Access Control

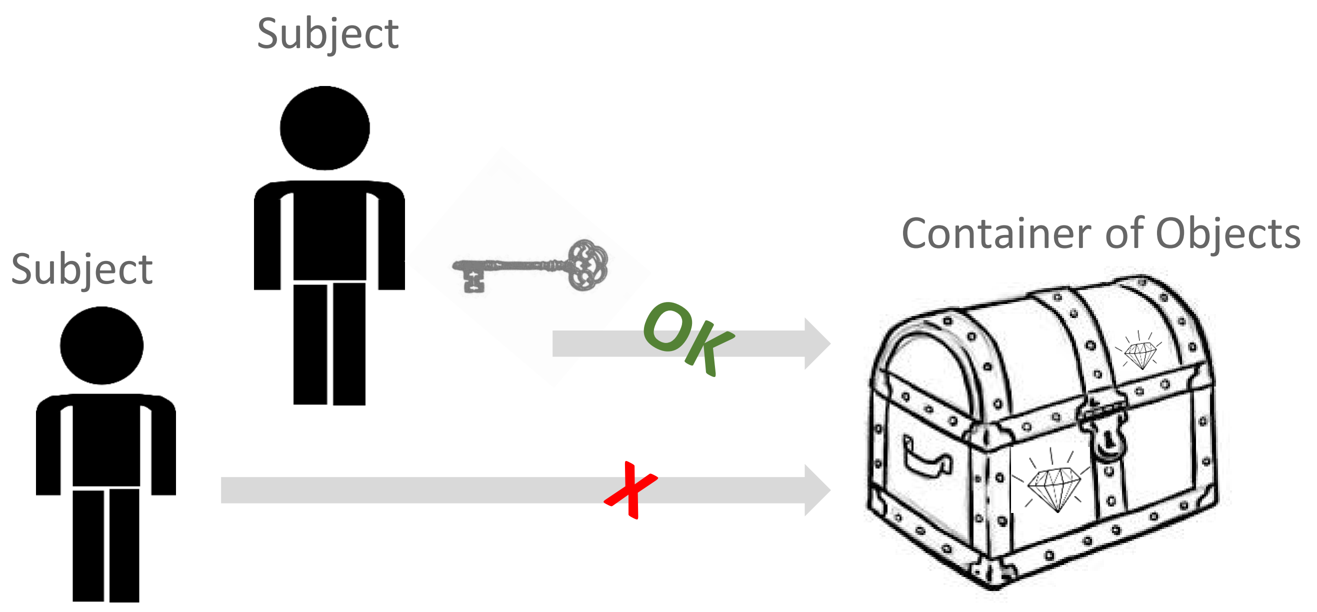
1. Principles and Terminology
2. Access control Methods
    1. Data protection
    2. Platform Protection

    ←- 5 min break -→

3. Access Control in Practice: SELinux
    1. Domain-type architecture
    2. Integration
    3. Policy language and configuration
4. Conclusions

Access Control Terminology

Access Control is the operation by which we selectively constrain access to objects from subjects

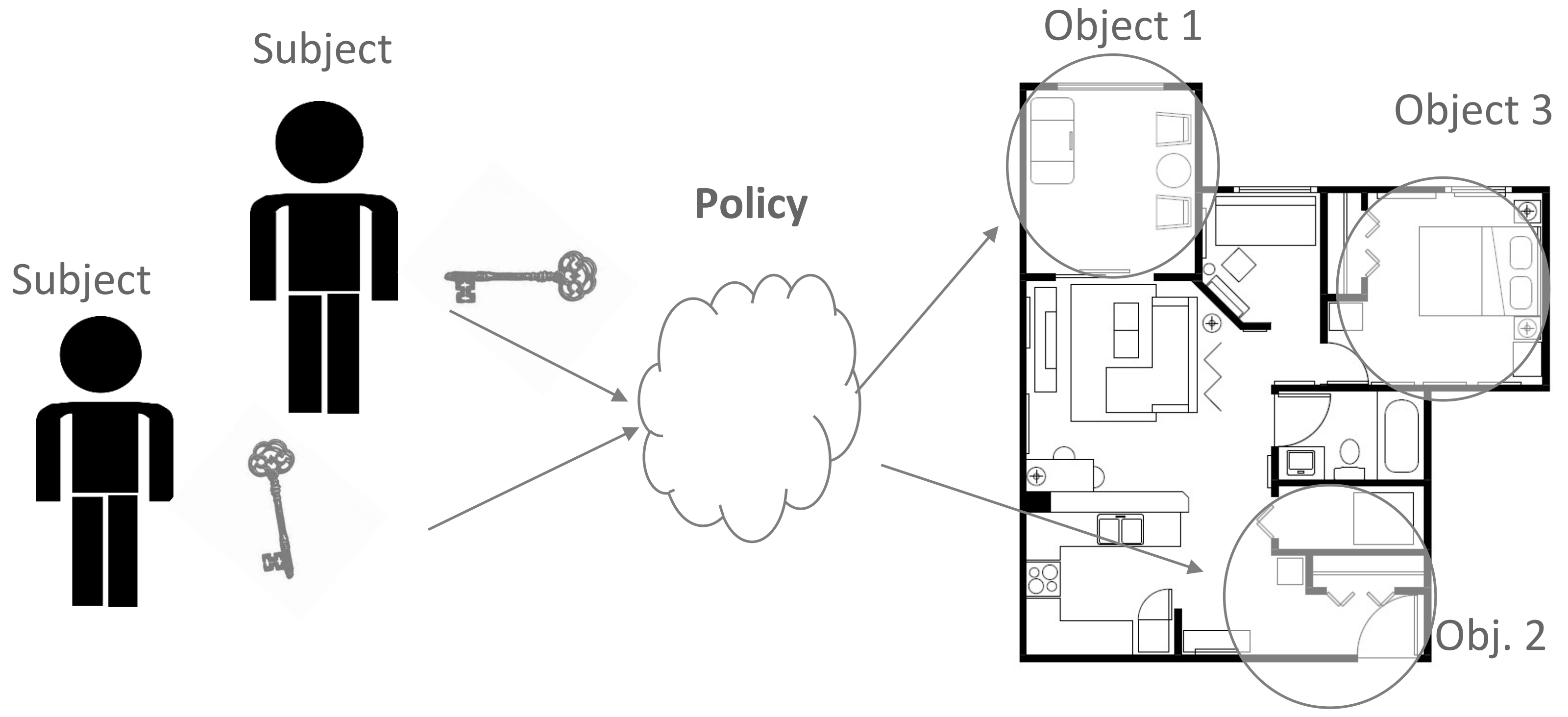


Here the access control is capability-based. Whoever has the capability (key) will get to the objects
Access control can also be identity-based. To resolve the subject identity, authentication may be needed

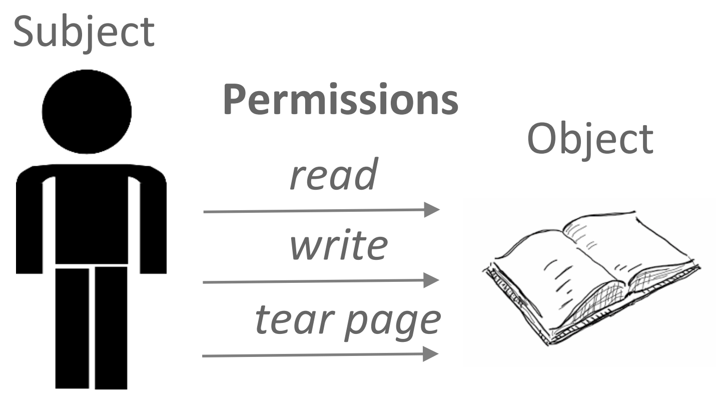
A subject can be a natural person. But it can also be a program, a computer in a network or either of these operating on behalf of a person
An object can be a physical object or space, data (a file, an area of memory), a resource, a program, a function, or another subject

Access Control Terminology 2

The ruleset that defines which subjects get access to what objects is called the access control policy



If there are more than one action that can be applied to an object (by a subject) these actions are called permissions, and can be specified in the policy for each subject/object pair of a given type



In Discretionary Access Control (DAC) the policy is set by users of a computing system
In Mandatory Access Control (MAC) the policy is decided by the operator of a computing system
In Role-Based Access Control (RBAC) the policy for roles is MAC, but subject are assigned to roles (business approach)

History: Protecting the Assets – Access control

Butler Lampson, Xerox PARC, 1971: **Protection**

"The original motivation for putting protection mechanisms into computer systems was to **keep one user's malice or error from harming other users**.

Harm can be inflicted in several ways:

- by destroying or modifying another user's data;
- by reading or copying another user's data without permission;
- by degrading the service another user gets, e.g. using up all the disk space, or … (DoS)"

Paper is significant. Maybe the biggest insight is that Lampson diverges from a time-sharing model, of computation, and starts thinking of a computer system with subjects and objects that are entities in their own right, rather than being a computational workload in representation of a physical person.

1

2

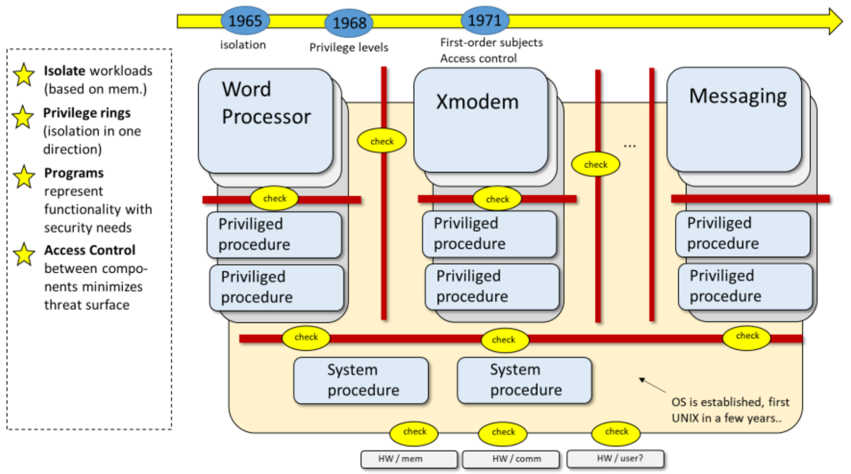
A program can be a qualified subject

"More recently it has been realized that all of the above reasons for wanting protection are just as strong if the word 'user' is replaced by 'program'. This line of thinking leads in two main directions:" .. [access control and programs-as-a-business]

Least-privilege

"Towards enforcing the rules of modular programming so that it is possible, using the protection system, to guarantee that errors in one module will not affect another one. With this kind of control it is much easier to gain confidence in the reliability of a large system, since the protection provides firewalls which prevent the spread of trouble"

"System security" first defined?



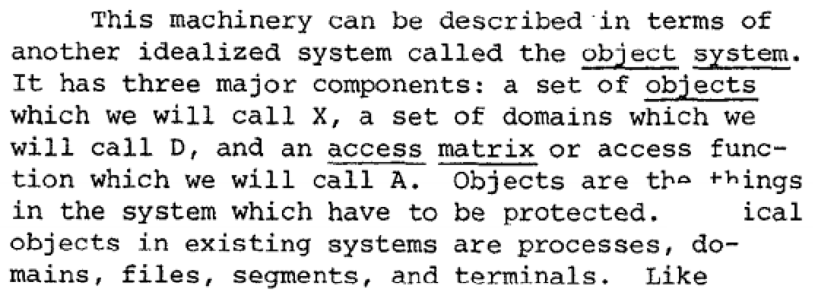
1965
isolation
1968
Privilege levels
1971
First-order subjects
Access control
Isolate workloads (based on mem.)
Privilege rings (isolation in one direction)
Programs represent functionality with security needs
Access Control between compo-nents minimizes threat surface
Word Processor
Xmodem
Messaging
check
...
Priviliged procedure
System procedure
OS is established, first UNIX in a few years..
HW / mem
HW / comm
HW / user?

https://dl.acm.org/doi/pdf/10.1145/775265.775268

History: Access Control Matrix

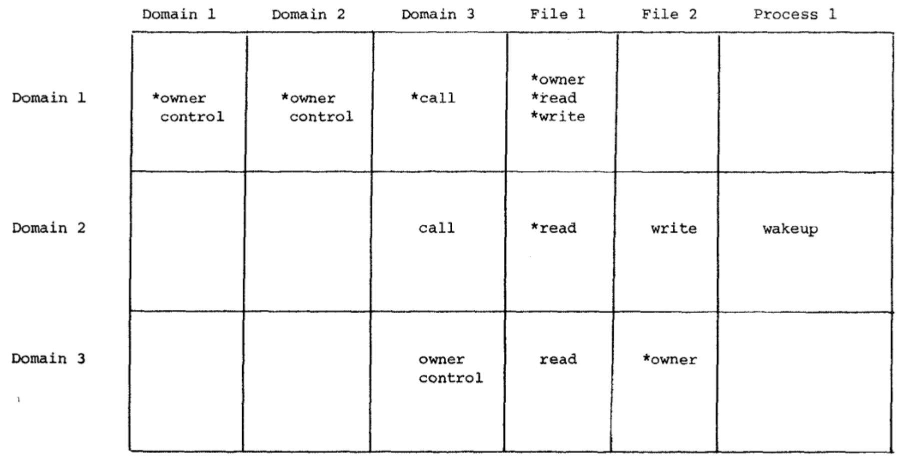
Butler Lampson, Xerox PARC, 1971: **Protection**

- Highlights the understanding that isolation is not enough, security is achieved by data-flow and operational constraints
- Domains are groups of subjects with similar access rights to objects
- Objects can be system resources (even HW), or domains/subjects providing services to other subjects
- Objects grant attributes (access rights) to subjects – e.g. read/write. One important attribute is “owner”, which allows for access rights to be set up and delegated (i.e. an attribute allowing control over the matrix)
- Memory Protection (e.g. segmented memory) fits well in this model of access control (Lampson’s note)
- The matrix is a model, and as it typically is sparse, different optimization patterns do exist (ACL, Caps, …)
- The model, and rules in the model, will give cause to a requirement for naming / identity. and the proper mapping of such names to the model (ed.note: integrity)



This machinery can be described in terms of another idealized system called the object system. It has three major components: a set of objects which we will call X, a set of domains which we will call D, and an access matrix or access function which we will call A. Objects are the things in the system which have to be protected. ical objects in existing systems are processes, domains, files, segments, and terminals. Like

Example matrix, from Lampson’s paper



	Domain 1	Domain 2	Domain 3	File 1	File 2	Process 1
Domain 1	*owner control	*owner control	*call	*owner *read *write		
Domain 2			call	*read	write	wakeup
Domain 3			owner control	read	*owner	

https://dl.acm.org/doi/pdf/10.1145/775265.775268

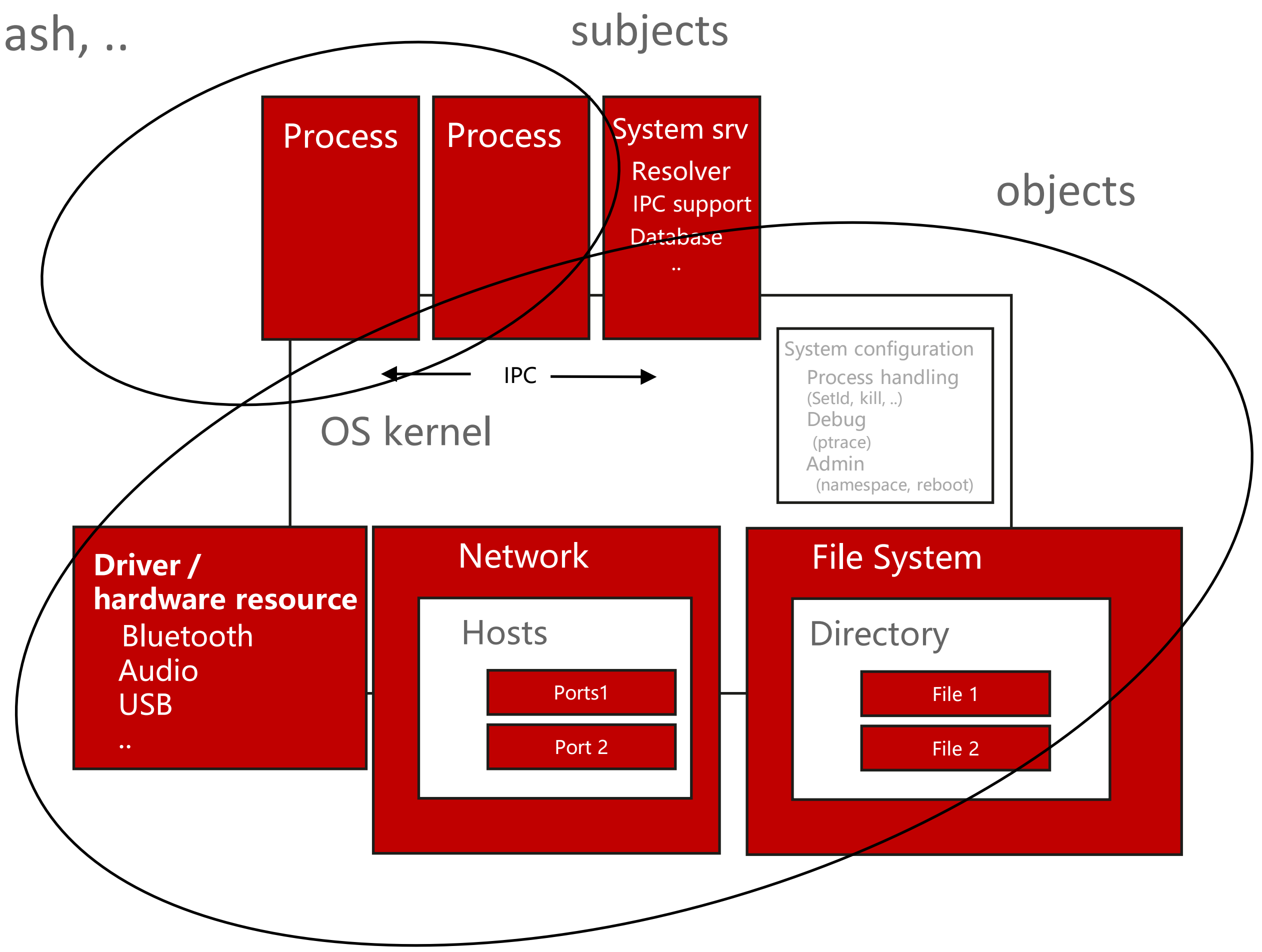
Access Control Terminology 3

Excluding physical persons and network access control,
what are typical subjects and objects in a computer system?

The common subject is a process. However, the identity of a process can be a program id, an executable file hash, ..

The objects are resources of some kind. Although processes (user-space services) can be a resource, also files, system configuration ability or driver access are common resources to control in an OS

It is obvious that the numbers of possible subjects and objects in a computer can be very large. E.g. a Debian distribution has around 500.000 files, 100+ device nodes and an 'idling desktop' has some 2-300 processes / threads running at any given time



Access Control Terminology 4

What is the logical place to do AC enforcement? Some de-facto rules apply

Where does enforcement take place?

Ideally in one place that cannot be by-passed by the attacker

Enforcement vs. Policy:

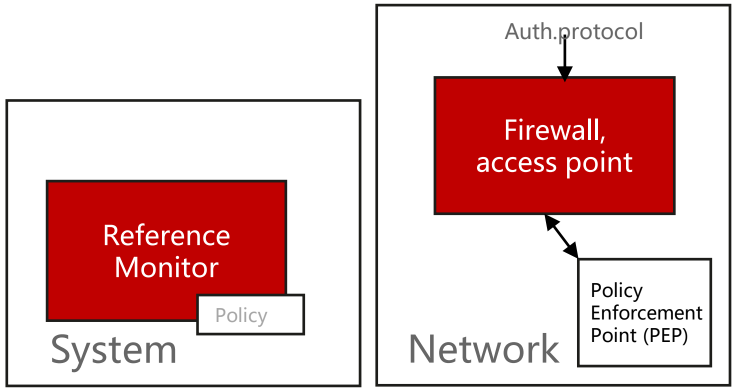
Consensus is to keep these strictly apart for maximum flexibility.

However, there is no technical show-stopper to e.g. compile policy into an executable filter

Theoretically

... The term reference monitor denotes a system component that enforces the access control for user (processes) in an un-bypassable secure way

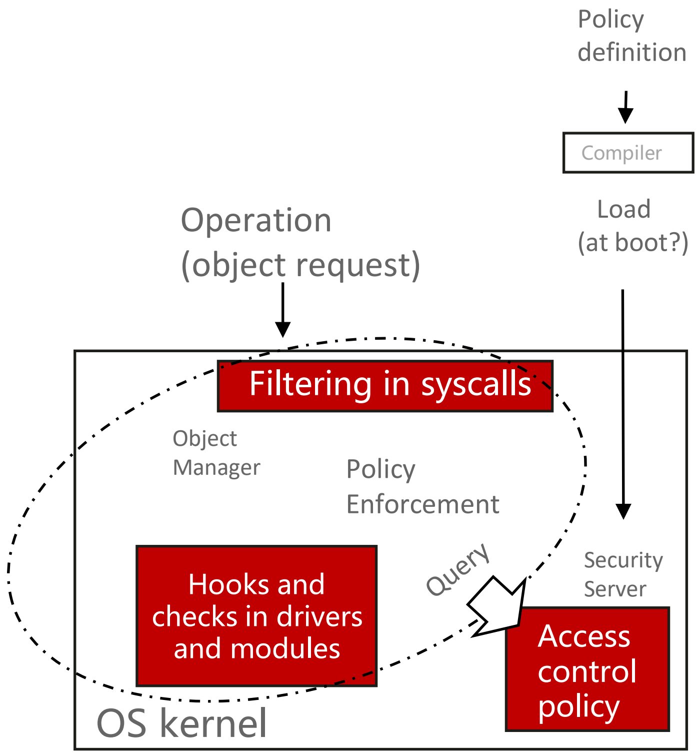
E.g. in WAN / LAN, a single Policy Enforcement Point (PEP) can handle the access control decisions for all components at the network perimeter



The Flask Architecture

In late 90s, the access control mechanisms stabilized in both Linux and BSD Unices

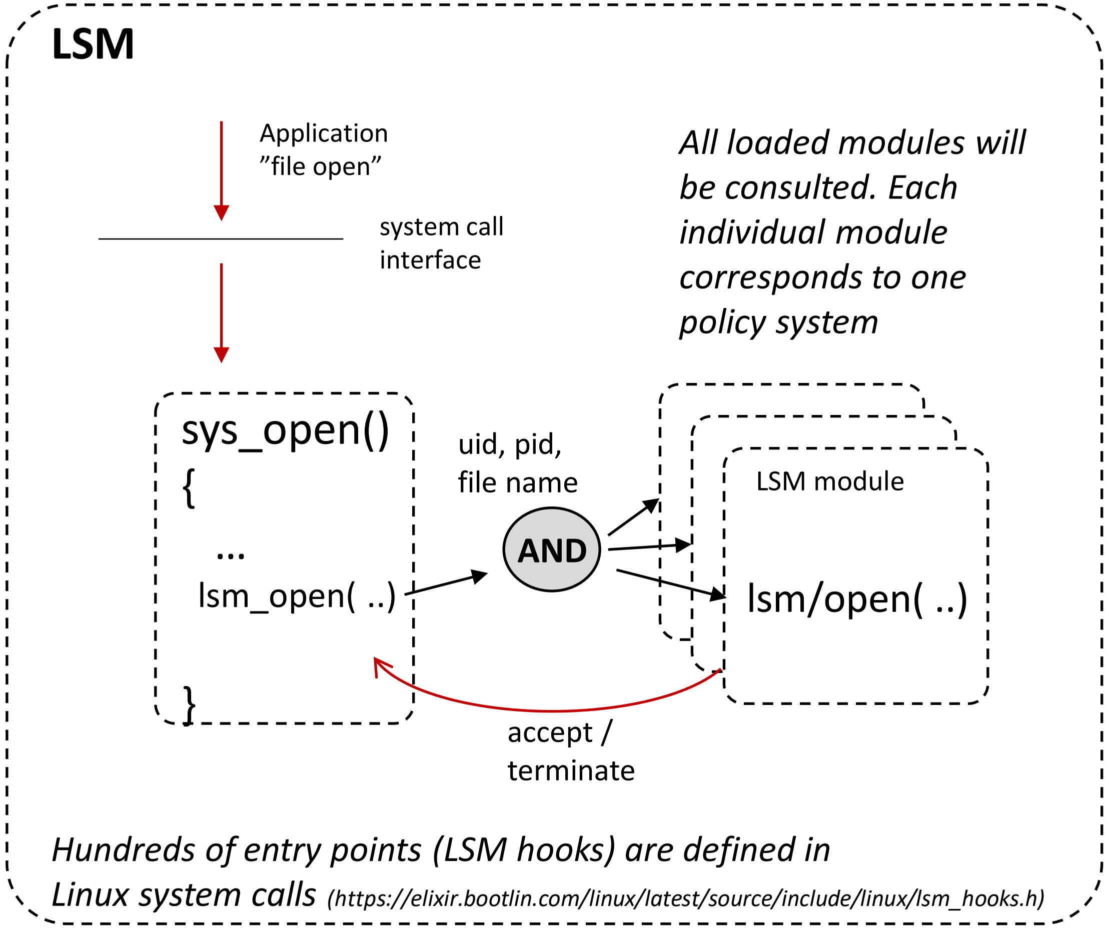
Also TrustedBSD MAC is similar, although relies more on syscall filtering



https://www.usenix.org/legacy/publications/library/proceedings/sec99/full_papers/spencer/spencer.pdf
https://www.usenix.org/legacy/publications/library/proceedings/usenix03/tech/freenix03/full_papers/watson/watson.pdf

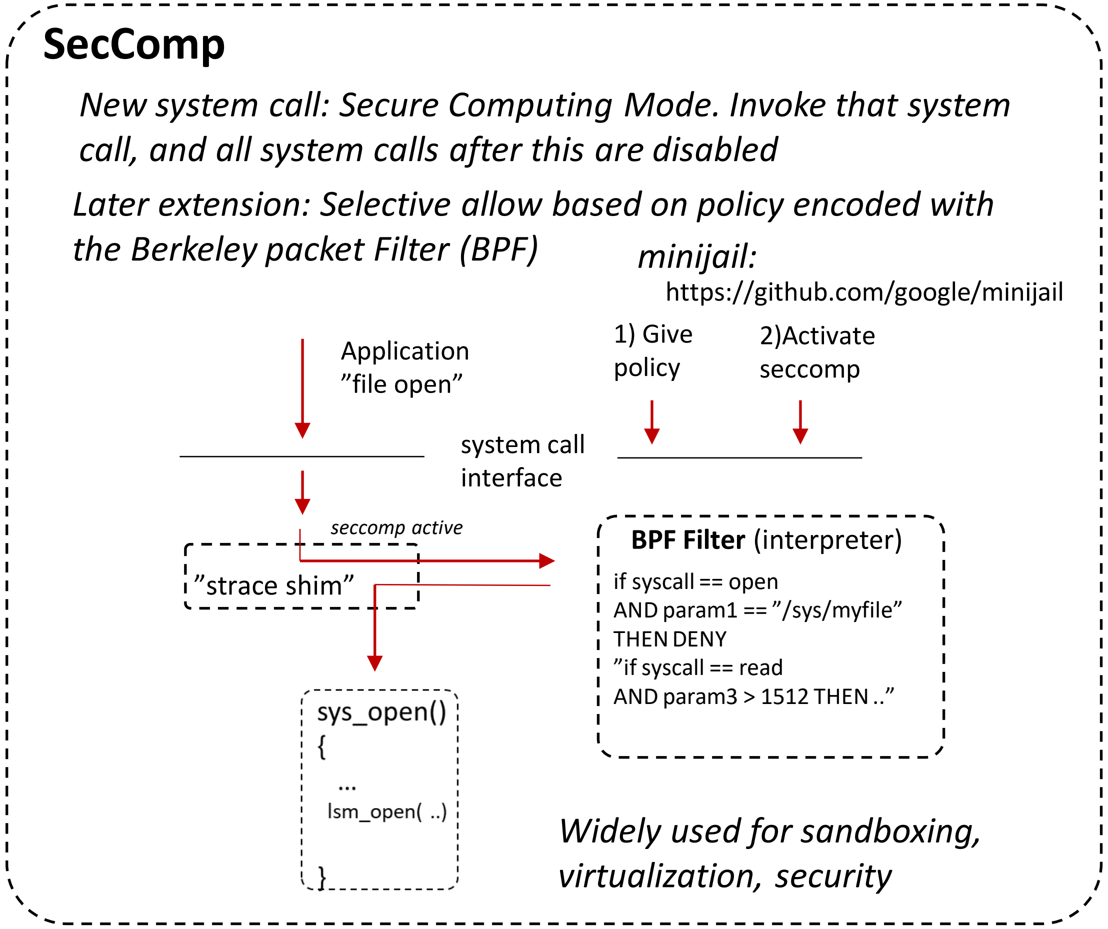
Terminology: Linux Security Modules (LSM) and SecComp /BPF

- A **reference monitor** is a tamper-proof OS component that mediates **every access** and which is small enough to reason about. Does not define policy by itself, but is the enforcing component (separation of mechanism and policy)
- The stackable LSM architecture has been present in Linux for 20 years already. SecComp/BPF is a more recent and fine-grained addition. In other versions of Unices, also TrustedBSD is a widely deployed monitor architecture.



All loaded modules will be consulted. Each individual module corresponds to one policy system

Hundreds of entry points (LSM hooks) are defined in Linux system calls *(https://elixir.bootlin.com/linux/latest/source/include/linux/lsm_hooks.h)*



New system call: Secure Computing Mode. Invoke that system call, and all system calls after this are disabled

Later extension: Selective allow based on policy encoded with the Berkeley packet Filter (BPF)

minijail: https://github.com/google/minijail

```
if syscall == open
AND param1 == "/sys/myfile"
THEN DENY
"if syscall == read
AND param3 > 1512 THEN .."
```

Widely used for sandboxing, virtualization, security

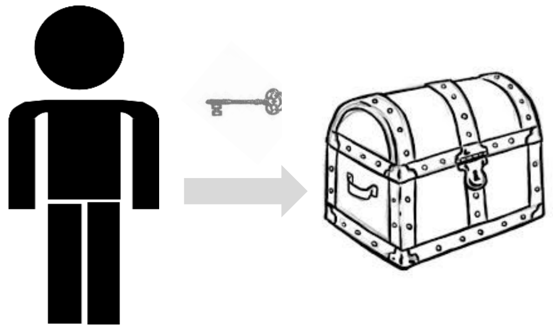
LSM: https://www.usenix.org/legacy/event/sec02/full_papers/wright/wright.pdf
SecComp/BPF: https://www.usenix.org/system/files/conference/atc13/atc13-kim.pdf

ReCap 1:

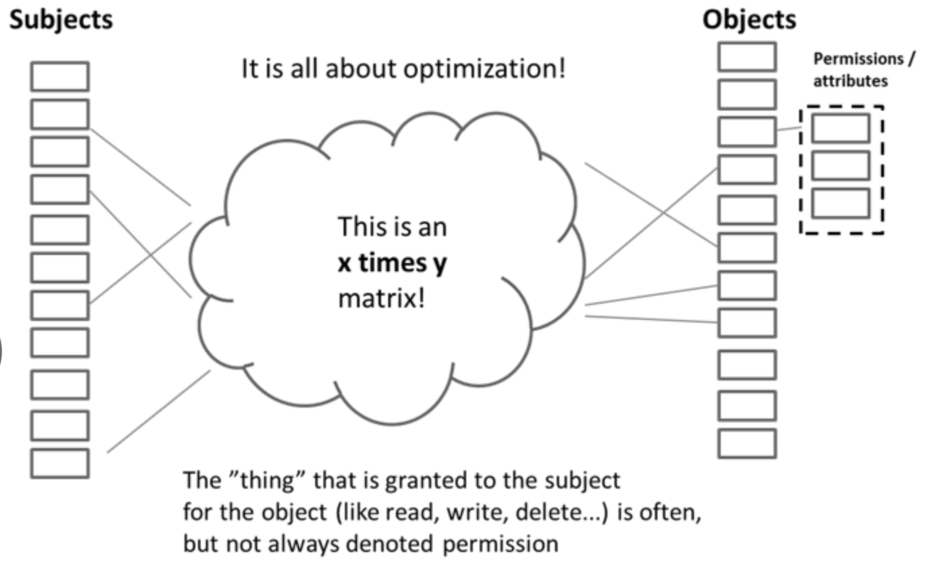
For now we have learned the following things about access control in a computer system:

- Subjects access Objects (or resources)
- 1-n permissions may apply to a single access rule
- The totality of access rights in a system can be represented by a matrix

- Access control decisions in a system are taken by a reference monitor
- What is allowed for whom is dictated by a policy
- Preferrably, the policy is represented as data, enforced by the monitor



Next, we will look at different access control methods / methodologies
(and there exists an exhaustive amount of them out there)


Subjects
Objects
It is all about optimization!
Permissions / attributes
This is an x times y matrix!
The "thing" that is granted to the subject for the object (like read, write, delete...) is often, but not always denoted permission

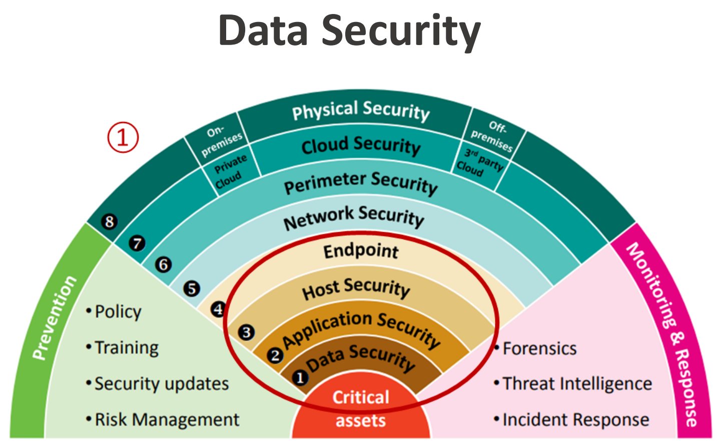
Access Control Methods

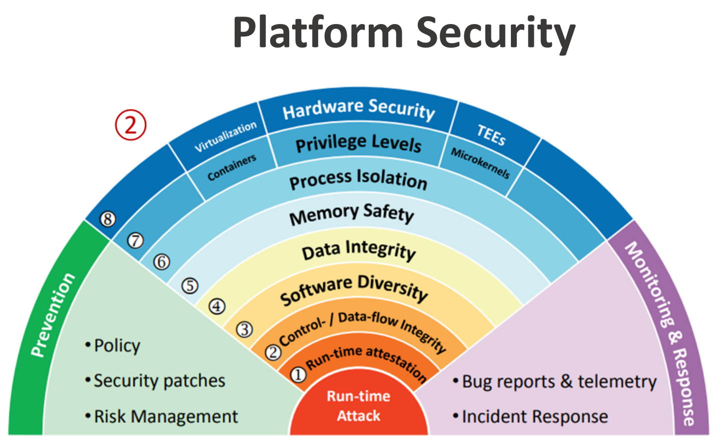
- When we are looking at how to abstract the access control matrix, there will be a distinction between
  a) Data Security: How to limit / control access to data → Protect the data
  b) Platform Security: Limit access from untrustworthy subjects to critical resources → Protect the device

- Of course these directions overlap somewhat, i.e for platform security
  Some configuration data (or cryptographic keys) is critical to correct device operation
  A fundamental concept of platforms security is isolation of workloads (and their data)
  → But typically, the criticality of the data itself is not considered

- Whereas with data security we consider data criticality (secret, confidential, business confidential, public), and this is the asset we want to protect. In data security, the subject is the processor of the data, and the object is the storage of the data

- Because of this, methods are only partially compatible across this boundary.
  We will start with a brief overview of data access control and focus more on platform mechanisms later.

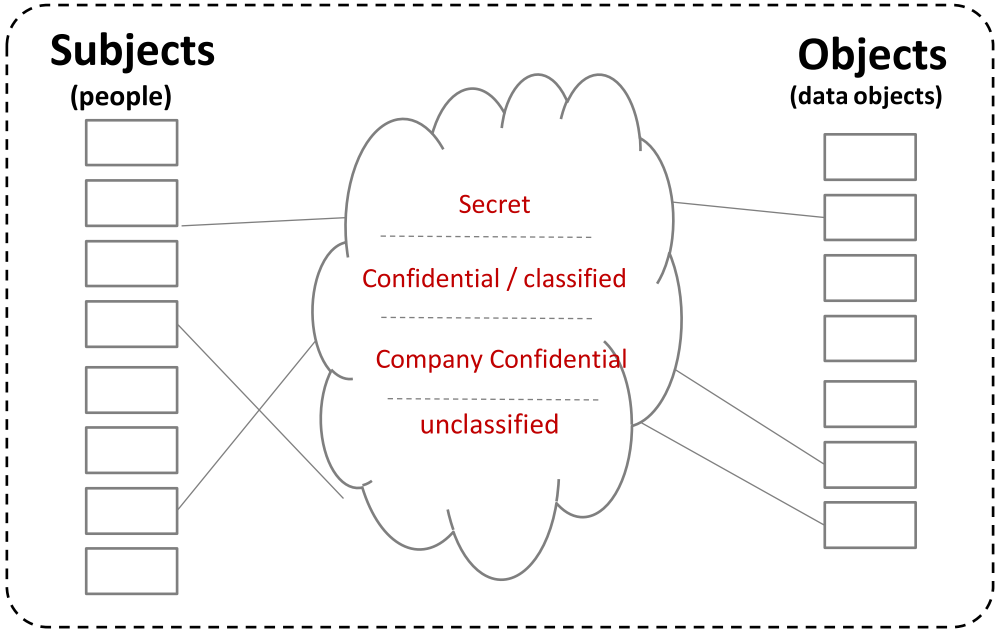




(1) **Savold & al**: A survey of the leading cyber reference architectures and frameworks. In Proceedings of the 4th IEEE International Conference on Cyber Security and Cloud Computing, CSCloud '17
(2) Thomas Nyman (PHd): https://aaltodoc.aalto.fi/bitstream/handle/123456789/47029/isbn9789526400655.pdf?sequence=1

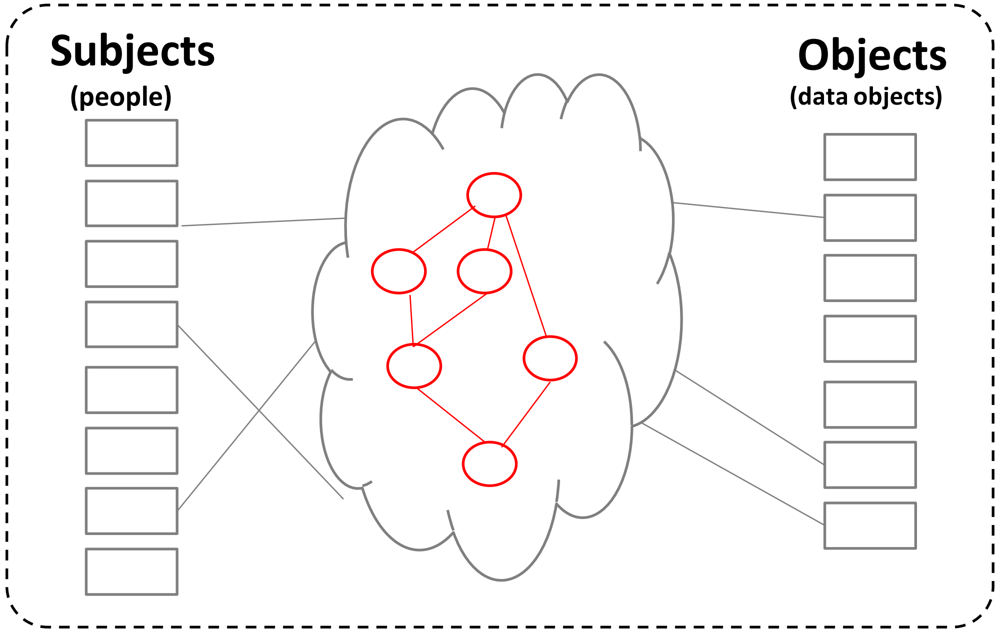
Multi-level Security (MLS)

- MLS is mostly a concept by which data access control is exercised. Main insight is that data security often can be described in ’levels’ of access control need.
- In MLS the access control matrix is ’collapsed’ into a number of security levels. Subjects have <u>security clearance</u> and data objects are labeled based on the <u>criticality</u> of the information. The control decision is resolved based on policy discussed in the next few slides



Lattice-based Access Control

- A lattice is a data structure describing a partial order between nodes in a graph. This can be considered a refinement of level-based security, where e.g. access control decisions can be done based on <u>supremum</u> (lowest common ancestor) or infimum.
- A lattice can e.g. be used to combine level-based thinking with the notion of mutually exclusive subject groups or object attributes.



Bell-LaPadula Model

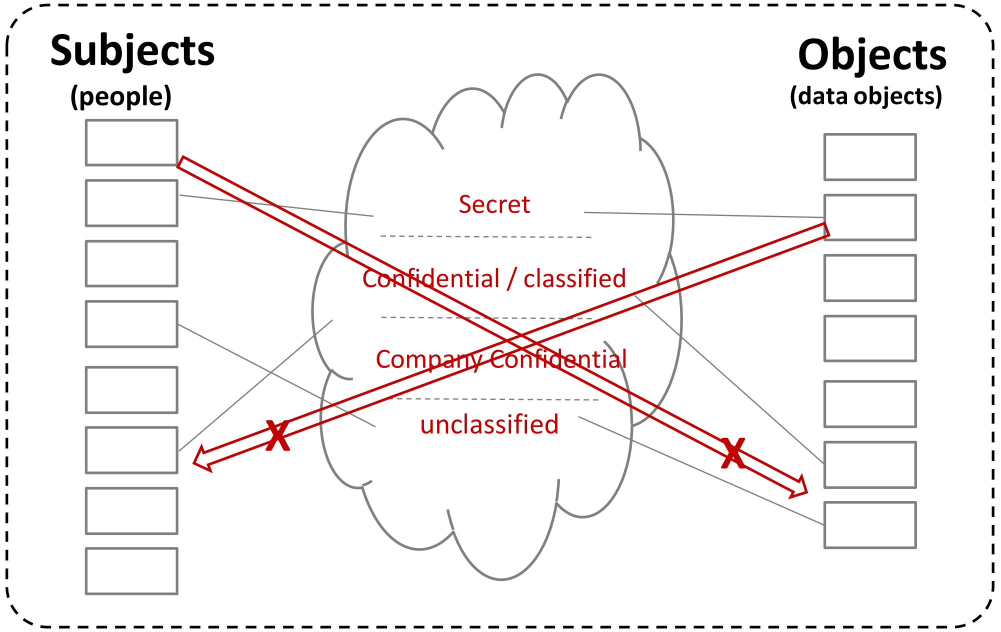
- An access control methodology for data confidentiality (origin in military computer system)
- Subjects can create content at or above their own security level
- Subjects can view content only at or below their own security level

Basic Rules:

No write down: Loss of confidentiality if written to insecure store (*-property)

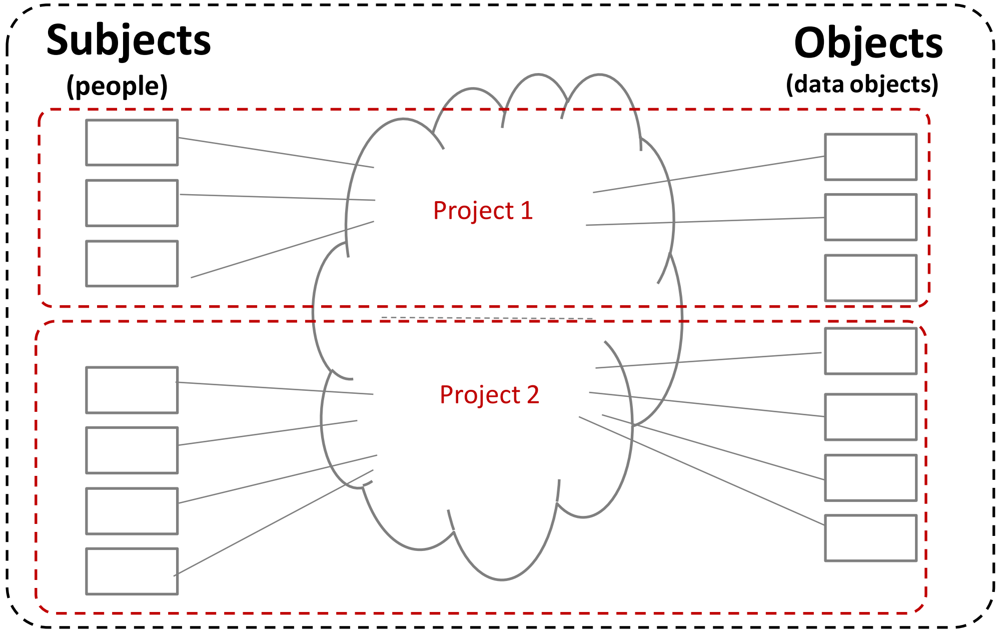
No read up: Secret documents cannot be read by people with low clearance

Trusted subjects can move data in violation of the rules.



Chinese Wall Model

- An access control methodology for data confidentiality but specifically targeting the issue of conflict-of-interest in business
- Also an accurate description of what is needed in application ecosystems for data produced and consumed by a given application



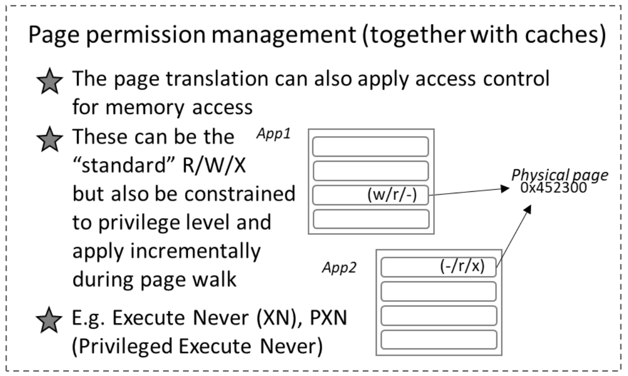
https://en.wikipedia.org/wiki/Bell%E2%80%93LaPadula_model

Discretionary Access Control (DAC)

- Most often associated with file-system protection, but basically applies to any protection that is configurable by the system user
- In most computers, there is a distinction of users, group membership administrator (root) identities, as well as the notion of “everyone”. The permissions in DAC apply to these identities as subjects.
- For each of the subject classes, permissions include file / directory read, write and execute. ‘Sticky’ permissions allow for limited inheritance. File metadata access or create/delete is seldom specifically set by permissions (but could be).

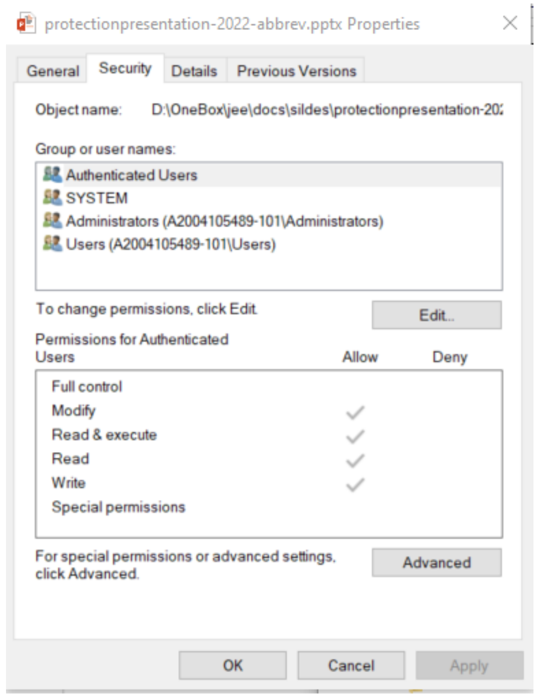
In most contemporary computers, also memory (MMU, MPU) is subject to similar access control (per memory page, R/W/C, W^X, PAN, ..)

For such permissions, the identity of the subject includes privilege level and potentially a hardware Id for running process (AID)



Page permission management (together with caches)

- ☆ The page translation can also apply access control for memory access
- ☆ These can be the “standard” R/W/X but also be constrained to privilege level and apply incrementally during page walk
- ☆ E.g. Execute Never (XN), PXN (Privileged Execute Never)



protectionpresentation-2022-abbrev.pptx Properties

General | Security | Details | Previous Versions

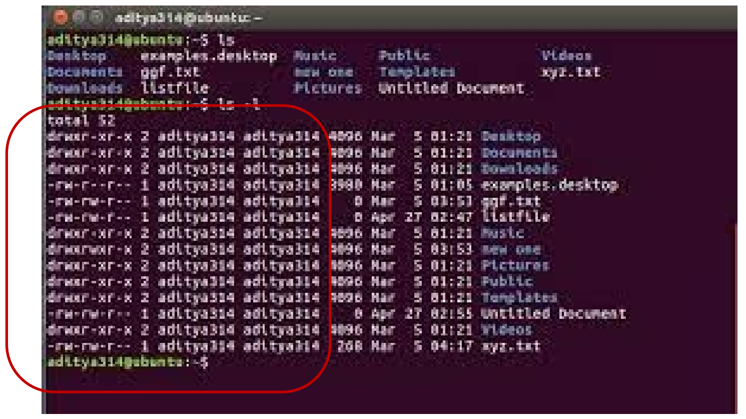
Object name: D:\OneBox\jee\docs\sildes\protectionpresentation-202

Group or user names:
- Authenticated Users
- SYSTEM
- Administrators (A2004105489-101\Administrators)
- Users (A2004105489-101\Users)

To change permissions, click Edit. [Edit...]

Permissions for Authenticated Users	Allow	Deny
Full control		
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

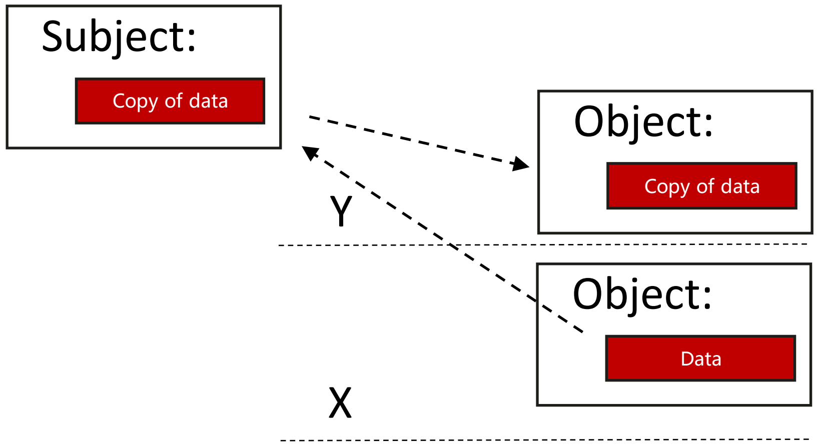
For special permissions or advanced settings, click Advanced. [Advanced]



```
aditya314@ubuntu: ~
aditya314@ubuntu:~$ ls
Desktop    examples.desktop  Music    Public            Videos
Documents  ggf.txt           new one  Templates         xyz.txt
Downloads  listfile          Pictures Untitled Document
aditya314@ubuntu:~$ ls -l
total 52
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Desktop
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Documents
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Downloads
-rw-r--r-- 1 aditya314 aditya314 8980 Mar  5 01:05 examples.desktop
-rw-rw-r-- 1 aditya314 aditya314    0 Mar  5 03:53 ggf.txt
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:47 listfile
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Music
drwxrwxr-x 2 aditya314 aditya314 4096 Mar  5 03:53 new one
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Pictures
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Public
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Templates
-rw-rw-r-- 1 aditya314 aditya314    0 Apr 27 02:55 Untitled Document
drwxr-xr-x 2 aditya314 aditya314 4096 Mar  5 01:21 Videos
-rw-rw-r-- 1 aditya314 aditya314  268 Mar  5 04:17 xyz.txt
aditya314@ubuntu:~$
```

BiBa model (Low-watermark MAC)

- An access control methodology to protect the integrity of data. The main thinking is that data processed at higher security levels is ’better’, and should not be contaminated by insecure data movements
- In principle security levels need not be absolute across subjects and objects, these are lattice models
- A formal analysis of data integrity protection in multi-level systems is provided in a 1987 paper by David Clark and David Wilson, which also provides a more complex model called the Clark-Wilson model.

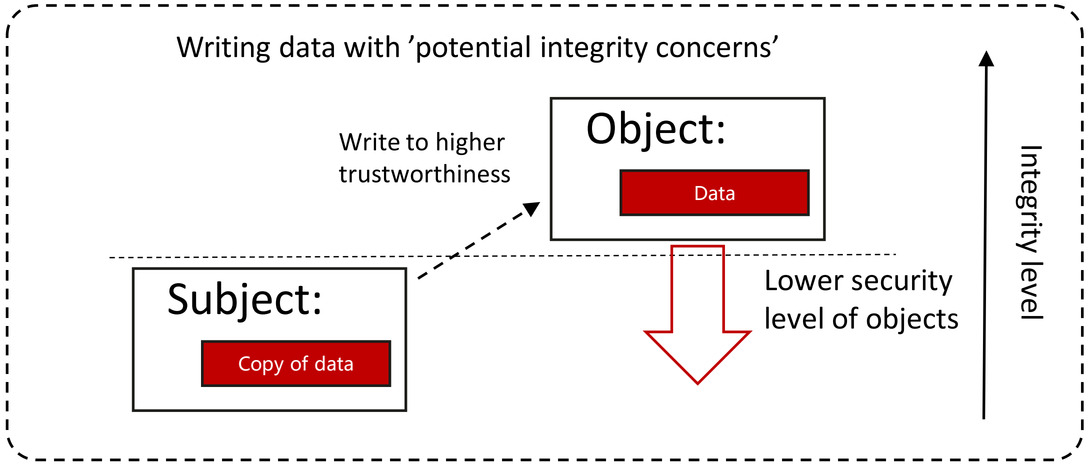


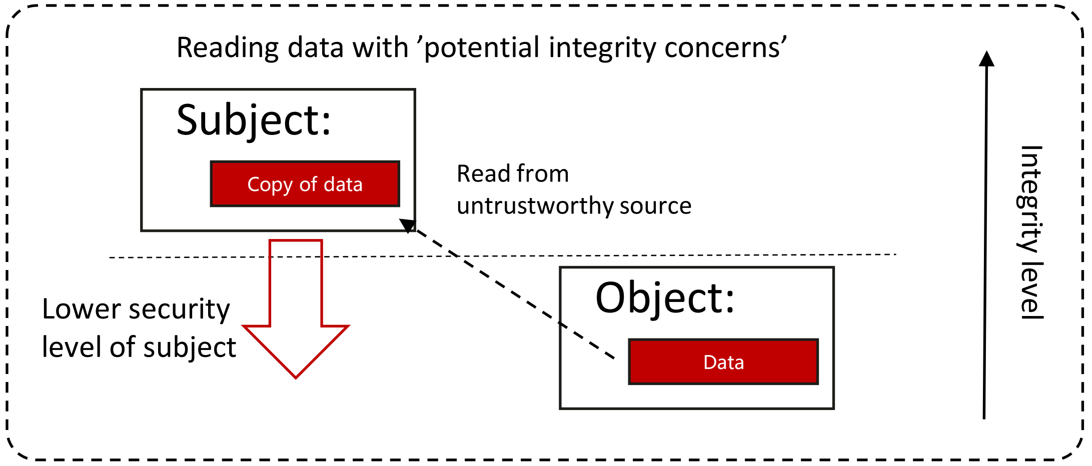
Basic Rules:

No write up: Contamination of higher-level data not allowed

Integrity *: If subject can read at level X, then it can not write at level Y, where Y > X

Low-watermark idea is to move subjects and objects consistently with the level of data they process


Writing data with ’potential integrity concerns’
Write to higher trustworthiness
Object:
Data
Subject:
Copy of data
Lower security level of objects
Integrity level


Reading data with ’potential integrity concerns’
Subject:
Copy of data
Read from untrustworthy source
Lower security level of subject
Object:
Data
Integrity level

https://en.wikipedia.org/wiki/Clark%E2%80%93Wilson_model

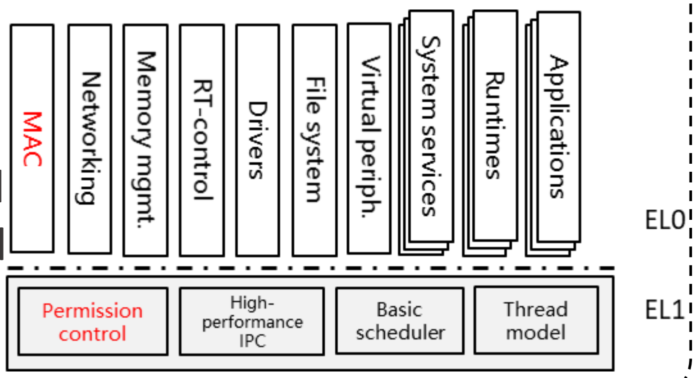
Capability vectors (in fact Permissions)

- The word 'capabilities' has dual meaning, and historically it refers to permissions stored in a bit vector. In access control a capability is a permission (handle) that can be freely delegated, which is not fully satisfied by e.g. Linux capabilities
- The most used capability vector are the Linux capabilities, which to date has some 40 bits in use. The main use of these capabilities is to split up the capabilities of the omnipotent 'root' user (uid=0) to more manageable permissions that can be granted based on need only.
- In principle, capabilities are inherited through execve calls, and executable binaries can have metadata that change the effective set of the new, launched thread.

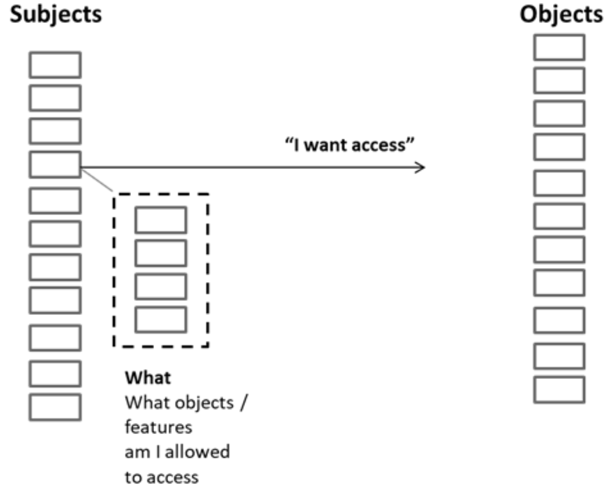
Capabilities in Microkernels

A microkernel is a kernel that externalizes to tasks as much of the OS operation as possible, and primarily handles task switching and IPC.

Again, some microkernels may implement full capability control. However, most have implement a task permission vector (called capability) covering memory resources and kernel services, and limiting task access that way



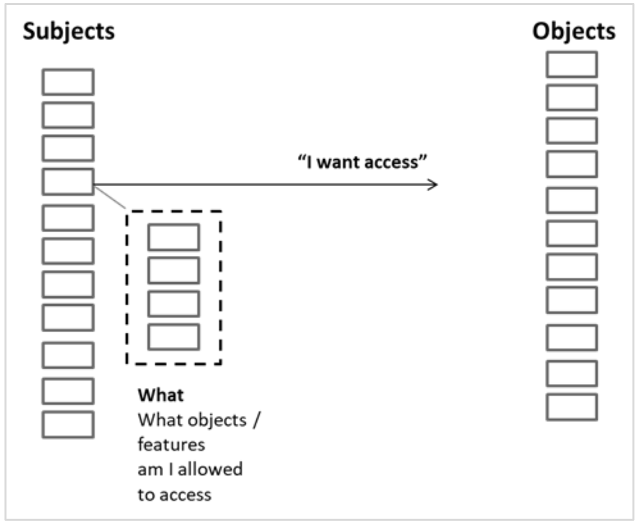
Permissions



Permissions

- The permissions pattern is today mostly used in system-level (user-space) frameworks in devices with managed application frameworks (Android, iOS, HarmonyOS, Windows).
- In these cases, the permissions are assigned to an application either by a remote trust root (app store), or by the user at his discretion (system permissions vs. user permissions). Often these are course-grained and user-understandable: Microphone, Address Book, Camera, Media files, Bluetooth, Network Access, Use whole screen …
- The framework maintains and enforces permission control on behalf of the application, and the number of permissions in a system is typically small: 30-150 permissions.
- No delegation or other fancy operations are allowed. An ongoing dispute is whether user-grantable permissions shall be requested when needed or at installation, or whether they expire over time, and need to be requested again.
- Sometimes extra policy can be applied to permissions. E.g. a MIC^SOUND policy makes it hard for applications to generate a communication application. Brings to front the issue of colluding applications.

Permissions



Capabilities

- Follows a scientific model: **The object-capability security model** for access control (see wikipedia)
- Layman's definition: *A capability is a transferable right to perform an operation on an object*
- Can be applied at many levels (HW / OS / System / Objects in a Program / System w. Network), ideally consistently across all these dimensions

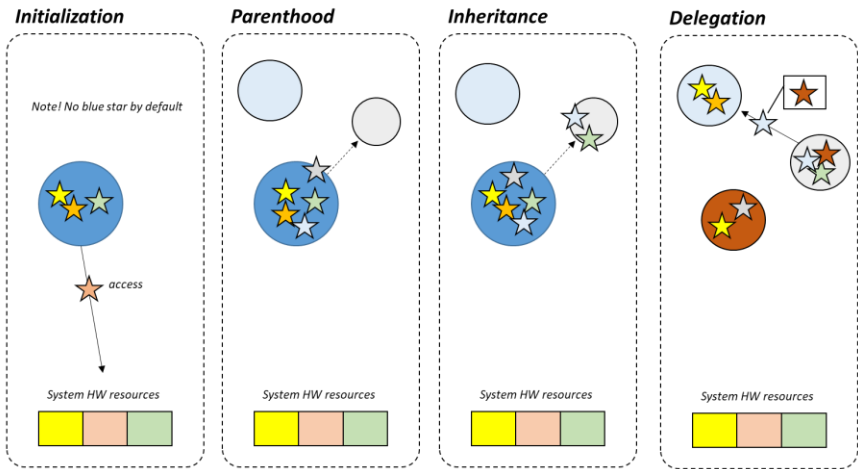
What:

1. ***Capabilities are unforgeable***: *Either HW enforced, or cryptographically enforced, or "kept track of"*
2. ***The capability references both object and operation*** *(ties to object naming)*

Abstractly there is no difference between object and operation (in the model). We can simplify to one capability for (object, operation)

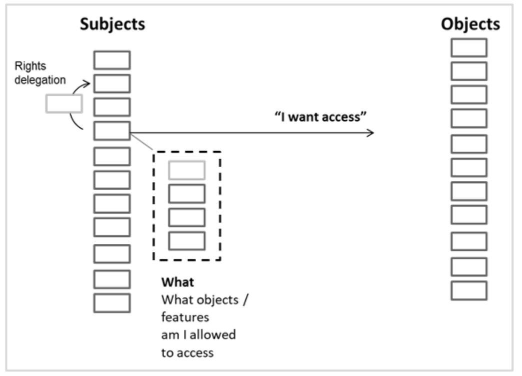
How:

1. ***Initialization***: *When the model starts, a capability may exist that is the eventual root of all future capabilities*
2. ***Parenthood***: *When a new object B is created by parent A, capabilities to all its (possible) operations C_x is obtained by A (not by B)*
3. ***Inheritance / Endowment***: *When a new object B is created by parent A, B is born with a subset of the capabilities owned by A (subset is defined by A)*
4. ***Delegation:*** *If A holds a capability for B AND A holds a communication capability for C, A is allowed to transmit (copy) the capability for B to C*



- A password capability is a handle that is too difficult to guess, maybe including a MAC
- Managed capabilities implement the mechanisms above, but protected e.g. by kernel

Capabilities



Access Control Lists

- Mostly has found its uses in networking: Which users are allowed to log in to the network or server
- A list of (authenticated) subjects that are allowed entry. In some sense, the password file in a computer is an access control list for a single object (the computer)
- In routing, used for accepting incoming traffic based on origin. Dynamic ACL are lists that adapt to traffic, e.g. as a consequence of a DoS attack

ACLs in Linux FS

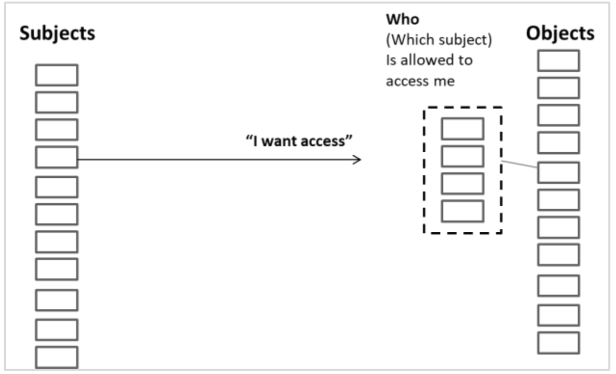
- Many popular filesystems in Linux (ext4fs,..) have extended attributes that allows file access beyond the traditional DAC
- These are implemented as access control lists and operated via fs extended attributes and *getfacl, setfacl, chacl* commands
- Allows e.g. separate access rules for different users

ACLs in Linux NW

- The Linux Netfilter kernel service operates an ACL for packet forwarding (subject = IP address)

https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa14/cse506.2/slides/ACLs-Vasu_and_Yaohui.pdf

Access Control Lists



SELinux: Domain-based Access Control with MLS and other extra ”salt”

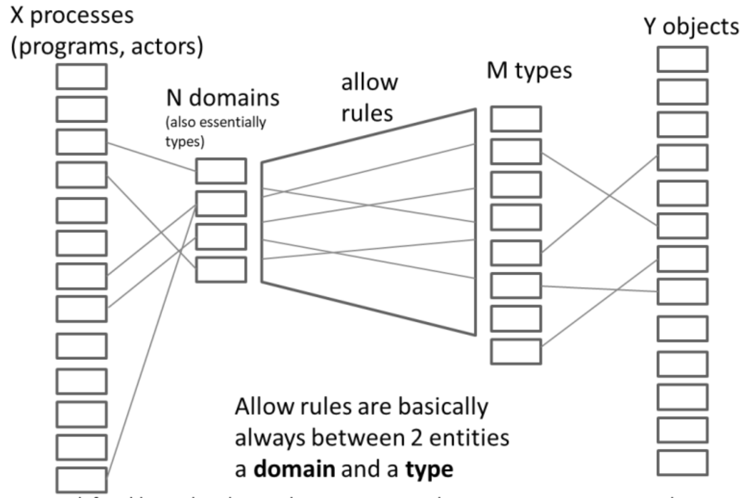
SELinux is an access control framework for Linux (mostly used for platform protection), in which 3 access control mechanisms co-exist somewhat in harmony

- The **Domain-Type** model is a classification of subjects and objects into groups, so that the access control matrix can be kept at reasonable complexity with thousands and millions of subjects and objects
- The Multi-level security **MLS** model in SELinux is a framework mechanism to support **data confidentiality and integrity** (Bell LaPadula, Biba).
- Also **ACL** (user-based access control) is in a limited way supported in SELinux.

Domain-Type

Subjects are primarily processes. Processes that participate in the access control are assigned a **domain**

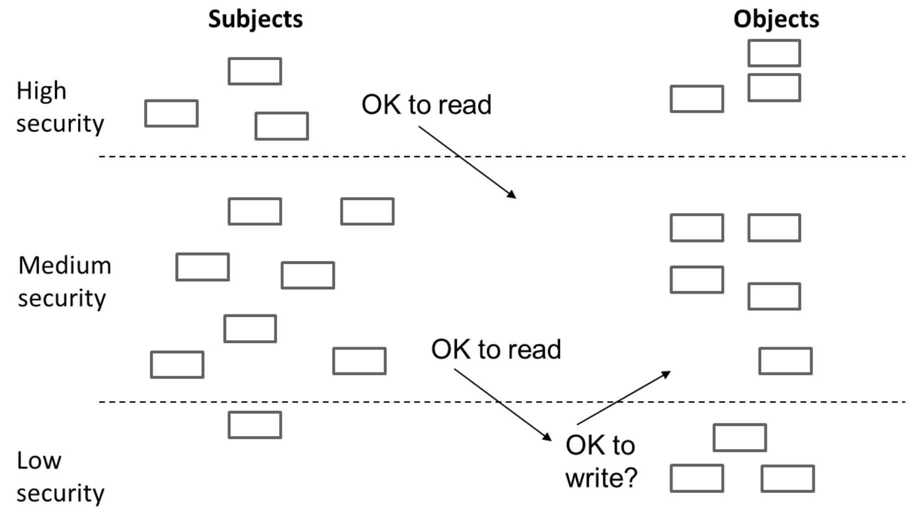
Objects are e.g. files, sockets.. Objects that participate in the access control are assigned a **type**



Allow rules are basically always between 2 entities a **domain** and a **type**

Domain and types are defined by each policy, and vary. Domain and type names are NOT FIXED by any agreement, i.e. **domains and types cannot be counted on to remain consistent in name or meaning across policy generations**.

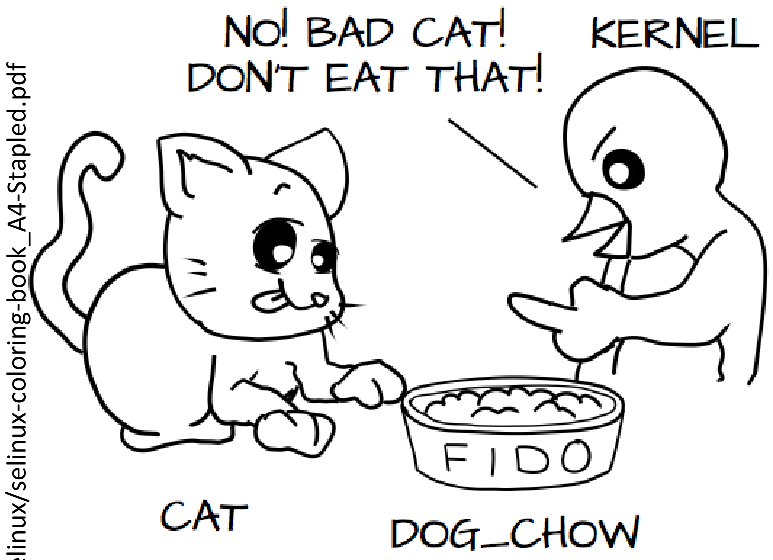
MLS (lattice, low-watermark systems)



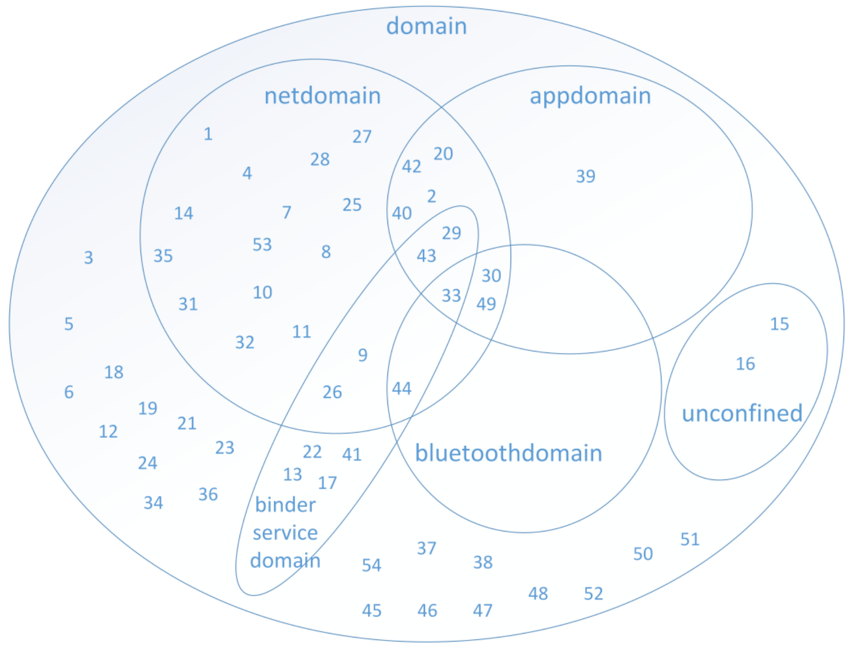
SEAndroid / SELinux, Policy languages and access control in practice

- … maybe a small break right here?

Pic: http://people.redhat.com/duffy/
selinux/selinux-coloring-book_A4-Stapled.pdf



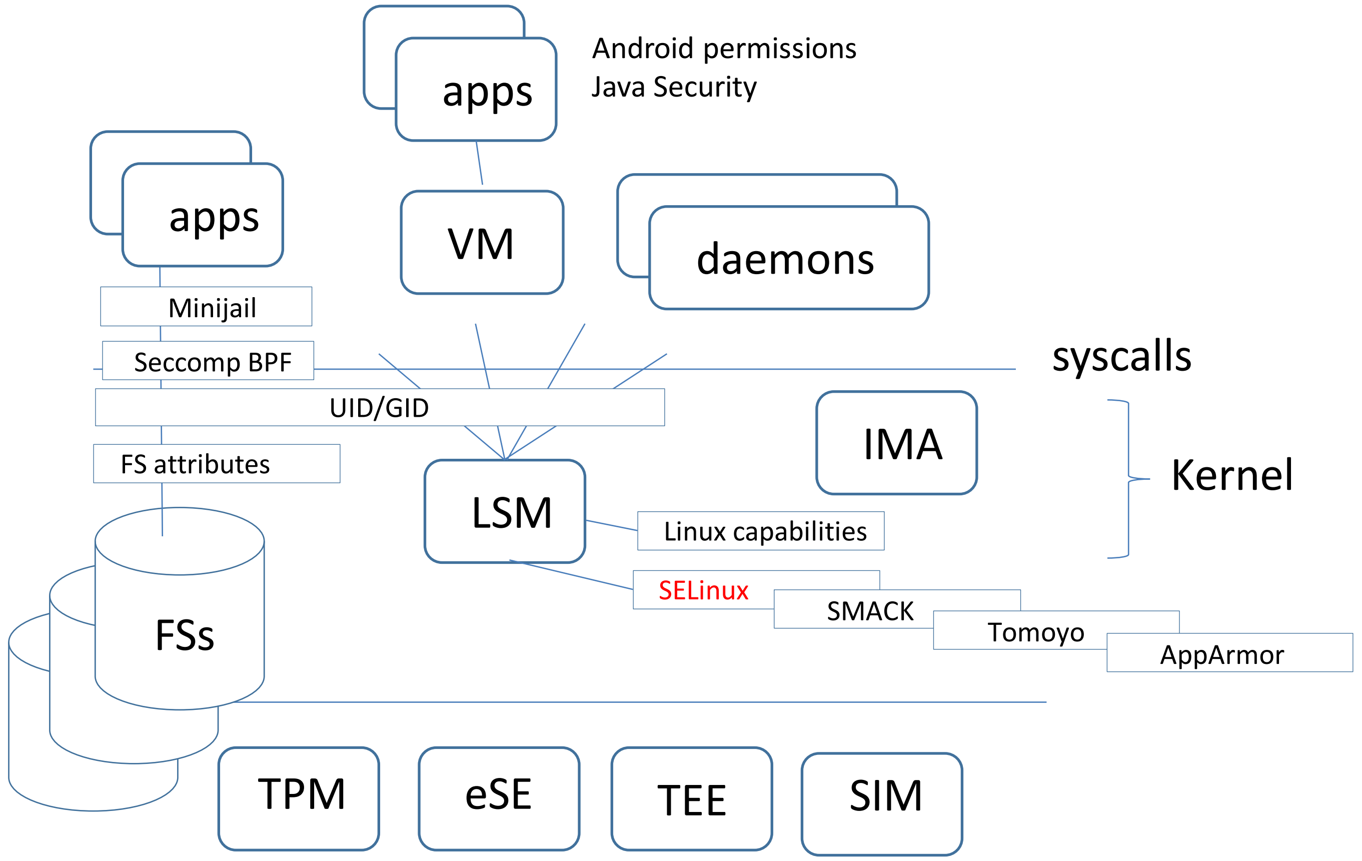
allow dog fido : dog_chow eat
neverallow {domain -dog} fido : * *



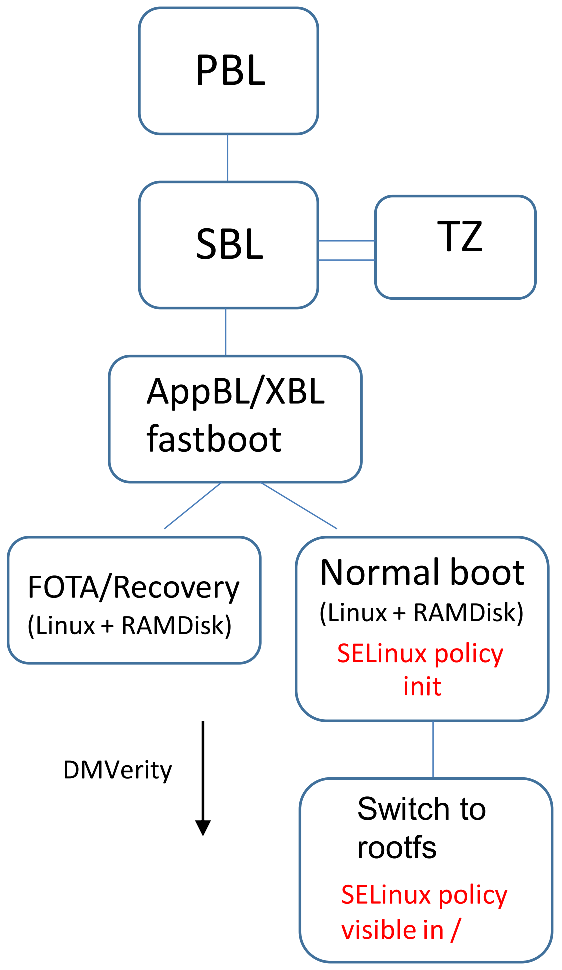
pic: Elena Reshtova

Platform Access Control (Linux)

1. Constrain access to least-privilege
2. Protect against infection
3. **Multi-user isolation**
4. **Multi-app isolation**



Secure Boot (~Android)



PBL
SBL
TZ
AppBL/XBL fastboot
FOTA/Recovery (Linux + RAMDisk)
Normal boot (Linux + RAMDisk)
SELinux policy init
DMVerity
Switch to rootfs
SELinux policy visible in /

1. Chain validated based on PK Hash in fuses
2. Many bootloaders from different stakeholders
3. Integrity guarantees (w. rollback protection)

1. Policy in RAMdisk
2. --> integrity “guaranteed”
   - Activation in init binary
   - --> Time of activation before full filesystem in use

Some milestones

- (Tech rep 1973) Multics Security Enhancements

System (kernel) access control

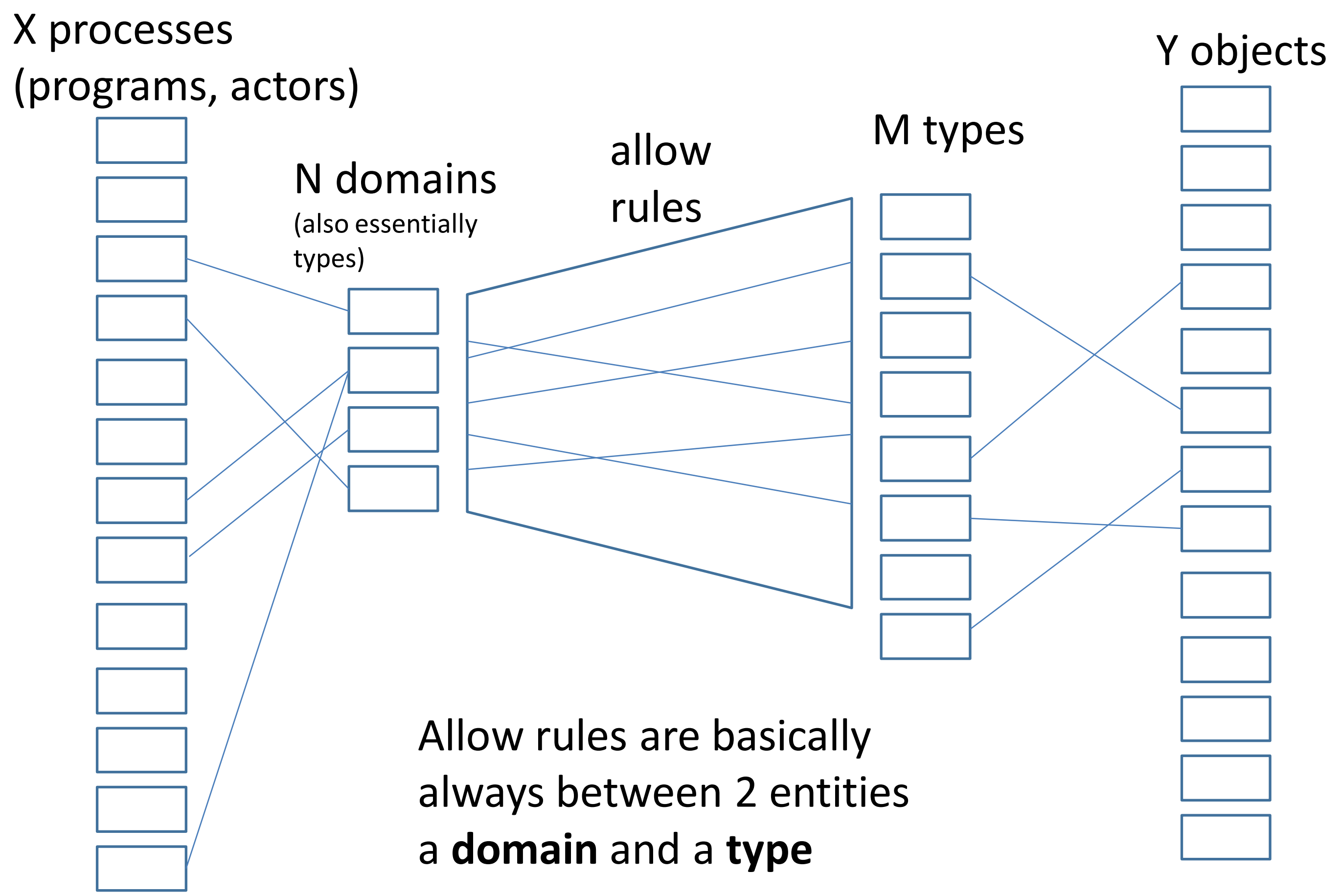
- (Usenix 99): The Flask Security Architecture
- Access control decision and enforcement separation
- (Usenix 01): Meeting Critical Security Objectives with
- Security-Enhanced Linux (Loscocco / Smalley)
-
- Domain-type enforcement for Linux (SELinux)

After 3 years of innovation and 15 years of implementation work →SEAndroid

Domain-type basic principle (dimension 1)

Subjects are primarily processes. Processes that participate in the access control are assigned a **domain**

Objects are e.g. files, sockets.. Objects that participate in the access control are assigned a **type**

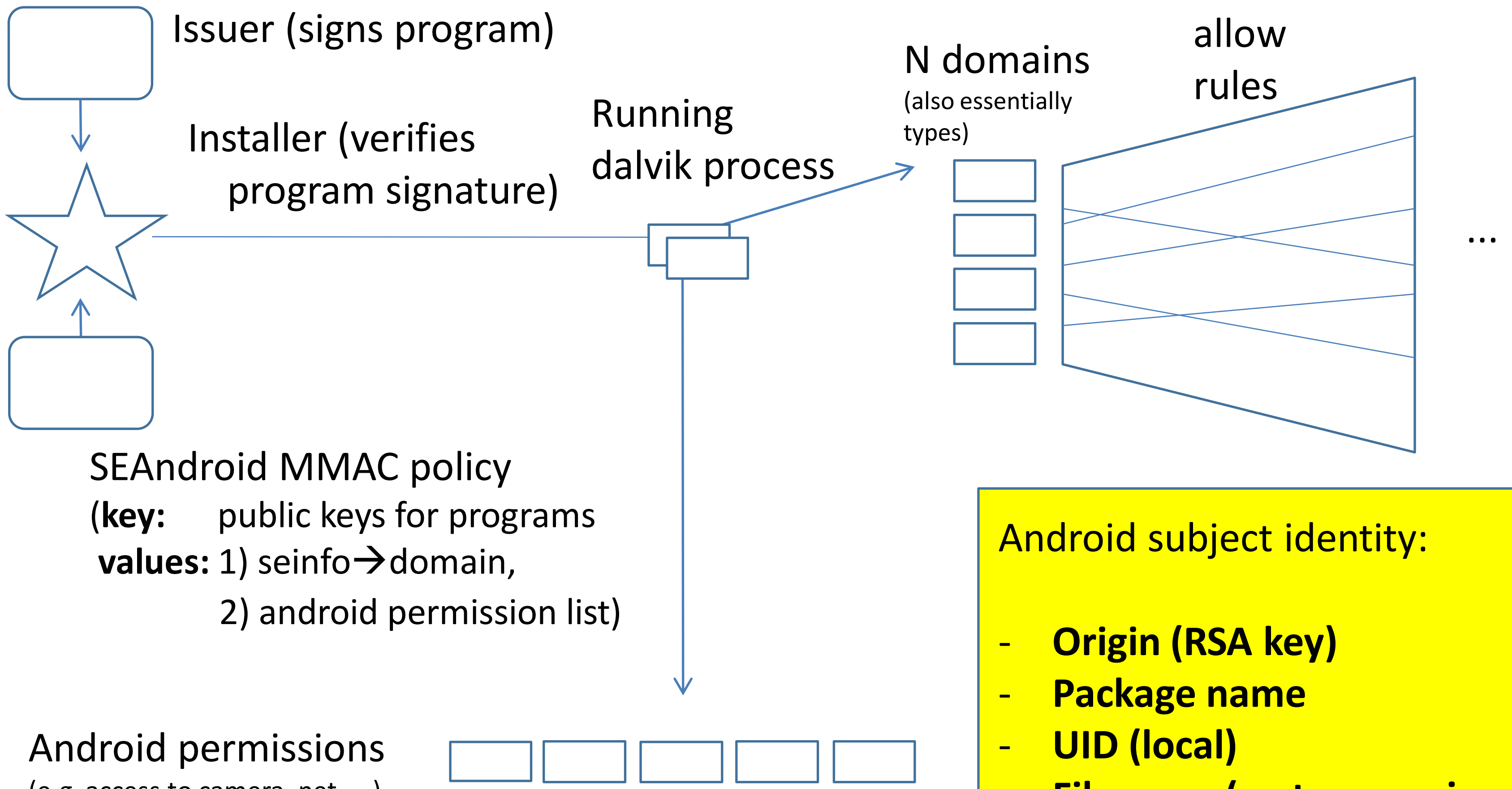


Allow rules are basically always between 2 entities a **domain** and a **type**

Domain and types are defined by each policy, and vary. Domain and type names are NOT FIXED by any agreement, i.e. **domains and types cannot be counted on to remain consistent in name or meaning across policy generations**.

Domain-type basic principle (dimension 1B)

In Android, the processes are to a large extent **middleware** processes running In the Dalvik virtual machine, and installed by the Android installer. The Middleware MAC (MMAC) addresses this detail



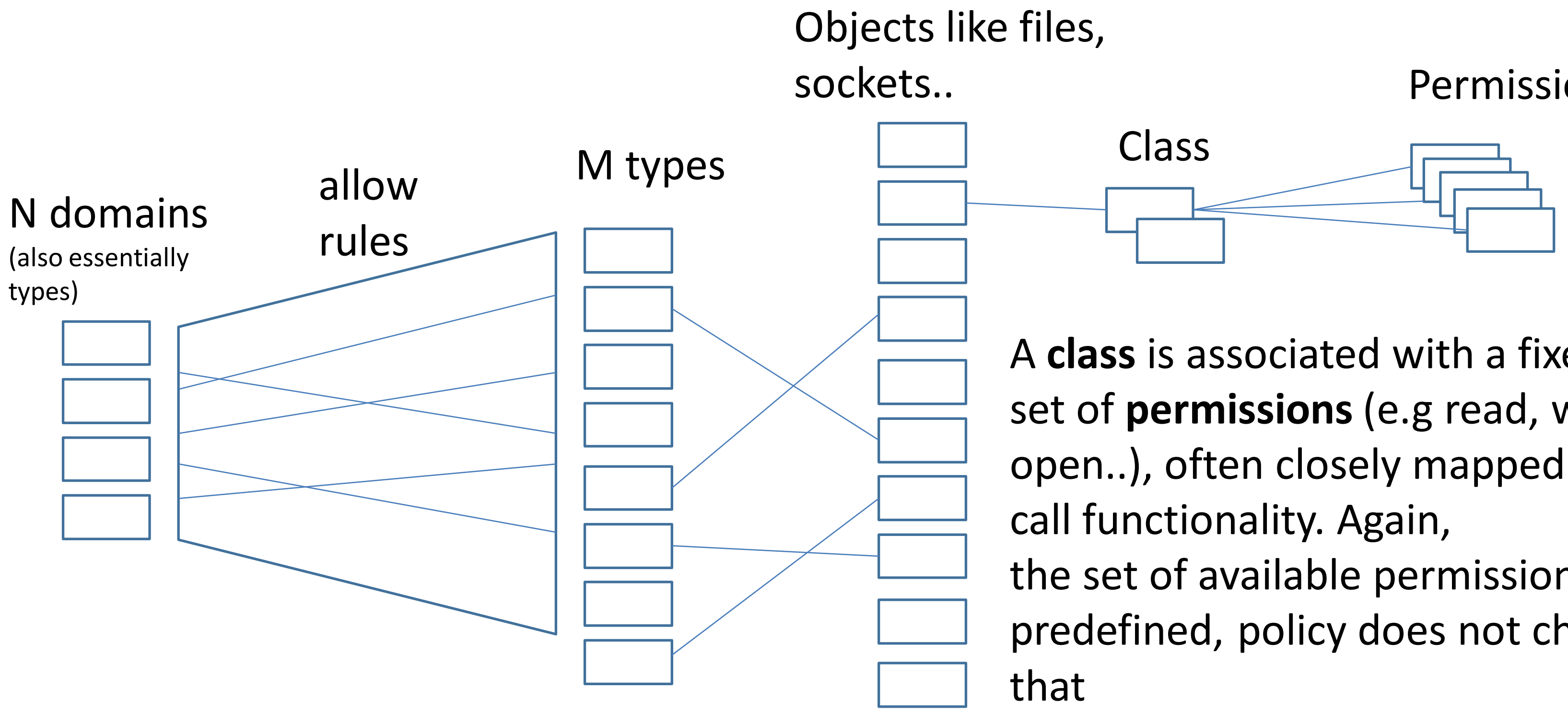
SEAndroid MMAC policy
(**key:** public keys for programs
values: 1) seinfo→domain,
2) android permission list)

Android subject identity:

- **Origin (RSA key)**
- **Package name**
- **UID (local)**
- **Filename (system services)**

Domain-type basic principle (dimension 2)

An object is **always** an OS primitive of some sort. This is not a policy issue, this is reality. An object therefore belongs to one or more **classes** which are predefined by the system proper. A class can be e.g. **file, socket, character device, ...**



A **class** is associated with a fixed set of **permissions** (e.g read, write, open..), often closely mapped to system call functionality. Again, the set of available permission is predefined, policy does not change that

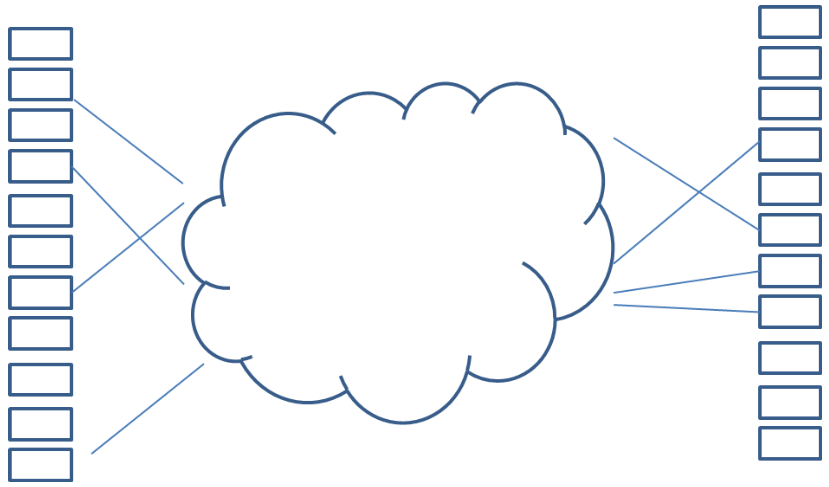
But! An allow rule includes also the class, and within that the permissions allowed by the rule. Class and permission names are ”well known” as defined by NSA / SELinux

A type is a domain when it is not a type

Not like this: “There is no practical difference between a type and a domain. The policy rules gives them significance. In particular, if an object has the same type as a process' domain, this means something only if the policy explicitly says so (it usually does). All types can be applied to any object since they are just names”

Subject types are domains **Object types are types**

A subject operates on an object, and as an optimization a domain operates on a type



SEAndroid types

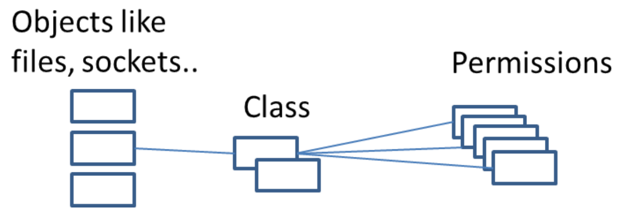
Computing system objects

SEAndroid types

Computing system objects

Classes and permissions (examples)

These are more or less defined by
http://selinuxproject.org/page/ObjectClassesPerms
(or in Android source code:
android / platform/external/sepolicy / master / . / access_vectors



A daemon or service provides Unix domain socket access for clients

unix_stream_socket
append, **bind, connect, create**, **write**, relabelfrom
ioctl, name_bind, sendto, recv_msg, send_msg
getattr, setattr, accept, getopt, **read**, setopt, shutdown, recvfrom
lock, relabelto, **listen**, **acceptfrom,** **connectto**, newconn

A device driver (say a serial port) provides device access

chr_file
append, create, execute, **write**, relabelfrom, link, unlink
ioctl, getattr, setattr, **read,** rename, lock, relabelto, mounton
quotaon, swapon, audit_access, entrypoint, execmod,
execute_no_trans, **open**

Class/ Permission illustration

Inside fs/open.c (sys_open syscall)

```
static int do_dentry_open(struct file *f, int (*open)(struct inode *, struct file *),
                          const struct cred *cred)
 ...
  error = security_file_open(f, cred);        ← class 'file', permission 'open'
  if (error) goto cleanup_all;
```

Inside security.c (LSM)

```
  int security_file_open(struct file *file, const struct cred *cred) {
     int ret;
     ret = security_ops->file_open(file, cred);
     if (ret) return ret;
     return fsnotify_perm(file, MAY_OPEN);
```

Inside selinux/hooks.c

```
  .file_open =  selinux_file_open,
```

SEAndroid policy allow rules

Since SEAndroid essentially is a single-user system, much of the complexity of users and roles is collapsed (one user, one role only)

A simple rule (there is in fact more to it) consists of:

ALLOW [domain] [type] : [class] {[allowed permissions]}

Subject
(a process in a domain)

Object(a file or other resource)

Resource class (from predefined set)

Class-specific permissions that are allowed by this rule

The [type] can be **self**. If applied to an attribute (explained later), it allows only access within each atomic domain, not among all pairs of domains covered by the attribute.

Allow rule examples (from a commercial Android phone)

1) allow **untrusteddomain hci_attach_dev : chr_file** { ioctl read getattr lock open } ;

2) allow **system_app dhcp_data_file : lnk_file** { ioctl read getattr lock open } ;

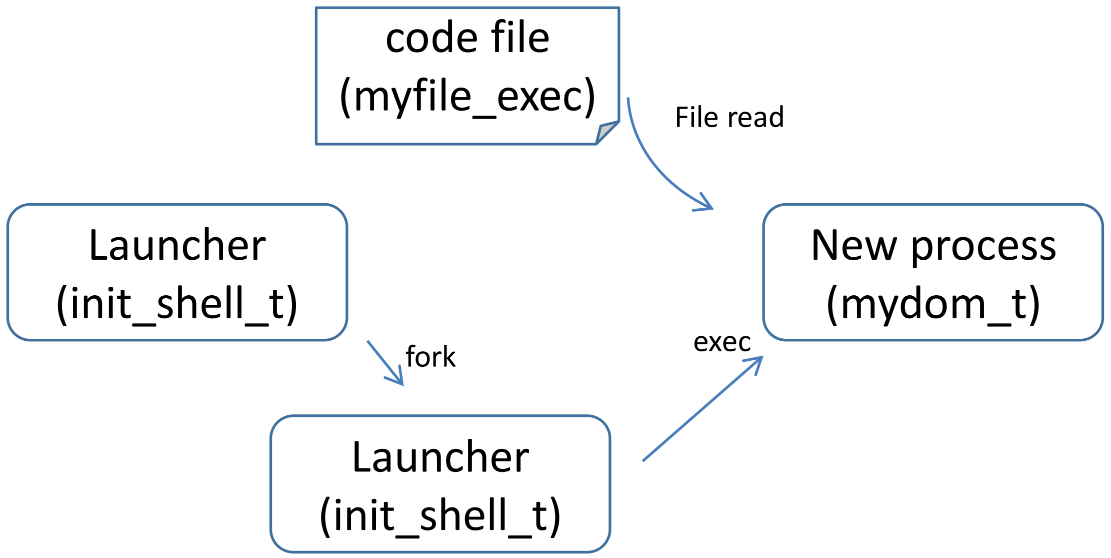
3) allow **untrusteddomain system_app : binder** { call transfer } ;

4) allow **logwrapper dhcp_system_file : dir** { ioctl read getattr search open } ;

5) allow **healthd healthd_exec : file** { read execute entrypoint } ;

Transition rule(sets)

Subjects and objects rules operate according to their domains/types : But how does a new process get into its domain? (http://selinuxproject.org/page/TypeRules#type_transition_Rule)



Intent ("what we want to happen")

type_transition init_shell_t myfile_exec: process mydom_t;

File execution right

allow init_shell_t myfile_exec: file execute;

File type is an entrypoint into a domain ("object firewall")

allow mydom_t myfile_exec: file entrypoint;

Process type needs transition right into a domain ("subject firewall")

allow init_shell_t mydomain_t: process transition;

File contexts / labeling & transition rule for files

The file contexts file is (in SEAndroid) the source of filesystem labeling

Example lines:

/var/log	u:object_r:**var_log_t**:s0
/dev/block/ram[0-9]*	u:object_r:ram_device:s0
/dev(/.*)?	u:object_r:device:s0
/system/bin/sh	u:object_r:shell_exec:s0

When we want to control the type of files being written (in a shared dir..)

Intent ("what we want to happen")

type_transition mydom_t var_log_t : file tmp_t;

Right to write to the directory (with type var_log_t)

allow mydom_t var_log_t: dir { add_name write search } ;

Right to write files of type tmp_t

allow mydom_t tmp_t: file { create write };

Context files have more attributes than allow rules

RegExp expansion

Security level (for MLS)

File contexts

```
/data/app(/.*)?    u:object_r:apk_data_file:s0
```

File / device name

user

Role (for RBAC)

Type for the object

SEApp contexts

(different parser, obviously..)

```
user=_app seinfo=myapp domain=myapp_app type=app_data_file
```

View of one SEAndroid policy

Different kinds of ”targets”

”target” permissions

```
Statistics for policy file: sepolicy
Policy Version & Type: v.26 (binary, mls)

  Classes:           84    Permissions:       249
  Sensitivities:      1    Categories:       1024
  Types:            646    Attributes:         44
  Users:              1    Roles:               2
  Booleans:           9    Cond. Expr.:         9
  Allow:         112271    Neverallow:          0
  Auditallow:         0    Dontaudit:         173
  Type_trans:       227    Type_change:         0
  Type_member:        0    Role allow:          0
  Role_trans:         0    Range_trans:         0
  Constraints:       63    Validatetrans:       0
  Initial SIDs:      27    Fs_use:             16
  Genfscon:          18    Portcon:             0
  Netifcon:           0    Nodecon:             0
  Permissives:        0    Polcap:              2
```

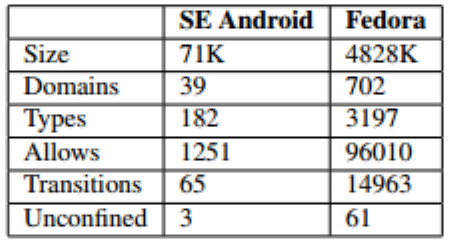
rules

Domains and types

”domain entry rules”

The complexity of the listed policy is close to that of Fedora

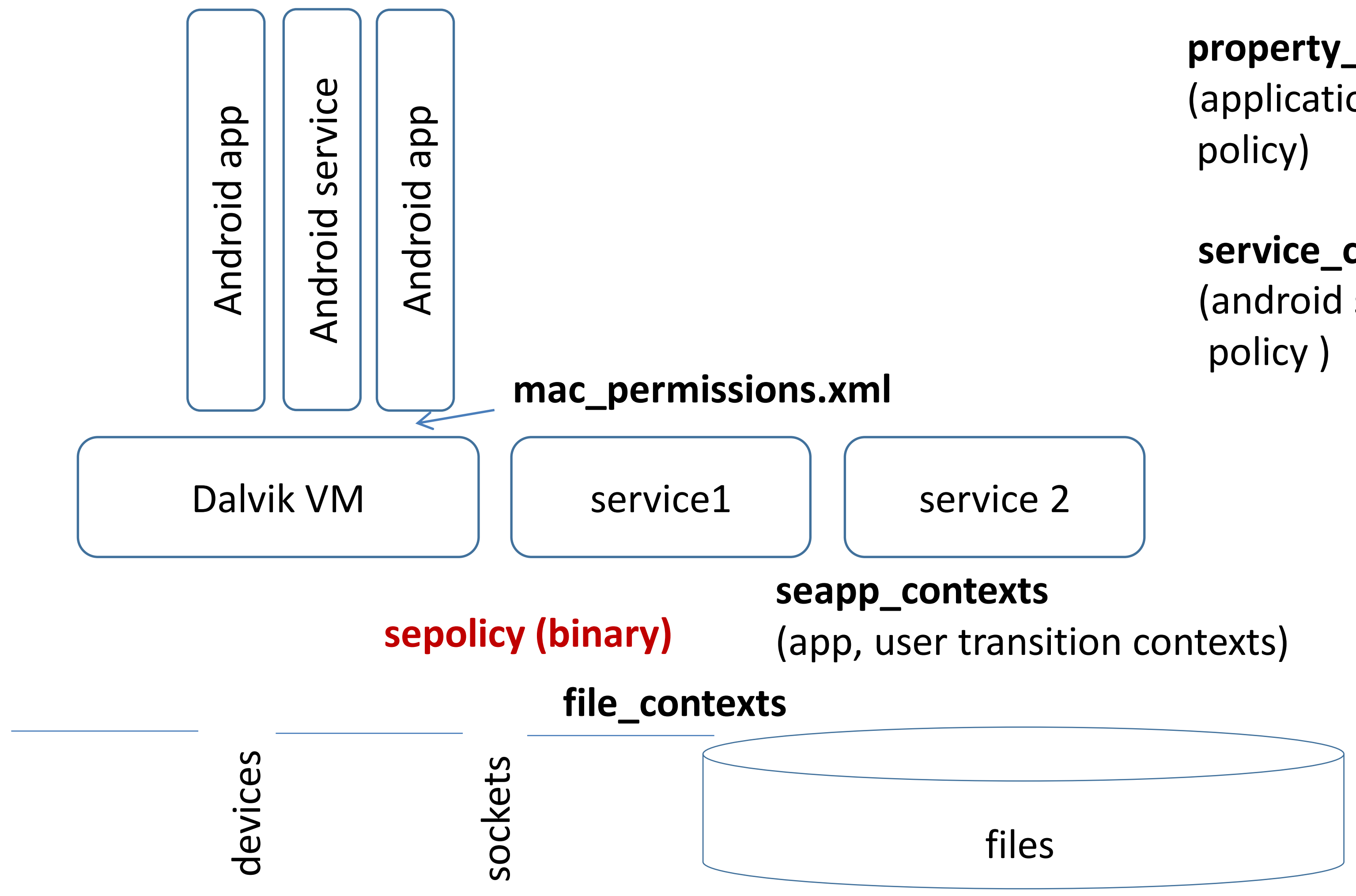
Smalley & al (SEAndroid)



	SE Android	Fedora
Size	71K	4828K
Domains	39	702
Types	182	3197
Allows	1251	96010
Transitions	65	14963
Unconfined	3	61

Table 3. Policy size and complexity.

System configuration view



property_contexts
(application-enforced policy)

service_contexts
(android service policy)

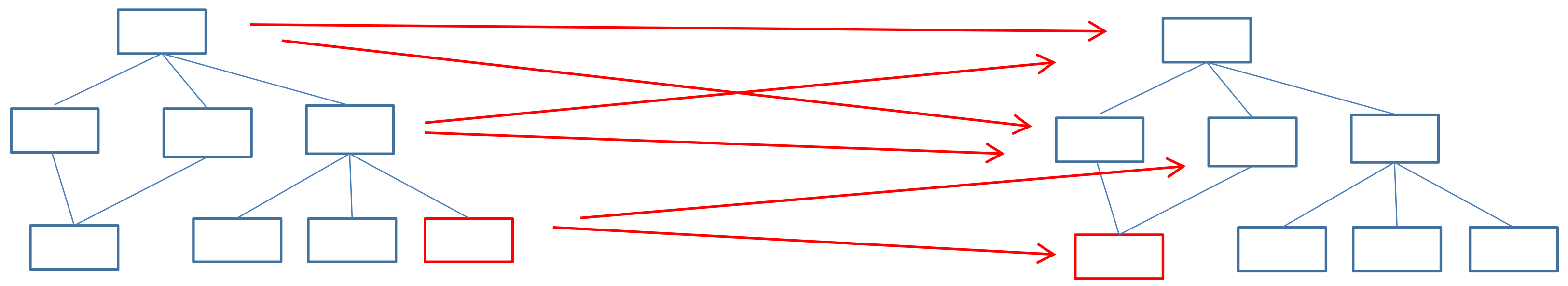
Attributes adds complexity to analysis

Attributes are groups of types (or domains). Rules can also be defined using attributes.

N domains in attributes
(also essentially types)

Examples of applicable allow rules

M types with attributes



It is typical that class attributes for overall access come from **different rules**. Say between a process and a resource, the read rule can come from attributes higher in the hierarchy, whereas the write access may be specific to individual domains and types.

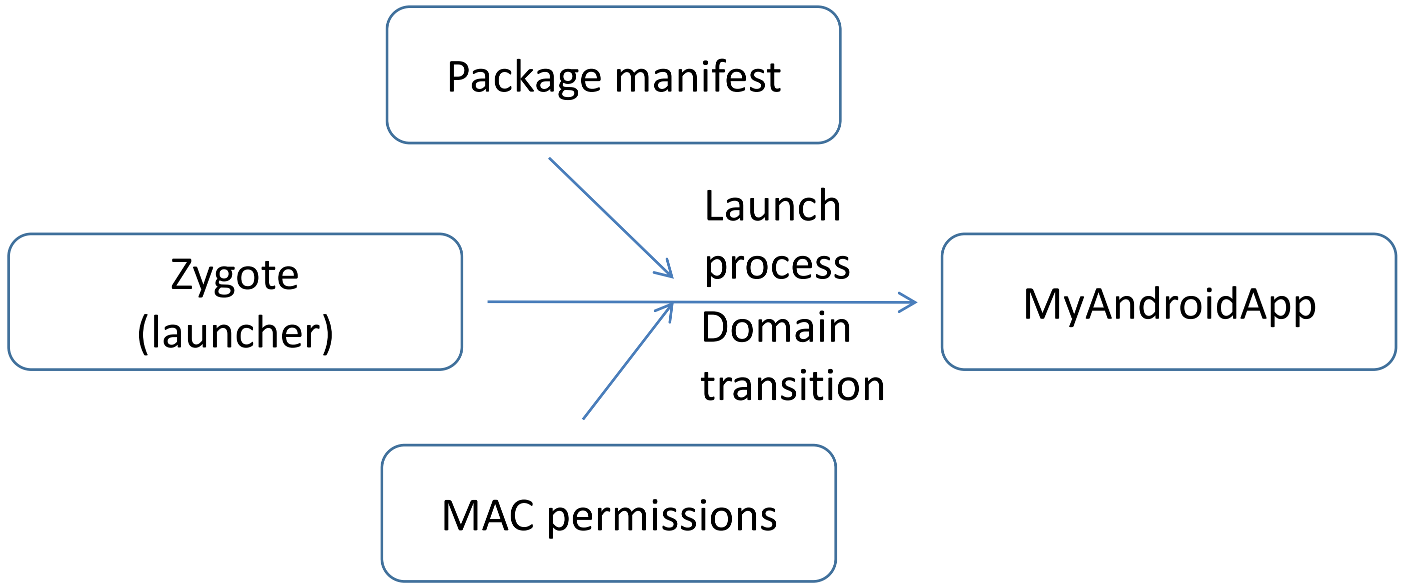
mac_permissions

(From comments in the XML file)

- A signature is a hex encoded X.509 certificate or a tag defined in keys.conf and is required for each signer tag.
- A signer tag may contain a seinfo tag and multiple package stanzas.
- A **default tag** is allowed that can contain policy for all apps not signed with a previously listed cert. It may not contain any inner package stanzas.
- Each signer/default/package tag is allowed to contain one seinfo tag. This tag represents additional info that each app can use in setting a SELinux security context **on the eventual process**.

```
<!-- Platform dev key in AOSP -->
<signer signature="@PLATFORM" >
  <seinfo value="platform" />
</signer>
<!-- All other keys -->
<default>
   <seinfo value="default" />
</default>
```

Launching with MAC



In Android, the PackageManager is a front-end to Installer data

```
String[] packageNames =
        getPackageManager().getPackagesForUid(uid);
try{
   PackageInfo pkgInfo = getPackageManager()
                         .getPackageInfo( packageNames[0],
                          PackageManager.GET_SIGNATURES);
 android.content.pm.Signature[] sigs = pkgInfo.signatures;
 Log.i("Signature", sigs[0].toCharsString());
...
```

”Example” mac_permissions.xml entry

```
<signer signature=“… 16ef8108a353a9f7300d06092a864886f70d01010b0500038
20101002ae36b53bd209841e4"><allow-all/><seinfo value=“myprog"/></signer>
```

For the MMAC part of the policy, the applications are bound by origin. I.e. the mac_permissions.xml contains a hundred or so public keys in x509/DER format encoded as XML as the example above. The fields are

-- **signature:** The x509 certificate containing the public part of the application signing key
-- **<allow-all/>:** The MMAC puts no extra constraints on the android permissions
-- **seinfo:** **A mapping to the domain of the policy**

As it happens we find the following line in the **seapp_contexts** policy file:

```
user=_app seinfo=myprog domain=myprog_app type=app_data_file
```

I.e. any application signed with a private RSA key corresponding to the public key mentioned in a certificate in the mac_permissions.xml that follows the template above, will be mapped to the **trustonicpartner_app** domain and be accessible (as an object) in accordance with the type **app_data_file**

Android Properties

A policy name resolver for property contexts
A database of configurations and status (like windows registry)

Listed in property_contexts file, e.g.
net.gprs u:object_r:net_radio_prop:s0
selinux. u:object_r:**security_prop**:s0
Associated with an allow rule, e.g.
allow system **security_prop** : property_service set

*)

Fixed

1) Property service calls **selabel_lookup (.., .., ”selinux.reload”, ..)**
2) Lookup finds, and returns *) → **u:object_r:security_prop:s0**
3) Service asks policy using **selinux_check_access()** for source request

SEAndroid protection is a moving target (1)

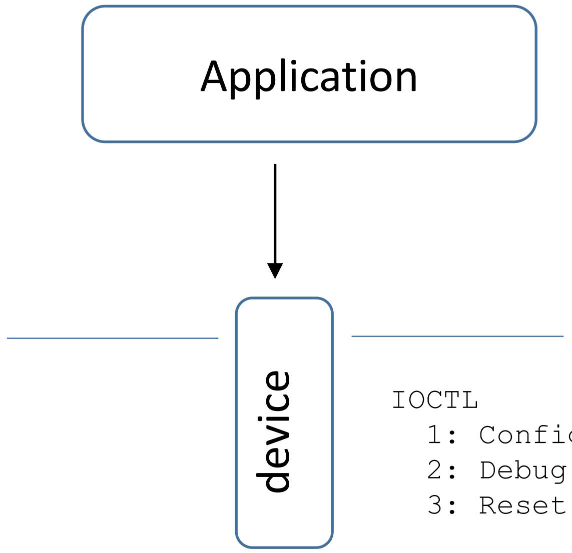
Addition (v30): "Extended permissions" == IOCTL protection

Available rules: allowxperm, dontauditxperm auditallowxperm and neverallowxperm

Examples:

```
allow src_t tgt_t : tcp_socket ioctl;
allowxperm src_t tgt_t : tcp_socket ioctl ~0x8927;

allow tee tee_device : chr_file open read write ioctl
allowxperm tee tee_device : chr_file ioctl 0x917
```



```
The ioctl command is a 32 bit number comprised of four fields,
number - sequence number of the command. 8 bits
type - magic number assigned to the driver. 8 bits
size - size of the user data involved. typically 14 bits (arch dep.)
direction - The direction of data transfer. typically 2 bits (arch dep.)
```

```
IOCTL
  1: Configure
  2: Debug (expose internal memory)
  3: Reset in case of error
```

https://selinuxproject.org/page/XpermRules

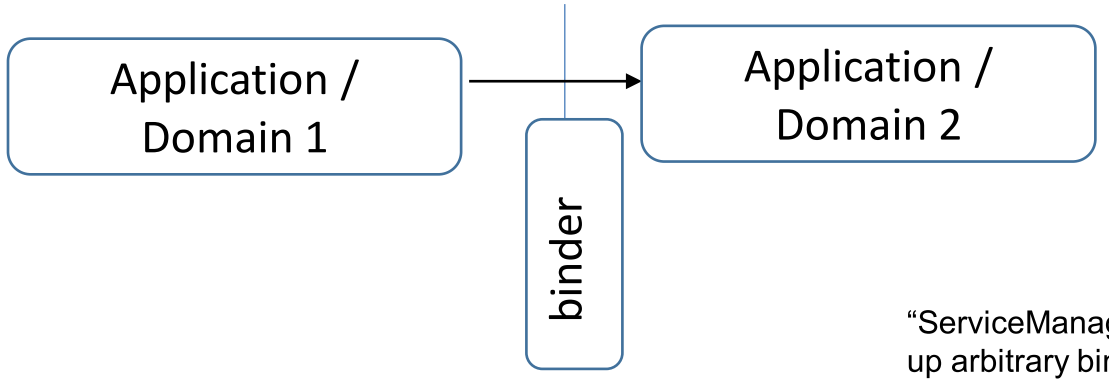
SEAndroid protection is a moving target (2)

Binder protection: "RPC" for Android applications

Has been available since day one, increasingly taken into use
If carefully used, can limit some obvious data flow attacks

Permissions for Binder class:

call	Perform a binder IPC to a given target process domain (can A call B?).
impersonate	(Not currently used)
set_context_mgr	Register self as the Binder Context Manager (aka service-manager)
transfer	Transfer a binder reference to another process (used by service-manager)



"ServiceManager prevents apps from looking up arbitrary binder services."

Permissions for Servicemanager class (userspace object):

add	Add a service
list	List services
find	Find services

http://kernsec.org/files/lss2015/lss2015_selinuxinandroidlollipopandm_smalley.pdf
http://selinuxproject.org/page/NB_SEforAndroid_1

SEAndroid protection is a moving target (3)

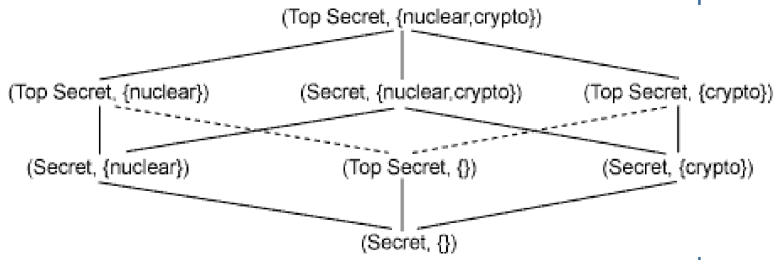
Multi-Level Security (used since v5 or thereabouts)**:**

Separation between apps and users on a policy level.
In general, MLS allows domains to access types based on **level** (ordered) and **category**(unordered)
What we see in SEAndroid are really categories

Example avc error (adb logcat):

```
type=1400 audit(0.0:7): avc: denied { search } for name="com.android.providers.downloads" dev="mmcblk0p23" ino=81932
scontext=u:r:system_app:s0 tcontext=u:object_r:app_data_file:s0:c512,c768 tclass=dir permissive=0
```

Google m4 macros (later) has support:
- mlstrustedsubject
- levelFrom=user, app, all



http://www.cs.cornell.edu/courses/cs5430/2012sp/mls.html

Process (domain) assignment:

```
            if (cur->levelFrom != LEVELFROM_NONE) {
char level[255];
switch (cur->levelFrom) {
case LEVELFROM_APP:
        snprintf(level, sizeof level, "s0:c%u,c%u",
                 appid & 0xff,
                 256 + (appid>>8 & 0xff));
        break;
case LEVELFROM_USER:
        snprintf(level, sizeof level, "s0:c%u,c%u",
                 512 + (userid & 0xff),
                 768 + (userid>>8 & 0xff));
        break;
case LEVELFROM_ALL:
        snprintf(level, sizeof level, "s0:c%u,c%u,c%u,c%u",
                 appid & 0xff,
                 256 + (appid>>8 & 0xff),
                 512 + (userid & 0xff),
                 768 + (userid>>8 & 0xff));
```

http://marc.info/?l=seandroid-list&m=143716685131494&w=2

Miscellaneous

neverallow rules (assertions)

- a way to shield off unwanted patterns

neverallow { domain -debuggerd -vold -dumpstate -system_server } self:capability sys_ptrace;

dac_override (and other capabilities)

- does show up in policies – one of the 32 linux capabilities
- overrides all ”standard” file permission checks

allow installd installd : capability { dac_override, sys_nice}

unconfined (macro expansion) – not available any more

- the macros are discussed later. A domain in the attribute unconfined, will be allowed all (class/permission) access to any type. Shows up as a domain in the final policy
- a testing tool, e.g. to determine interaction with DAC

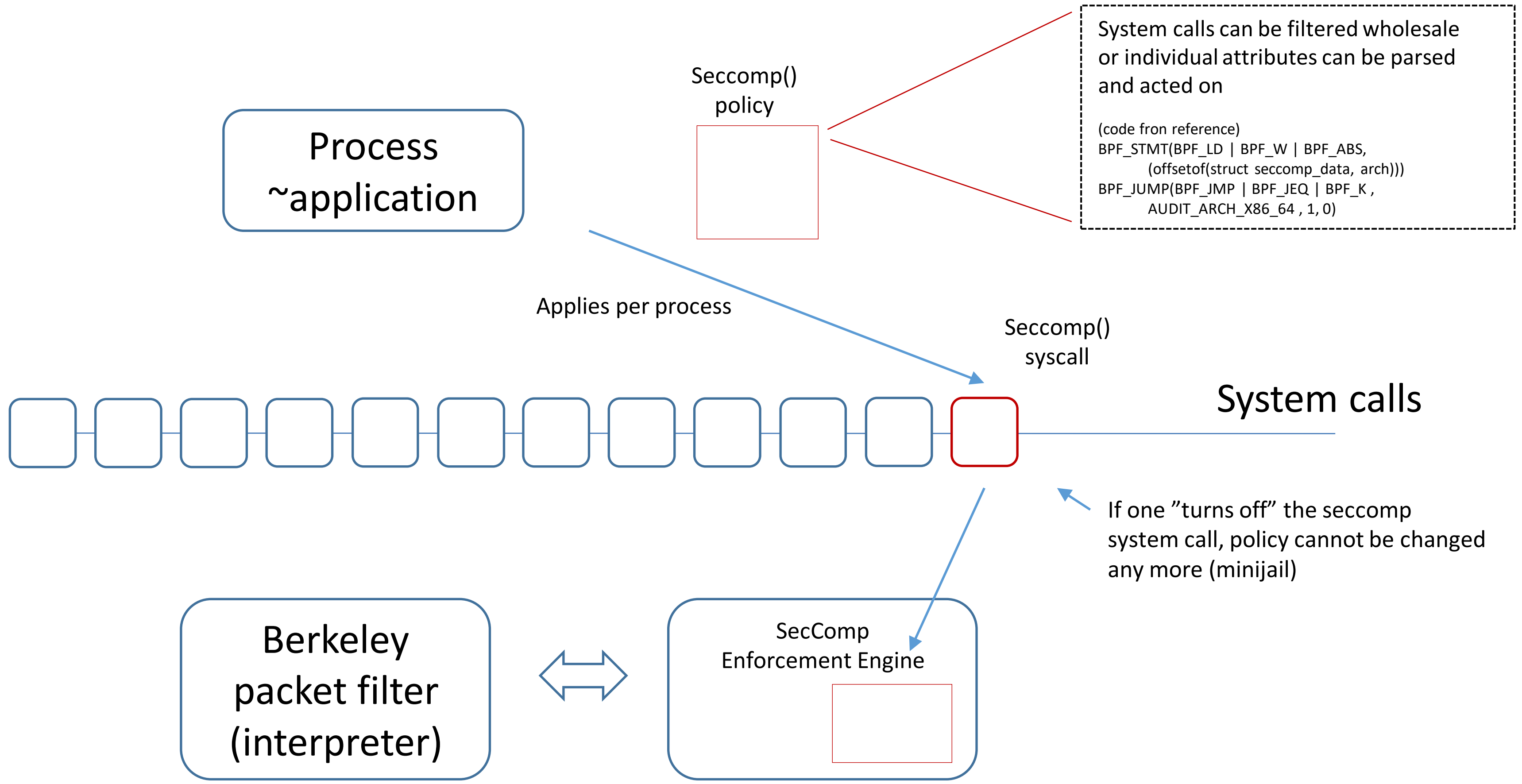
self (macro expansion)

allow netd self: { tcp_socket udp_socket} *

- As a target, denotes the domain itself, but not any parent attributes

SecComp as the ”next level of access control”
-- Android O forward – all apps have a seccomp filter

- ”Secure Computing” filter == SecComp, first version 2005
- ”Progammable policy for application on system call layer – very fine grained
- Depending on policy, may cause double-digit performance overhead


Seccomp() policy
Process ~application
Applies per process
Seccomp() syscall
System calls
If one ”turns off” the seccomp system call, policy cannot be changed any more (minijail)
Berkeley packet filter (interpreter)
SecComp Enforcement Engine

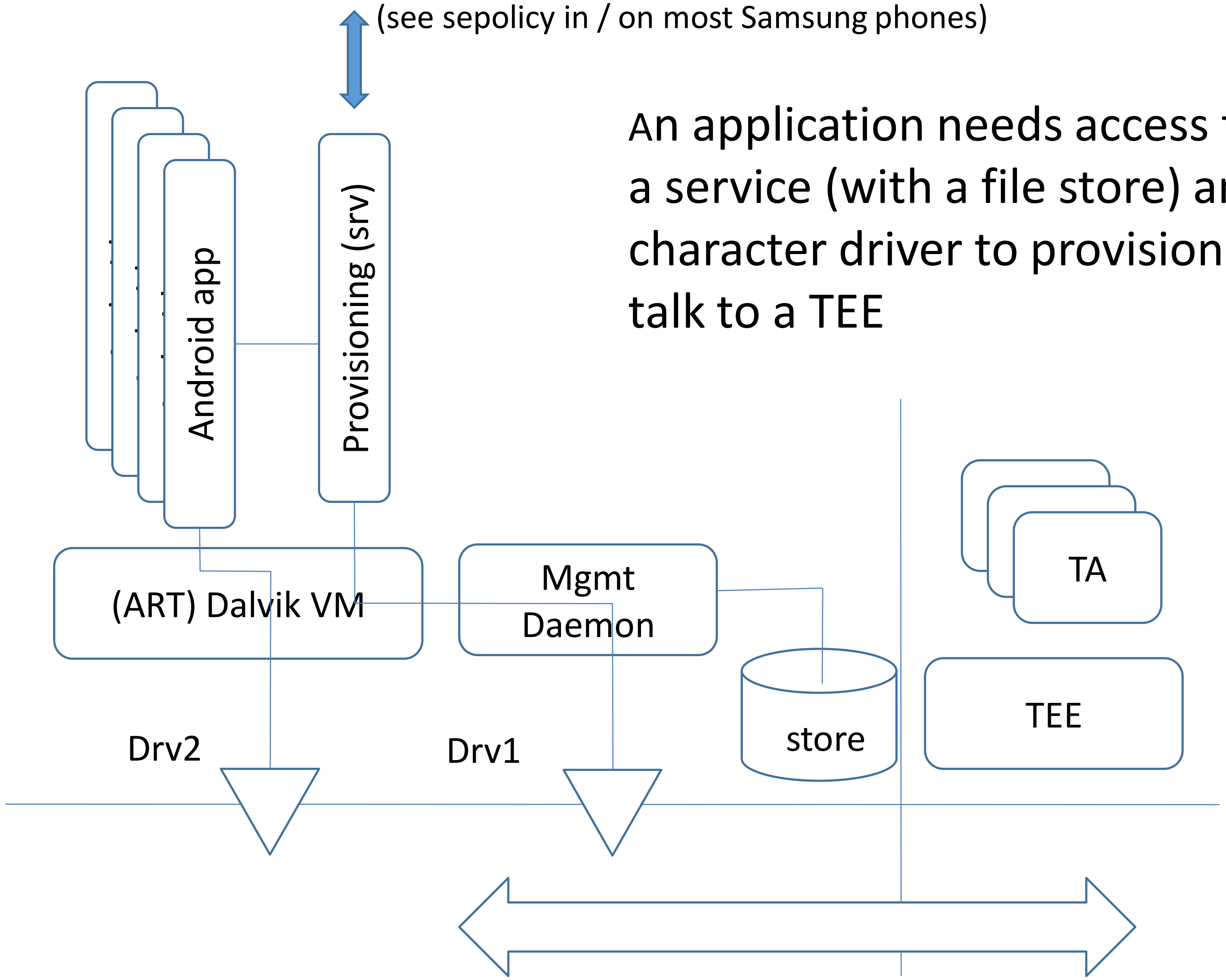
System calls can be filtered wholesale or individual attributes can be parsed and acted on

(code fron reference)

```
BPF_STMT(BPF_LD | BPF_W | BPF_ABS,
         (offsetof(struct seccomp_data, arch)))
BPF_JUMP(BPF_JMP | BPF_JEQ | BPF_K ,
         AUDIT_ARCH_X86_64 , 1, 0)
```

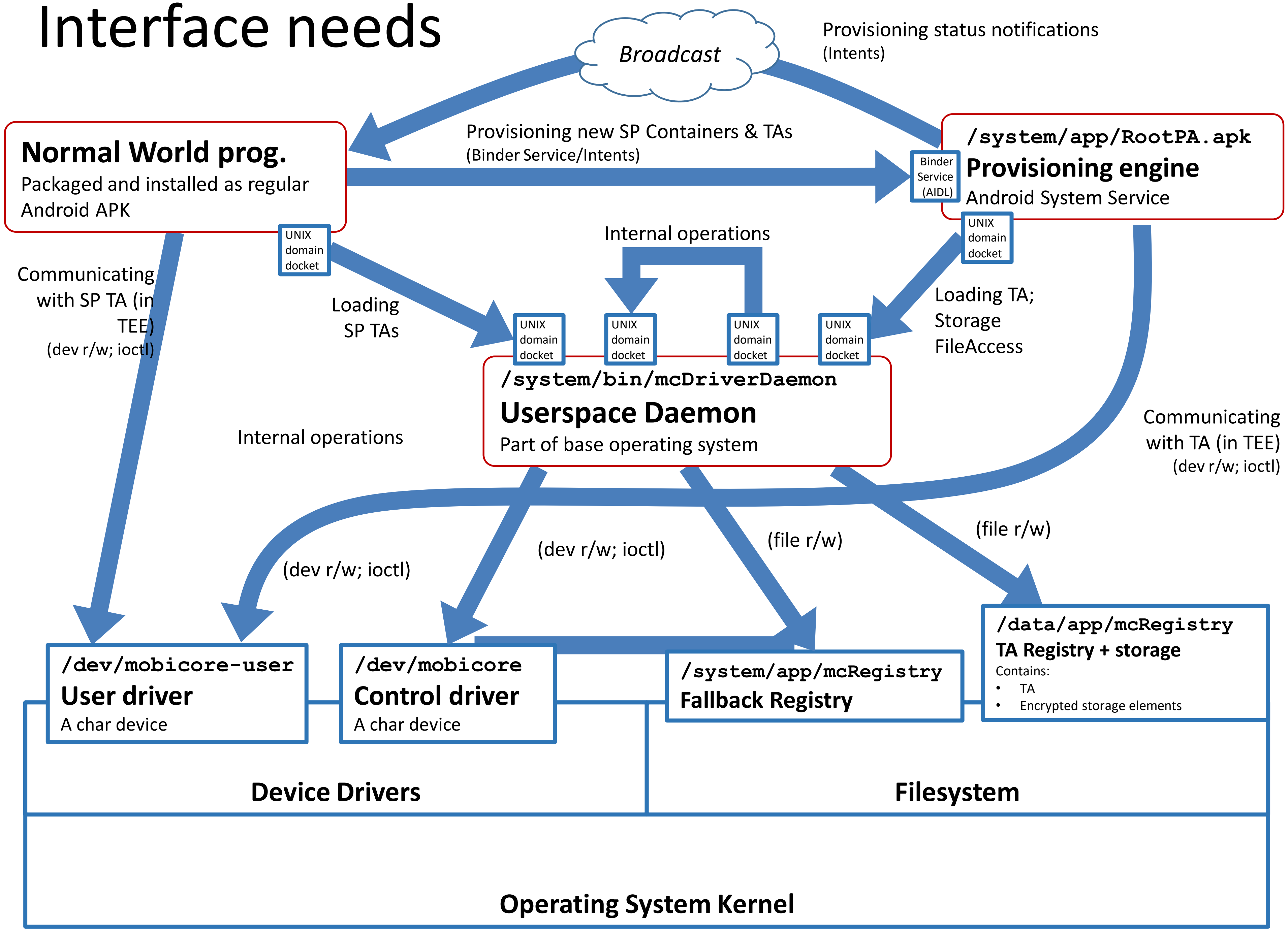
Introduction: https://lwn.net/Articles/656307/

”My” example use case (TEE in Samsung)

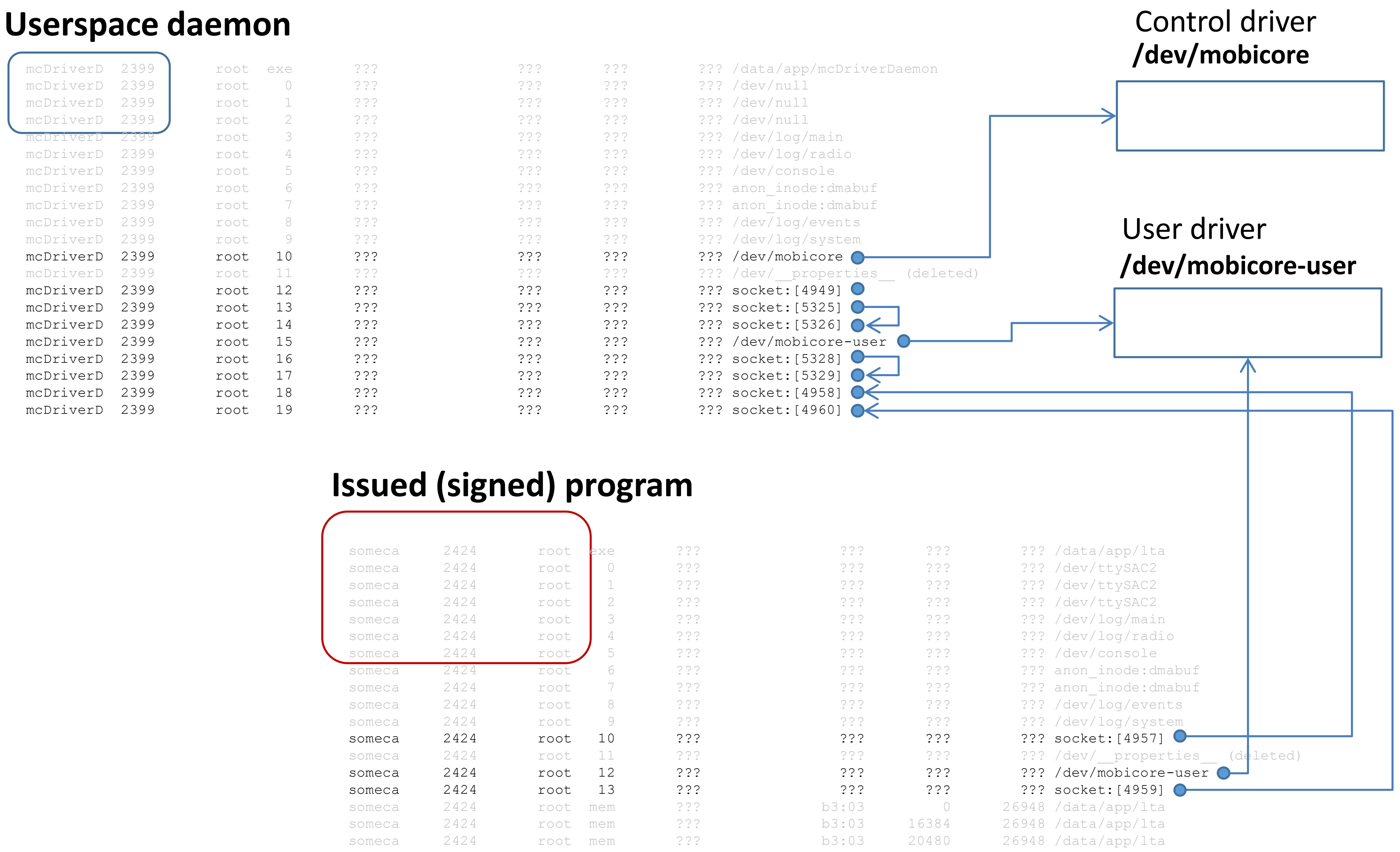


An application needs access to a service (with a file store) and a character driver to provision and talk to a TEE

Interface needs



Interfaces in practice



```
mcDriverD  2399     root  exe     ???          ???     ???      ??? /data/app/mcDriverDaemon
mcDriverD  2399     root    0     ???          ???     ???      ??? /dev/null
mcDriverD  2399     root    1     ???          ???     ???      ??? /dev/null
mcDriverD  2399     root    2     ???          ???     ???      ??? /dev/null
mcDriverD  2399     root    3     ???          ???     ???      ??? /dev/log/main
mcDriverD  2399     root    4     ???          ???     ???      ??? /dev/log/radio
mcDriverD  2399     root    5     ???          ???     ???      ??? /dev/console
mcDriverD  2399     root    6     ???          ???     ???      ??? anon_inode:dmabuf
mcDriverD  2399     root    7     ???          ???     ???      ??? anon_inode:dmabuf
mcDriverD  2399     root    8     ???          ???     ???      ??? /dev/log/events
mcDriverD  2399     root    9     ???          ???     ???      ??? /dev/log/system
mcDriverD  2399     root   10     ???          ???     ???      ??? /dev/mobicore
mcDriverD  2399     root   11     ???          ???     ???      ??? /dev/__properties__ (deleted)
mcDriverD  2399     root   12     ???          ???     ???      ??? socket:[4949]
mcDriverD  2399     root   13     ???          ???     ???      ??? socket:[5325]
mcDriverD  2399     root   14     ???          ???     ???      ??? socket:[5326]
mcDriverD  2399     root   15     ???          ???     ???      ??? /dev/mobicore-user
mcDriverD  2399     root   16     ???          ???     ???      ??? socket:[5328]
mcDriverD  2399     root   17     ???          ???     ???      ??? socket:[5329]
mcDriverD  2399     root   18     ???          ???     ???      ??? socket:[4958]
mcDriverD  2399     root   19     ???          ???     ???      ??? socket:[4960]
```

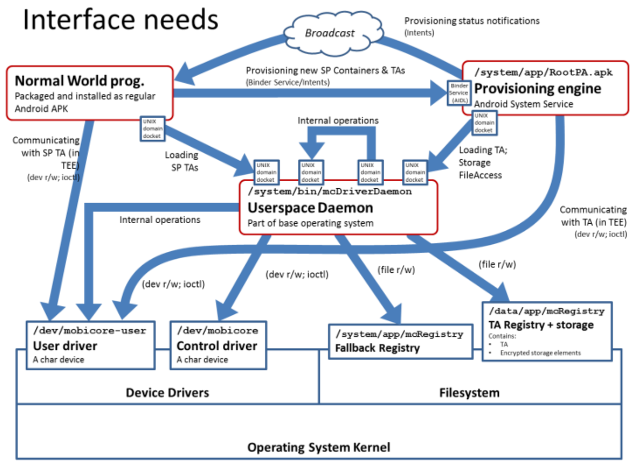
```
someca     2424       root  exe     ???            ???     ???      ??? /data/app/lta
someca     2424       root    0     ???            ???     ???      ??? /dev/ttySAC2
someca     2424       root    1     ???            ???     ???      ??? /dev/ttySAC2
someca     2424       root    2     ???            ???     ???      ??? /dev/ttySAC2
someca     2424       root    3     ???            ???     ???      ??? /dev/log/main
someca     2424       root    4     ???            ???     ???      ??? /dev/log/radio
someca     2424       root    5     ???            ???     ???      ??? /dev/console
someca     2424       root    6     ???            ???     ???      ??? anon_inode:dmabuf
someca     2424       root    7     ???            ???     ???      ??? anon_inode:dmabuf
someca     2424       root    8     ???            ???     ???      ??? /dev/log/events
someca     2424       root    9     ???            ???     ???      ??? /dev/log/system
someca     2424       root   10     ???            ???     ???      ??? socket:[4957]
someca     2424       root   11     ???            ???     ???      ??? /dev/__properties__ (deleted)
someca     2424       root   12     ???            ???     ???      ??? /dev/mobicore-user
someca     2424       root   13     ???            ???     ???      ??? socket:[4959]
someca     2424       root  mem     ???          b3:03       0    26948 /data/app/lta
someca     2424       root  mem     ???          b3:03   16384    26948 /data/app/lta
someca     2424       root  mem     ???          b3:03   20480    26948 /data/app/lta
```

Actual policy left as an exercise

One real-world policy for this setup, with support for virus checkers, file management, backups etc. has

6 types
852 allow rules

In the exercises you will get, one task is to define a policy for a set-up close to the one above. Simpler, of course...



As the multitude of classes and permissions makes policy writing ”by hand” tedious and error-prone, Google/Android has introduced a macro expansion tool, described next...

Google Policy Macros

An m4. macro set for representing common rule patterns in a a more readable format

tee.te

```
##
# trusted execution environment (tee) daemon
#
type tee, domain;
type tee_exec, exec_type, file_type;
type tee_device, dev_type;
type tee_data_file, file_type, data_file_type;
init_daemon_domain(tee)
allow tee self:capability { dac_override };
allow tee tee_device:chr_file rw_file_perms;
allow tee tee_data_file:dir rw_dir_perms;
allow tee tee_data_file:file create_file_perms;
allow tee self:netlink_socket create_socket_perms;
```

file_contexts

```
/dev/tf_driver u:object_r:tee_device:s0
/system/bin/tf_daemon u:object_r:tee_exec:s0
```

Recursion and relations between policy files

tee.te
allow tee *tee_data_file* : dir rw_dir_perms;
allow tee *tee_data_file* : file create_file_perms;

global_macros
define(`rw_file_perms', `{ r_file_perms w_file_perms }')
define(`create_file_perms', `{ create setattr rw_file_perms link_file_perms }')

define(`r_file_perms', `{ getattr open read ioctl lock }')
define(`w_file_perms', `{ open append write }')
define(`link_file_perms', `{ getattr link unlink rename }')

access_vectors

// contains a meta-definition of classes and their relations
// for validating the macro expansion?

te_macros

Contains template rules for some ’special’ operations

```
define(`domain_trans', `
# Old domain may exec the file and transition tonew domain.
allow $1 $2:file { getattr open read execute };
allow $1 $3:process transition;
# New domain is entered by executing the file.
allow $3 $2:file { entrypoint open read execute getattr };
# New domain can send SIGCHLD to its caller.
allow $3 $1:process sigchld;
# Enable AT_SECURE, i.e. libc secure mode.
dontaudit $1 $3:process noatsecure;
allow $1 $3:process { siginh rlimitinh };
')
```

```
define(`domain_auto_trans', `
# Allow the necessary permissions.
domain_trans($1,$2,$3)
# Make the transition occur by default.
type_transition $1 $2:process $3;
')
```

Similar macros exit for file type transition (based on e.g. dir), binder use (NEW), and special cases like DRM and debugging

Looking at things

The ”master” SEAndroid branch

https://android.googlesource.com/platform/
external/sepolicy/+/master

**/sepolicy , /file_contexts ... on
most Android 4.4 or 5 phones**

Google strongly recommends that the policy
files are kept in the root directory of the phone. However alternative locations
sometimes apply, especially if the policy is dynamically updatable
(http://seandroid.bitbucket.org/PolicyUpdates.html)

Tools for parsing the binary ’sepolicy’

The **apol** GUI (https://github.com/TresysTechnology/setools3/wiki)
seinfo / sesearch tools in setools (Ubuntu) package

E.g. **sesearch –allow** sepolicy gives a nice textual dump of all the allow rules
in the provided policy. **seinfo –x –t sepolicy** gives a list of types and parent attributes

SELinux Notebook
http://www.freetechbooks.com/efiles/selinuxnotebook/The_SELinux_Notebook_The_Foundations_3rd_Edition.pdf

Final words (if any)

1) SEAndroid is a ”tweaked” SELinux, with increased functionality especially for middleware (like the VM)

2) Earlier MAC systems in user devices include Symbian capabilities and CentOS SELinux. Android 4.4-> SEAndroid is destined to **become the most widespread and complex MAC** ever deployed on consumer devices.

3) SEAndroid policies are inevitably complex and writing them requires an understanding of both the target environment AND the policy framework simultaneously.

Executive reference

http://events.linuxfoundation.org/images/stories/pdf/lcna_co2012_smalley.pdf