PV204 Security Technologies

Multilevel security: isolation, confinement, security kernels, …

Zdeněk Říha & Petr Švenda

CROCS
Centre for Research on Cryptography and Security

Confinement

- Confinement problem
- Isolation: virtual machines, sandboxes
- Covert channels
  - Detection
  - Mitigation

Confinement problem

Problem of preventing a server from leaking information that the user of the service considers confidential

Total isolation

- Processes cannot communicate.
- Processes cannot be observed.
- Then the process cannot leak information.

- In practice not practical or not possible
- Processes use observable resources as CPU, filesystems, networks, …

Covert channels

A path of communication not designed to be used for communication

Covert channel examples

- Filesystems
- CPU usage
- Disk usage

Example of a covert channel

- CPU usage
- During each second
  - Process A either cycles (uses 100% of CPU) or leaves the CPU idle
  - Process B monitors the CPU usage
    - High CPU usage => transmission of bit 1
    - Low CPU usage => transmission of bit 0
  - Noise from other processes

Covert channel types

- Covert channels use shared resources.
- Covert storage channel
  - Based on an attribute of the shared resource
- Covert timing channel
  - Based on temporal or ordering relationship among multiple accesses to a shared resource

Key properties of covert channels

- Existence
  - Whether the channel exists…
- Bandwidth
  - How much information can be sent over the channel
- Noise
  - Noiseless covert channels
    - Available to sender and receiver only
  - Noisy covert channels
    - Also available to others
    - Need to minimize interference

Rule of transitive confinement

If a confined process invokes a second process, the second process must be as confined as the first one.

Covert channel detection

- Covert channels require sharing of resources
  - Sharing: which subjects can send, which subjects can receive information using that resource
- Covert flow tree
  - Porras, Kemmerer
    - Model of the flow of the information through shared resources

Illustration on the 12 following slides: Matt Bishop: Introduction to Computer Security, 2004

Example: Opening and locking files

- 3 attributes
  - locked: file is locked?
  - isopen: file is open?
  - inuse: set of process ID having the file open
- Functions:
  - read_acces(process, file): can *process* read *file*?
  - empty(s): Is *s* an empty set?
  - random(): return one of the arguments at random

Example: File routines

```
(* lock the file if it is not locked and
not opened; otherwise indicate it is
locked by returning false *)
procedure Lockfile(f: file): boolean;
begin
   if not f.locked and empty(f.inuse)
then
      f.locked := true;
end;
(* unlock the file *)
procedure Unlockfile(f: file);
begin
   if f.locked then
      f.locked := false;
end;
(* say whether the file is locked *)
function Filelocked(f: file): boolean;
begin
   Filelocked := f.locked;
end;
```

```
(* open the file if it isn't locked and
the process has the right to read the
file *)
procedure Openfile(f: file);
begin
   if not f.locked and
         read_access(process_id, f) then
      (* add process ID to inuse set *)
      f.inuse = f.inuse + process_id;
end;
(* if the process can read the file, say
if the file is open, otherwise return a
value at random *)
function Fileopened(f: file): boolean;
begin
   if not read_access(process_id, f) then
      Fileopened := random(true, false);
   else
      Fileopened := not isempty(f.inuse);
end
```

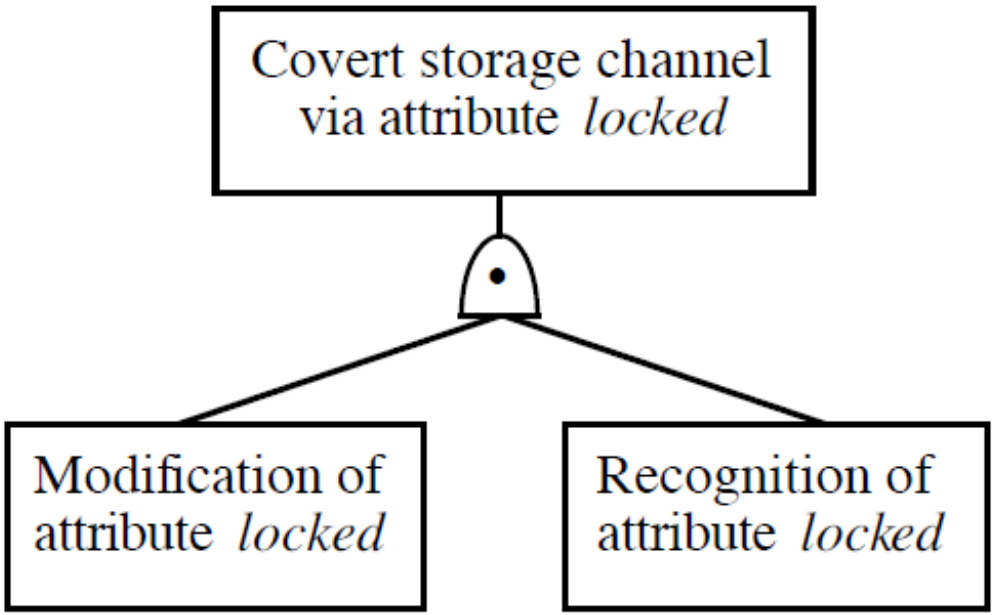
Example: Overview of attributes and operations

Function/ attributes	Lockfile	Unlockfile	Filelocked	Openfile	Fileopened
References	*locked, inuse*	*locked*	*locked*	*locked, inuse*	*inuse*
Modifies	*locked*	∅	∅	*inuse*	∅
Returns	∅	∅	*locked*	∅	*Inuse*

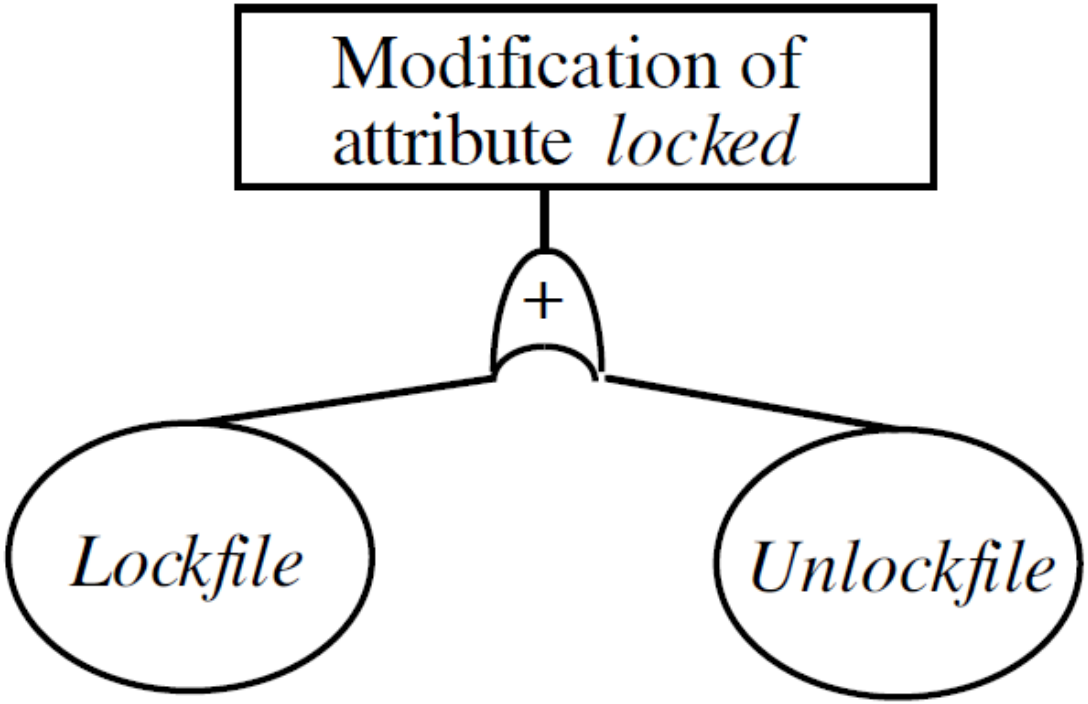
Covert tree flow: Constructing the tree

- The tree will contain information about possible attribute
  - Modification
  - Recognition
    - Direct
    - Inferred (via)
- Let's construct the tree for the attribute *locked*
- The goal is to establish a covert storage channel via the attribute *locked*

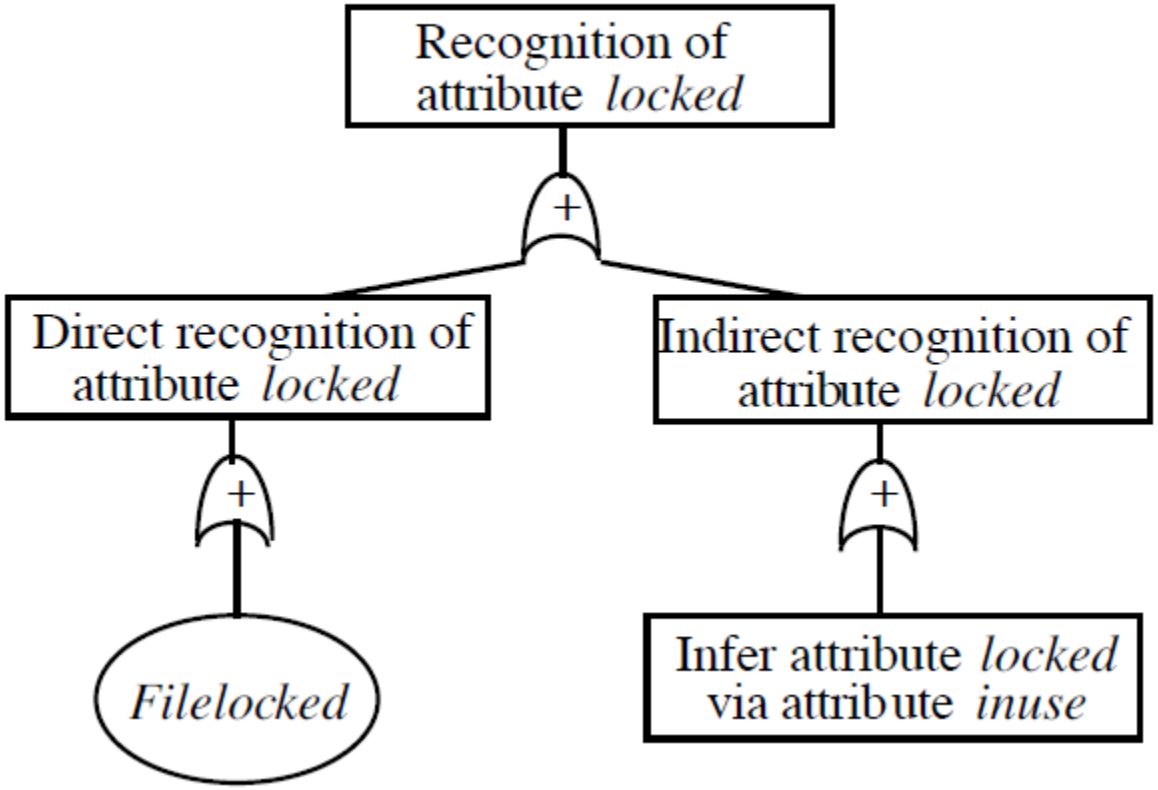
Covert tree flow: first step



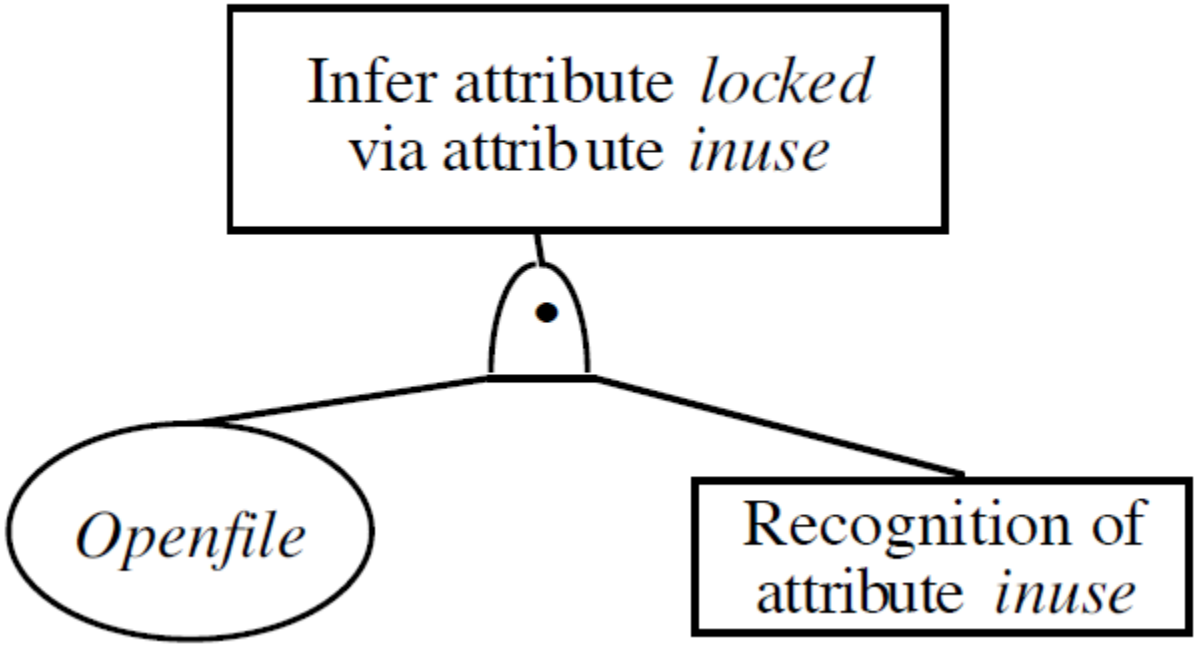
Covert tree flow: second step



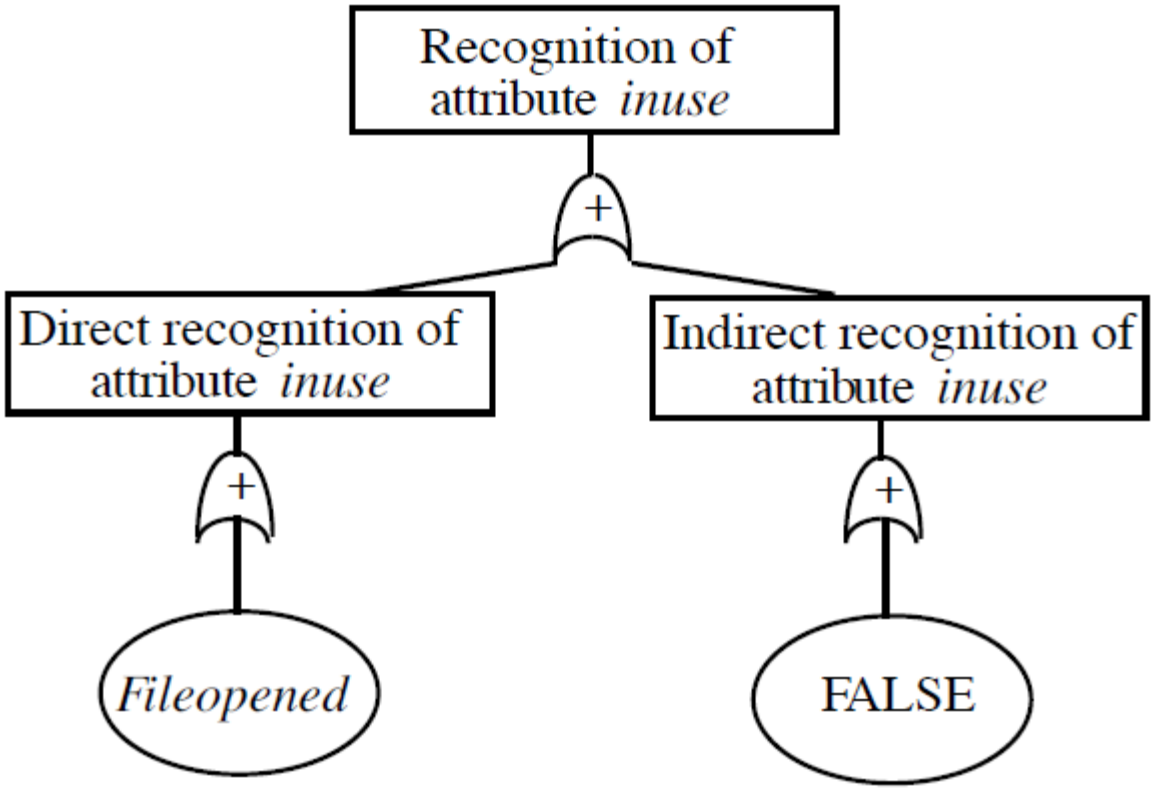
Covert tree flow: third step



Covert tree flow: fourth step

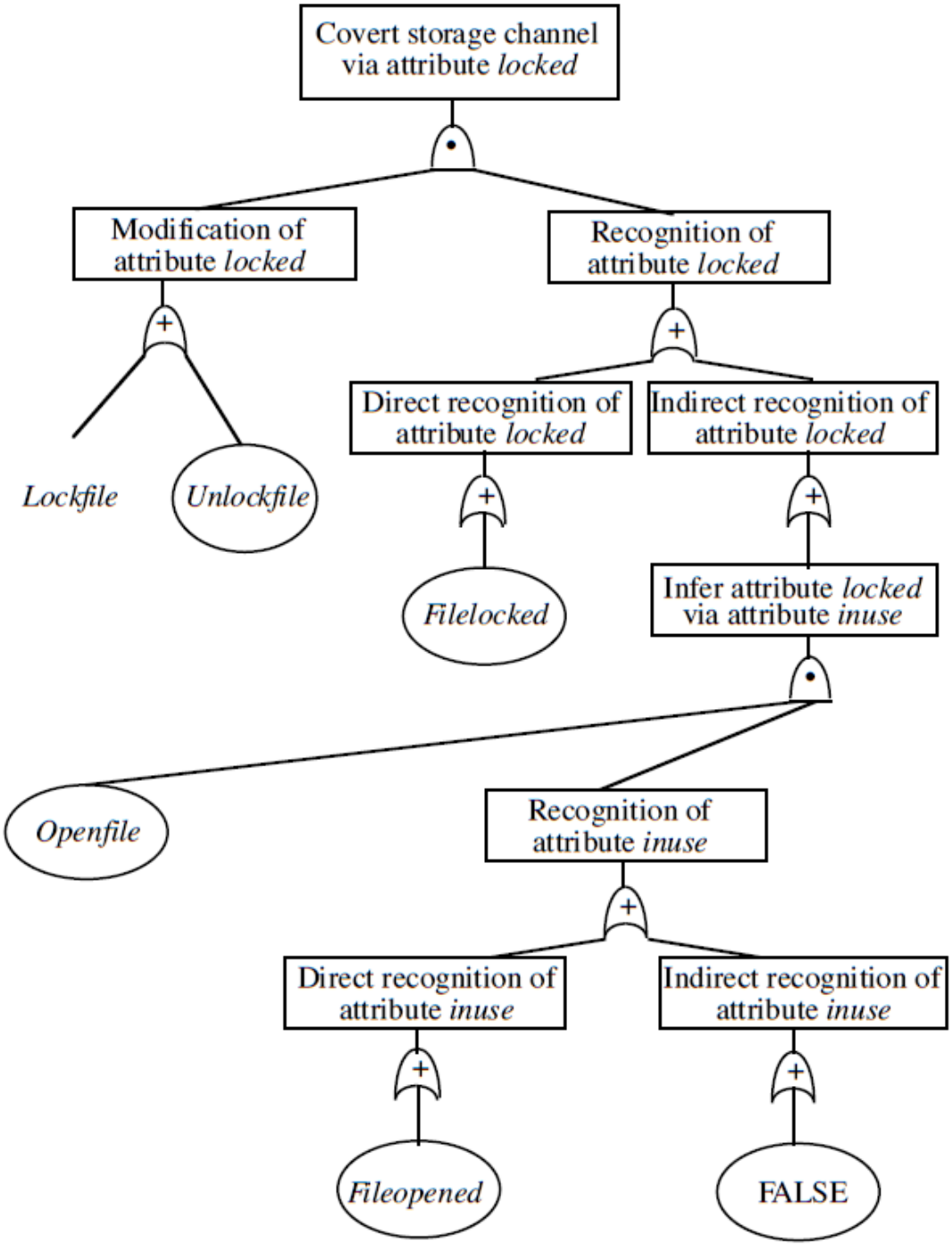


Covert tree flow: fifth step



Covert tree flow

- The final tree
  - For the attribute *locked*



How to recognize covert channels

- File locking example
  - Find sequences of operations that modify attributes
    - Example: (Lockfile), (Inlockfile)
  - Find sequences of operations that recognize modifications of attributes
    - Example: (Filelocked), (Openfile, Fileopened)

Covert channel commands

- Sequences with first element from first list and second element from second list
- File locking example
  - *Lockfile*, then *Filelocked*
  - *Unlockfile*, then *Filelocked*
  - *Lockfile*, then *Openfile*, then *Fileopened*
  - *Unlockfile*, then *Openfile*, then *Fileopened*

Covert channel mitigations

- Uniform/fixed amount of resources to each process.
  - CPU
  - Disk space
  - Disk access (speed)
- Injecting randomness into using resources

- The aim is to reduce the bandwidth
  - The drawback is often suboptimal performance

Isolation

- Resource sharing between users is the key cause of security and privacy issues [James Anderson].
- Execution of programs must be controlled to build a secure resource sharing system.
- The term “Reference Monitor” introduced many years ago…
- Isolation closely related to the reference monitor notion…

Taxonomy of isolation techniques [Viswanathan]

- Language based techniques
  - Type systems
    - E.g. Java, Modula-3
  - Certifying compilers/components
    - E.g. Proof Carrying Code (PCC)
- Sandboxing techniques
  - Instruction Set Architecture based
    - E.g. Software Fault Isolation (SFI)
  - Application Binary Interface based
    - E.g. Janus, MAPBox
  - Access Control based
    - E.g. chroot, BSD jail

http://www.arunviswanathan.com/survey_isolation_techniques.pdf

Taxonomy of isolation techniques

- Virtual Machines based isolation
  - Process virtual machines
    - E.g. Java VM
  - Hypervisor virtual machines
    - E.g. XEN, VMWare GSX Server
  - Hosted virtual machines
    - E.g. VMWare Workstation, MS Virtual PC
  - HW virtual machines
    - E.g. Intel VT-x, AMD-V, KVM

Taxonomy of isolation techniques

- OS-kernel based isolation
  - The most traditional way of isolation
  - E.g. common monolithic kernels, Mach microkernel
- Hardware based isolation
  - Strongest form of isolation
  - E.g. MMU (virtual address space)

Multilevel systems (MLS)

- Classification of data
- Security clearance of users & need to know
- Mandatory access control
- Mandatory security policy enforced.
- E.g. Bell-LaPadula model for data confidentiality
- E.g. Biba model for data integrity

MLS Systems

- MLS covered in PV157
- Self study:
  - Anderson: Security engineering, chapter 8 (MLS):
  - http://www.cl.cam.ac.uk/~rja14/Papers/SEv2-c08.pdf
  - Mandatory reading: pages 239 – 250
  - Additional reading: complete chapter 8 (+ pages 251-273)
  - Additional reading: some of the classified Snowden docs

MLS systems

- Noninterference
  - Goguen & Meseguer, 1982
  - Actions on higher levels have no effect on what can be seen on lower levels
- Nondeductibility
  - Sutherland, 1986
  - On lower levels nothing can be deduced with 100 percent probability about the input on higher levels.
    - Users on lower level can see actions of higher level users, just not to understand them.
    - *But isn't 99% probability sufficient anyway…*

MLS: Bell-LaPadula

- Designed in 1973
- Aimed at data confidentiality
- Classification
  - Of subjects (users) – based on their trustworthiness
  - Of objects (data, files, clipboard) – based on their confidentiality
  - Labels: Hierarchical level + set of categories
    - Unclassified
    - Confidential
    - Secret
    - Top Secret
  - E.g. [Secret, {Crypto}]

MLS: Bell-LaPadula

- The Bell-LaPadula model enforces 2 properties:
  - "Simple security property" - No Read Up (NRU)
    - Processes cannot read data at higher levels (users are not allowed to access more secret data than they are cleared for)
  - "* -property" – No Write Down (NWD)
    - Processes cannot write data to lower levels (not to leak confidential data to unclassified files, e.g. by malware)
- Bell-LaPadule is build on top of a discretionary access control system (access rights matrix)

MLS: Weak points

- System Z
  - Asking admin to declassify all files ☺
- Overclassification
  - High watermark principle
- Lowering classification
  - Over time documents get less confidential
  - “Trusted editor”, “Trusted subject”
    - To be able to edit a secret document from a top secret document
  - Implementation of MLS systems
    - And applications (adjustments for MLS needed)
  - Using MLS systems – users
    - Classified clipboard, …

Securing Linux Kernel

- SELinux (policy based) developed by NSA
- In 2001 inclusion in standard kernel rejected by Linus Torvalds as SElinux is not the only and ultimate security model.
- Then the Linux Security Modules (LSM) framework was created.
- LSM inserts hooks at points in the kernel where a user-level system call is about to result in access to an important internal kernel object.

LSM framework

- Standard part of Linux kernel since 2.6
- Solving the problem of fine-tuning access control and avoiding complex changes of the mainstream Linux kernel.
- Modular approach.
- Currently accepted modules in the official Linux kernel:
  - AppArmour, SElinux, Smack, TOMOYO Linux

AppArmour

- Application Armour
- Implements Mandatory Access Control (MAC)
- Path name access control scheme to confine applications

- Seen as a simpler alterative to SELinux
  - Overhead estimated to 1-2% as opposed to 7%

SELinux

- Mandatory Access Control system supporting:
  - least privilege, confidentiality, integrity, isolation, information flow control; exploit containment
- Allows the composition of multiple security models under a single analyzable policy
- Currently ships with: Type Enforcement, RBAC and MLS/MCS
- Very flexible, meets very wide range of usage scenarios

SMACK

- Simplified Mandatory Access Control Kernel
- Labeling of subjects and objects
  - System labels define hierarchical limits
  - Admin-defined labels can be any short string
  - Policy is written as triples:
    - Subject Object [–rwxa]

TOMOYO

- Path-based MAC system
- Supports automatic real-time policy generation
- Enforces previously observed behavior (in learning mode)
- **Domains** are trees of process invocation
- Rules apply to domains