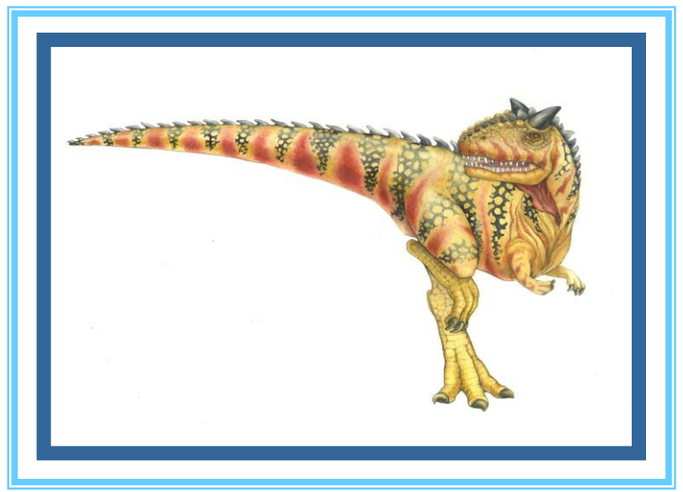
Chapter 14: Protection

هذا التشابتر يتكلم عن حماية النظام (Protection) داخل أنظمة التشغيل، وكيف النظام يتحكم في صلاحيات المستخدمين والبرامج بحيث يمنع الوصول غير المسموح للملفات أو الذاكرة أو الأجهزة.





Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS

Protection: التحكم في صلاحيات الوصول داخل النظام
Security: حماية النظام من الهجمات الخارجية والاختراقات

- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service

الفيروسات
الاختراقات
سرقة الحسابات
هجمات تعطيل الخدمة (DoS)

- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** (increase) allows user to change to effective ID with more rights

كيف النظام يفرق بين المستخدمين؟
النظام لازم يعرف من هذا المستخدم؟ وما صلاحياته؟

User ID (UID)
لكل مستخدم: اسم، رقم تعريف
استخدام الـ **UID**
الـ **OS** يربط الـ **UID** مع: الملفات، العمليات **(Processes)**

Group ID (GID)
أحيانًا نحتاج مجموعة مستخدمين لهم نفس الصلاحيات.
لذلك يوجد:
Group ID
مثال: **Students Group، Teachers Group**

Privilege Escalation
رفع الصلاحيات
يعني المستخدم يغيّر هويته الفعالة للحصول على صلاحيات أعلى.





Goals of Protection أهداف الحماية

- In one protection model, 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

النظام عبارة عن:
Collection of Objects
يعني مجموعة Objects:
ملفات
ذاكرة
CPU
Printers
Programs
كلها تعتبر Objects.
كل Object له:
1) اسم فريد
Unique Name
2) عمليات مسموحة
Well-defined operations

Protection Problem
النظام يجب أن يضمن:
كل Object يتم الوصول له بشكل صحيح ومن العمليات المسموح لها فقط.
يعني:
منع الوصول غير المصرح
منع التعديل غير المسموح
منع البرامج من تخريب بعضها





Principles of Protection

- Guiding principle – **principle of least privilege** كل **User** أو **Program** يأخذ أقل صلاحيات يحتاجها فقط.
  - Programs, users and systems should be given just enough privileges to perform their tasks
  - Limits damage if entity has a bug, gets abused
  - Can be static (during life of system, during life of process)
  - Or dynamic (changed by process as needed) – **domain switching**, **privilege escalation**
  - "Need to know" a similar concept regarding access to data

    مفهوم مشابه يعني المستخدم يحصل فقط على المعلومات التي يحتاجها.

إذا البرنامج فيه **Bug** أو تم اختراقه
فالضرر يكون محدود

أنواع الـ **Least Privilege**
1) Static
صلاحيات ثابتة
تبقى طول تشغيل النظام أو العملية.
2) Dynamic
صلاحيات تتغير عند الحاجة
مثل:
Domain Switching
Privilege Escalation
العملية ترفع أو تقلل صلاحياتها حسب الحاجة.

- Must consider "grain" aspect
  - Rough-grained privilege management easier, simpler, but least privilege now done in large chunks
    - For example, traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management more complex, more overhead, but more protective
    - File ACL lists, RBAC

درجة دقة التحكم بالصلاحيات
يوجد نوعان:
1) Rough-Grained
تحكم عام وبسيط، أسهل وأسرع.
لكن الصلاحيات تكون كبيرة.
مثال **Unix** التقليدي
2) Fine-Grained
تحكم دقيق جداً
أكثر أماناً لكن أعقد.
أمثلة
ACL
Access Control List
RBAC
Role-Based Access Control

- Domain can be user, process, procedure



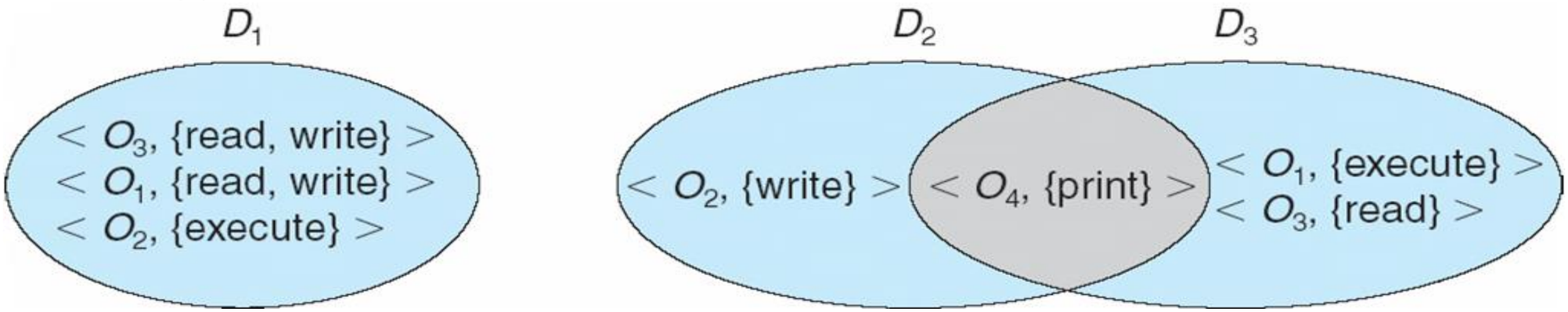
Domain Structure

- Access-right = <*object-name*, *rights-set*>
  where *rights-set* is a subset of all valid operations that can be performed on the object

- Domain = set of access-rights

ما هو الـ Access-right؟
الصيغة هي:
Access-right = <object-name, rights-set>
يعني:
اسم المورد أو الشيء المراد الوصول له = object-name
مجموعة العمليات المسموح بها عليه = rights-set
مثال:
<file1, {read, write}>
يعني هذا الحق يسمح بالقراءة والكتابة على file1.

ما هو الـ Domain؟
Domain = set of access-rights
يعني الـ Domain هو:
مجموعة من الصلاحيات المسموحة لعملية أو مستخدم معين.
مثال:
لمستخدم معيّن قد يحتوي على Domain:
<file1, read>
<file2, read, write>
<printer1, print>
إذن هذا الـ Domain يحدد ماذا يستطيع أن يفعل هذا المستخدم أو الـ process.





Domain Implementation (UNIX)

هذا الجزء يشرح كيف تُنفّذ الـ domains في Unix

- Domain = user-id

  كل مستخدم له domain خاص فيه، والـ OS يقرر الصلاحيات بناءً على الـ user ID

- Domain switch accomplished via file system

  أحيانًا النظام يغيّر الـ domain أثناء تنفيذ ملف معين.
  كيف؟ بواسطة setuid bit
  كل ملف يمكن أن يكون معه bit خاص اسمه domain bit أو setuid bit.

  - Each file has associated with it a domain bit (setuid bit)
  - When file is executed and setuid = on, then user-id is set to owner of the file being executed
  - When execution completes user-id is reset

  ماذا يفعل setuid؟
  إذا كان الملف setuid = on
  ثم تم تشغيله، فإن:
  الـ user-id يصبح مؤقتًا مالك الملف
  وبعد انتهاء التنفيذ يرجع الـ user-id الأصلي

- Domain switch accomplished via passwords

  - `su` command temporarily switches to another user's domain when other domain's password provided

    هذا الأمر يسمح بالانتقال مؤقتًا إلى domain مستخدم آخر إذا تم إدخال كلمة مروره.

- Domain switching via commands

  - `sudo` command prefix executes specified command in another domain (if original domain has privilege or password given)

    هذا الأمر يشغّل أمرًا معينًا بصلاحيات أعلى.





Access Matrix

- View protection as a matrix (*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 $Object_j$

النظام هنا يتم تمثيله على شكل مصفوفة حماية.
الفكرة
Rows = Domains
Columns = Objects
وفي كل خلية:
Access(i, j) = مجموعة العمليات التي يستطيع Domain i تنفيذها على Object j



Access Matrix (P.538)

شكل توضيحي للـ Access Matrix

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	

Domain D1 يستطيع قراءة file1



Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix
- User who creates object can define access column for that object
- Can be expanded to dynamic protection
  - Operations to add, delete access rights
  - Special access rights:
    - *owner* of O_i
    - *copy* op from O_i to O_j (denoted by "*")
    - *control* – D_i can modify D_j access rights
    - *transfer* – switch from domain D_i to D_j
  - *Copy* and *Owner* applicable to an object
  - *Control* applicable to domain object

القاعدة الأساسية
إذا حاول process داخل Domain Di أن ينفذ العملية op على object Oj،
فإن:
العملية op يجب أن تكون موجودة داخل access matrix
يعني النظام لا يسمح إلا بما هو مسجل في المصفوفة.
من يحدد صلاحيات object الجديد؟
عندما ينشئ مستخدم object جديدًا، فإنه يستطيع تحديد access column الخاصة به.
يعني هو يقرر:
من يقرأ؟
من يكتب؟
من ينفذ؟
5) Dynamic Protection
المصفوفة يمكن أن تدعم حماية متغيرة مع الزمن، وليس فقط صلاحيات ثابتة.
يعني:
إضافة صلاحيات
حذف صلاحيات
نقل صلاحيات
6) Special Access Rights
هذا الجزء مهم جدًا.
1) owner
المالك الخاص بالـ object.
يعني صاحب الملف أو المورد.
مثال:
مالك الملف قد يملك حق تعديل صلاحياته.
2) copy
نسخ الصلاحية من object إلى object آخر.
ويرمز له غالبًا بـ *
الفكرة:
إذا كان لديك حق معين على object، يمكنك نسخه إلى object آخر حسب السياسة المسموحة.
3) control
يعني:
آخر Domain يمكنه تعديل صلاحيات Domain
وهذا ينطبق على domain object
مثال:
قد يسيطر على صلاحيات مستخدمين آخرين Admin.
4) transfer
يعني:
الانتقال من Domain Di إلى Domain Dj
أي تغيير النطاق الذي تعمل فيه العملية.





Use of Access Matrix (Cont.)

- **Access matrix** design separates mechanism from policy
  - Mechanism
    - Operating system provides access-matrix + rules
    - If 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

Mechanism

يعني:

الآلية التي يوفرها نظام التشغيل

النظام يوفر:

access matrix

rules

enforcement

أي أنه يضمن أن:

المصفوفة لا يغيّرها إلا المصرح لهم

القواعد تُطبّق بدقة

Policy

يعني:

القرار نفسه: من يصل إلى ماذا؟

وهذا يحدده المستخدم أو المسؤول.

مثال:

من يقرأ الملف؟

من يكتب؟

من ينفذ؟





Access Matrix of Figure A with Domains as Objects (P.539)

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
D_3		read	execute					
D_4	read write		read write		switch			



Access Matrix with *Copy* Rights

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

(a)

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

(b)



Implementation of Access Matrix

- Generally, a sparse matrix

مصفوفة الوصول نظريًا تكون جميلة، لكن عمليًا غالبًا تكون: **Sparse matrix**
يعني معظم الخانات فارغة، لأن ليس كل **Domain** يملك صلاحية على كل **Object**.
لذلك لا نخزّنها كجدول كامل، بل نستخدم طرق مختلفة.

- Option 1 – Global table
  - Store ordered triples *< domain, object, rights-set >* in table
  - A requested operation M on object O_j within domain D_i -> search table for $< D_i, O_j, R_k >$
    - with $M \in R_k$
  - But table could be large -> won't fit in main memory
  - Difficult to group objects (consider an object that all domains can read)

Option 1: Global Table
نخزن الصلاحيات كلها في جدول واحد على شكل:
<domain, object, rights-set>
المشكلة:
الجدول قد يصبح كبير جدًا
قد لا يناسب الذاكرة الرئيسية
صعب التعامل مع **objects** المشتركة بين الجميع

- Option 2 – Access lists for objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs *< domain, rights-set >* defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If M ∈ default set, also allow access

2) Access Lists for Objects
هنا ننظر إلى كل **object** وحده.
يعني كل **column** في المصفوفة تتحول إلى **Access Control List (ACL)**



continue

- Each column = Access-control list for one object
  Defines who can perform what operation

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each Row = Capability List (like a key)
  For each domain, what operations allowed on what objects

  Object F1 – Read
  Object F4 – Read, Write, Execute
  Object F5 – Read, Write, Delete, Copy

ما هي ACL؟
هي قائمة تحدد:
من يستطيع فعل ماذا على هذا object
مثال:
Object = file1
D1 -> read, write
D2 -> read
D3 -> read
يعني كل domain أو user مرتبط بصلاحياته على هذا الملف.
ميزة مهمة
يمكن إضافة default set.
يعني:
إذا كانت هناك صلاحيات افتراضية، وأي M موجودة فيها، يُسمح بالوصول تلقائيًا.





Implementation of Access Matrix (Cont.)

- Option 3 – Capability list for domains
  - 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
  - Execute operation M on object O_j, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

3) Capability List for Domains
هنا بدل ما نفكر في object، نفكر في domain.
يعني:
كل row في المصفوفة تتحول إلى Capability List
ما هي Capability؟
هي مثل:
مفتاح أو تصريح يثبت أن العملية تملك حق الوصول
Capability = object + allowed operations
مثال:
Object F1 -> Read
Object F4 -> Read, Write, Execute
Object F5 -> Read, Write, Delete, Copy
كيف تستخدم؟
إذا أراد process داخل domain أن ينفذ عملية M على object Oj:
يرسل capability كجزء من الطلب
إذا كان يملك capability المناسبة، يسمح له النظام
الفكرة:
Possession of capability means access is allowed
يعني مجرد امتلاك capability يعني عندك صلاحية.
نقطة مهمة جدًا
الـ capability list:
تكون مرتبطة بالـ domain
لكن لا يستطيع الـ domain الوصول إليها مباشرة
النظام نفسه يحميها
لذلك توصف بأنها مثل:
secure pointer

- Option 4 – Lock-key
  - Compromise between access lists and capability lists
  - Each object has list of unique bit patterns, called locks
  - Each domain as list of unique bit patterns called keys
  - Process in a domain can only access object if domain has key that matches one of the locks

4) Lock-Key Scheme
هذا حل وسط بين:
Access lists
Capability lists
كيف يعمل؟
كل object لديه:
مجموعة locks
وكل domain لديه:
مجموعة keys
إذا كان هناك key يطابق lock:
يُسمح بالوصول
الفكرة ببساطة
قفل = object
مفتاح = domain





Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized so difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation (CANCEL) capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

كل طريقة لها مزايا وعيوب.
Global Table
المميزات:
بسيطة جدًا
العيوب:
كبيرة جدًا
بطيئة
غير عملية مع الأنظمة الكبيرة
Access Lists
المميزات:
مناسبة لاحتياجات المستخدمين
سهلة لمعرفة من يملك حق الوصول إلى **object** معين
العيوب:
معرفة صلاحيات **domain** الواحد تصبح صعبة
كل **access** إلى **object** يحتاج فحص
بطيئة إذا كان عدد **objects** كبير
Capability Lists
المميزات:
جيدة في تنظيم صلاحيات كل **process**
تحفظ المعلومات بشكل محلي لـ **domain**
العيوب:
سحب الصلاحيات **revocation** يكون صعبًا أحيانًا
Lock-Key
المميزات:
مرن
يمكن تمرير **keys** بسهولة بين **domains**
سحب الصلاحيات أسهل
العيوب:
يحتاج إدارة دقيقة لـ **keys** والـ **locks**

- Most systems use combination of access lists and capabilities
  - First access to an object -> access list searched
    - If allowed, capability created and attached to process
      - Additional accesses need not be checked
    - After last access, capability destroyed
    - Consider file system with ACLs per file

معظم الأنظمة لا تستخدم طريقة واحدة فقط، بل مزيجًا.
الفكرة:
أول مرة يدخل **process** على **object**:
النظام يفحص الـ **access list**
إذا كان الوصول مسموحًا:
ينشئ **capability** ويربطها بالعملية
بعد ذلك:
لا يحتاج فحصًا كاملًا كل مرة
وإذا انتهى آخر استخدام:
تُحذف **capability**
مثال
في نظام الملفات:
يوجد **ACL** على الملف
وبعد السماح، يمكن إنشاء **capability** مؤقتة لتسريع الوصول





Access Control

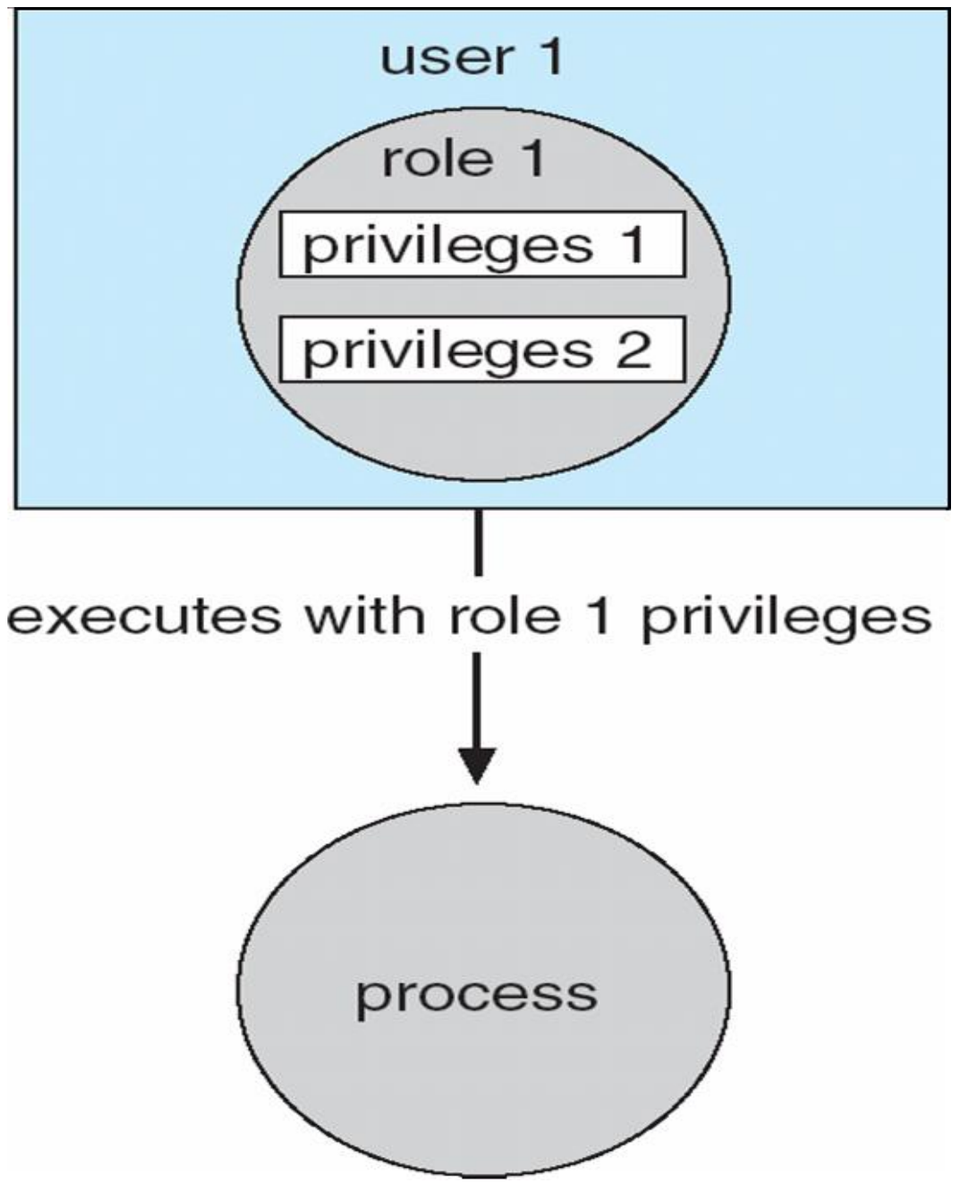
- Protection can be applied to non-file resources
- Solaris 10 provides role-based access control (RBAC) to implement least privilege
  - *Privilege* is right to execute system call or use an option within a system call
  - Can be assigned to processes
  - Users assigned *roles* granting access to privileges and programs
    - Enable role via password to gain its privileges
  - Similar to access matrix

7) Access Control
هذا الجزء يوضح أن الحماية ليست فقط للملفات.
يمكن تطبيقها على أي resource:
processes
devices
system calls
options داخل system call
Role-Based Access Control (RBAC)
في Solaris 10 مثلاً، يوجد RBAC لتطبيق least privilege.
الفكرة:
بدل أن نعطي المستخدم كل الصلاحيات، نعطيه roles.
كل role:
يملك privileges معينة
يمكن تفعيله بكلمة مرور
عند تفعيله يحصل المستخدم على صلاحياته
ما هي Privilege؟
هي:
حق تنفيذ system call معين أو استخدام خيار معين داخل system call
العلاقة مع Access Matrix
:لأنك فعليًا تحددين Access Matrix تشبه RBAC
من له صلاحية
وعلى أي شيء
وبأي شكل





Role-based Access Control in Solaris 10







Revocation of Access Rights

أحيانًا نحتاج سحب الصلاحية.
مثلاً:
موظف خرج من الشركة
يجب إلغاء وصوله للملفات

- Various options to remove the access right of a domain to an object
  - Immediate vs. delayed
  - Selective vs. general
  - Partial vs. total
  - Temporary vs. permanent
- **Access List** – Delete access rights from access list
  - Simple – search access list and remove entry
  - Immediate, general or selective, total or partial, permanent or temporary
- **Capability List** – Scheme required to locate capability in the system before capability can be revoked
  - Reacquisition – periodic delete, with require and denial if revoked
  - Back-pointers – set of pointers from each object to all capabilities of that object (Multics)
  - Indirection – capability points to global table entry which points to object – delete entry from global table, not selective (CAL)
  - 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?

أنواع السحب
1) Immediate vs Delayed
Immediate: تُسحب فورًا
Delayed: تُسحب بعد وقت أو عند حدوث شرط
2) Selective vs General
Selective: سحب صلاحية معينة فقط
General: سحب صلاحيات أوسع
3) Partial vs Total
Partial: سحب جزء من الحقوق
Total: سحب كل الحقوق
4) Temporary vs Permanent
Temporary: مؤقت
Permanent: نهائي

9) Revocation with Access Lists
إذا كانت لدينا **ACL**:
نبحث في القائمة
نحذف **entry** الخاصة بالمستخدم أو الـ **domain**
المميزات: سهل، فوري

10) Revocation with Capability Lists
هنا أصعب، لأن **capability** قد تكون موزعة عند أكثر من **process**.
لذلك توجد عدة حلول:
1) Reacquisition
تُحذف دوريا **capabilities**
ثم يعاد طلبها
إذا كانت الصلاحية أُلغيت، يتم الرفض
2) Back-pointers
كل **object** يحتفظ بمؤشرات إلى كل **capabilities** المرتبطة به.
مثال: إذا الغيت **object**، تستطيع النظام تتبع كل **capabilities** الخاصة به وحذفها.
هذا استُخدم في **Multics**.
3) Indirection
بدل أن تشير **capability** مباشرة إلى **object**، تشير إلى:
entry في **global table**
وهذه **entry** تشير إلى **object**
إذا أردت السحب:
تحذف **entry** من الجدول العالمي
عيبها: ليست **selective** جدًا
4) Keys
كل **capability** لها **key** فريد.
ولكل **object**:
master key
إذا تطابق **key** مع **master key** → **access allowed**.
عند **revocation**:
تولد **master key** جديد
فيصبح المفتاح القديم غير صالح.
لكن هنا يوجد سؤال مهم:
من يملك حق إنشاء وتعديل **keys**؟
المالك؟ أو جهة أخرى؟
وهذا قرار **policy**.





Language-Based Protection

- 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
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system

هنا الحماية تُكتب داخل لغة البرمجة نفسها.
يعني بدل الاعتماد فقط على hardware:
اللغة نفسها تساعد في فرض الحماية
الفكرة
لغة البرمجة تستطيع:
تحديد سياسة الموارد
منع الاستخدام الخاطئ
توليد calls إلى نظام التشغيل لتنفيذ الحماية
متى يكون هذا مفيدًا؟
عندما لا يوجد hardware checking مباشر أو عندما نريد حماية على مستوى أعلى.
مثال
اللغة يمكن أن تفرض:
من يقرأ الذاكرة
من يعدّل كائنًا
من يستدعي method معيّن





Protection in Java 2

- 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
- If a library method is invoked that performs a privileged operation, the stack is inspected to ensure the operation can be performed by the library

Protection in Java 2
في Java 2، الحماية تُدار بواسطة:
JVM
Java Virtual Machine
كيف؟
عندما يتم تحميل class:
يُعطى protection domain
هذا domain يحدد:
ما العمليات المسموحة لهذا الـ class
وما العمليات الممنوعة
إذا استدعت library method عملية حساسة:
النظام يفحص الـ stack
يعني:
ينظر إلى سلسلة الاستدعاءات
ويتأكد أن كل جزء في السلسلة يملك الصلاحية المناسبة
إذا لم يكن كذلك: تُمنع العملية





Stack Inspection (P.554)

protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	...
	get(url);
	open(addr);
	...
		get(URL u):
		...
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		...
			open(Addr a):
			...
			checkPermission (a, connect);
			connect (a);
			...

