object must be checked: Many objects and access rights $\rightarrow$ slow

Comparison of Implementations (2/3)

- Capability list is useful for localizing info for a given process.
  - But revocation capabilities can be inefficient.

Comparison of Implementations (2/3)

- Capability list is useful for localizing info for a given process.
  - But revocation capabilities can be inefficient.

- Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

Comparison of Implementations (3/3)

- Most systems use combination of access lists and capabilities.

Comparison of Implementations (3/3)

- Most systems use combination of access lists and capabilities.
- First access to an object $\rightarrow$ access list searched.
  - If allowed, capability created and attached to process: additional accesses need not be checked
  - After last access, capability destroyed.

Access Control

Access Control (1/2)

- Oracle Solaris 10 provides role-based access control (RBAC) to implement least privilege.

Access Control (1/2)

- Oracle Solaris 10 provides role-based access control (RBAC) to implement least privilege.
- Privilege is a right to execute a system call or use an option within a system call.

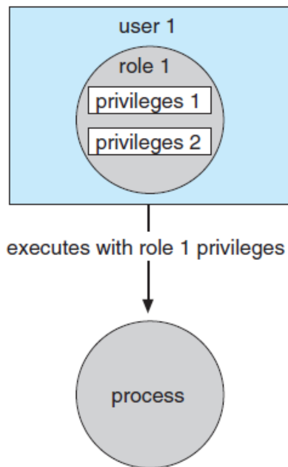
Access Control (1/2)

- Oracle Solaris 10 provides role-based access control (RBAC) to implement least privilege.
- Privilege is a right to execute a system call or use an option within a system call.
- Can be assigned to processes.

Access Control (1/2)

- Oracle Solaris 10 provides role-based access control (RBAC) to implement least privilege.
- Privilege is a right to execute a system call or use an option within a system call.
- Can be assigned to processes.
- Users are assigned roles granting access to privileges and programs: enable role via password to gain its privileges.

Access Control (2/2)



Revocation of Access Rights

Revocation of Access Rights (1/2)

- Various options to remove the access right of a domain to an object.

Revocation of Access Rights (1/2)

- Various options to remove the access right of a domain to an object.
- Immediate vs. delayed
  - If delayed, can we find out when it will take place?

Revocation of Access Rights (1/2)

- Various options to remove the access right of a domain to an object.
- Immediate vs. delayed
  - If delayed, can we find out when it will take place?
- Selective vs. general
  - Affect all the users who have access right to that object, or just a selected group of users?

Revocation of Access Rights (2/2)

- Partial vs. total
  - Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?

Revocation of Access Rights (2/2)

- Partial vs. total
  - Can a subset of the rights associated with an object be revoked, or must we revoke all access rights for this object?
- Temporary vs. permanent
  - Can access be revoked permanently, or can access be revoked and later be obtained again?

Access List

- Delete access rights from access list
- Simple: search the access list and remove entry
- Immediate, general or selective, total or partial, permanent or temporary.

Capability List (1/5)

- More difficult: because the capabilities are distributed throughout the system.
- Scheme required to locate capability in the system before capability can be revoked:
  - Reacquisition
  - Back-pointers
  - Indirection
  - Keys

Capability List (2/5)

- Reacquisition: periodically, capabilities are deleted from each domain.
- If a process needs a capability, it may try to reacquire the capability.
- If access has been revoked, the process will not be able to reacquire the capability.

Capability List (3/5)

- Back-pointers: set of pointers from each object to all capabilities of that object.
- When revocation is required, we can follow these pointers, changing the capabilities as necessary.
- Its implementation is costly.

Capability List (4/5)

- Indirection: capability points to global table entry, which points to the object.
- Delete entry from global table.
- Not selective.

Capability List (5/5)

- Keys: unique bits associated with capability, generated when capability created.
- Master key associated with object, key matches master key for access.
- Revocation: create new master key
- Policy decision of who can create and modify keys - object owner or others?

Language-Based Protection

Language-Based Protection

- Protection is usually achieved through an OS kernel: high overhead

Language-Based Protection

- Protection is usually achieved through an OS kernel: high overhead
- OSs have become more complex.

Language-Based Protection

- Protection is usually achieved through an OS kernel: high overhead

- OSs have become more complex.
  - Concern for the function to be invoked extends beyond a set of system-defined functions, such as standard file-access methods, to include user-defined functions as well.

Language-Based Protection

- Protection is usually achieved through an OS kernel: high overhead

- OSs have become more complex.
  - Concern for the function to be invoked extends beyond a set of system-defined functions, such as standard file-access methods, to include user-defined functions as well.

- Policies for resource use may also change over time.
  - So, protection should be available as a tool for use by the application designer.

Compiler-Based Enforcement

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources

- Language implementation can provide software for protection enforcement when automatic hardware-supported checking is unavailable.

Protection in Java

- Protection is handled by the Java Virtual Machine (JVM).
- A class is assigned a protection domain when it is loaded by the JVM.
- The protection domain indicates what operations the class can (and cannot) perform.

Summary

Summary

- Protection problem

Summary

- Protection problem
- Principle of least privilege

Summary

- Protection problem
- Principle of least privilege
- The need-to-know principle:

Summary

- Protection problem
- Principle of least privilege
- The need-to-know principle:
- Domain structure: user, process, procedure

Summary

- Protection problem
- Principle of least privilege
- The need-to-know principle:
- Domain structure: user, process, procedure
- Access matrix: switch, copy, owner, control

Summary

- Protection problem
- Principle of least privilege
- The need-to-know principle:
- Domain structure: user, process, procedure
- Access matrix: switch, copy, owner, control
- Access matrix implementation: global table, access lists for objects, capability list for domains, lock-key

Summary

- Protection problem
- Principle of least privilege
- The need-to-know principle:
- Domain structure: user, process, procedure
- Access matrix: switch, copy, owner, control
- Access matrix implementation: global table, access lists for objects, capability list for domains, lock-key
- Revocation of access rights

Questions?