# SELinux

Igor Vuk

# What are we going to talk about?

- Overview
- How it works
- Everything else

# Overview (1/8)

- http://selinuxproject.org/page/Main_Page
- Security enhancement to the GNU/Linux OS
- Mandatory Access Control(MAC) framework
- Shipped by Fedora, RHEL{4,5,6}, Debian, ...
- Provides the mechanism for supporting access control security policies, including US DoD mandatory access controls, through the use of LSM in the Linux kernel
    - http://en.wikipedia.org/wiki/SELinux
- Included in mainline Linux, as of 2.6

# Overview (2/8)

- The goal is to create a better form of system security
  - Tries to protect you from bugs in applications
- The restrictions SELinux imposes are mandatory
  - Default policy is deny
  - There is no equivalent of a root user
  - Access rules depend on attributes given to a certain subject and object pair
- The protection stacks with DAC
  - Both are required for an action to be allowed

# Overview (3/8)

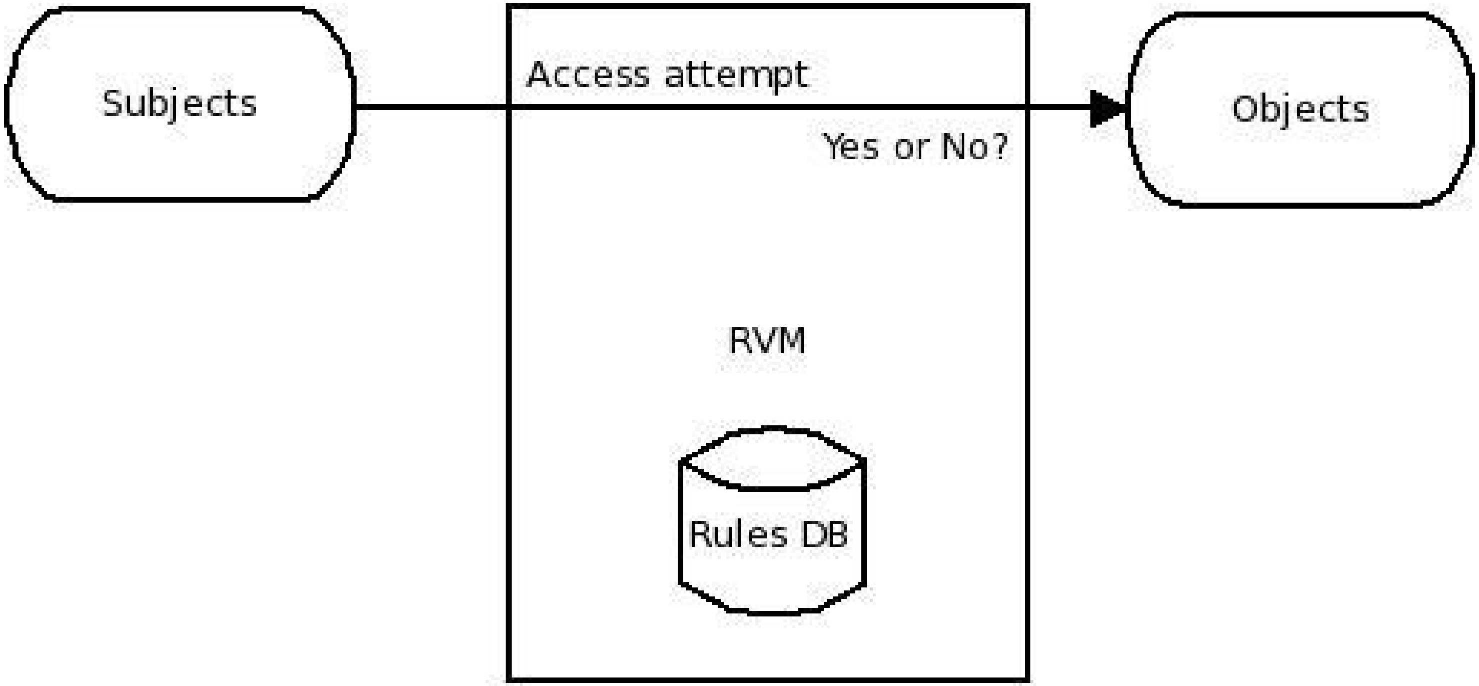
- Relies on several basic concepts
  - Subjects (i.e. processes)
  - Objects (i.e. files, folders, sockets...)
  - Access vectors (rules)
- Attributes of subjects and objects are called security contexts
- A combination of kernel modules and user-space tools
  - Don't forget about the reference policy
- Licensed under the GPL licence

# Overview (4/8)

- Mayer, MacMillan, Capman, "SELinux by Example: Using Security Enhanced Linux"
- **Reference monitor concept**
  - Subjects
  - Objects
  - Reference validation mechanism
    - Tamperproof
    - Non-bypassable
    - Verifiable

# Overview (5/8)

- Reference monitor concept
  - This is where it all started



# Overview (6/8)

- This is closely interlinked with military-funded work for developing a policy that would be secure enough for classified government documents
- **That's how we got MultiLevel Security (MLS)**
  - Based on Bell-LaPadula model
    - Bell, LaPadula, "Secure Computer Systems: Unified Exposition and MULTICS Interpretation"
- Top secret, secret, confidential, unclassified
  - No read up
  - No write down
  - Write up
  - Read down

# Overview (7/8)

- MLS has been implemented plenty of times
  - Trusted Tru64, Trusted HP-UX, Trusted AIX, Trusted Solaris
- Trusted Solaris components made their way into Solaris 10 and 11
  - RBAC is turned on by default in Solaris (10 and newer)
  - There is no way to turn it off
- **SELinux does not use MLS by default**
  - Type enforcement (TE)
  - Stuff I'm talking about is mostly based on the targeted policy

# Overview (8/8)

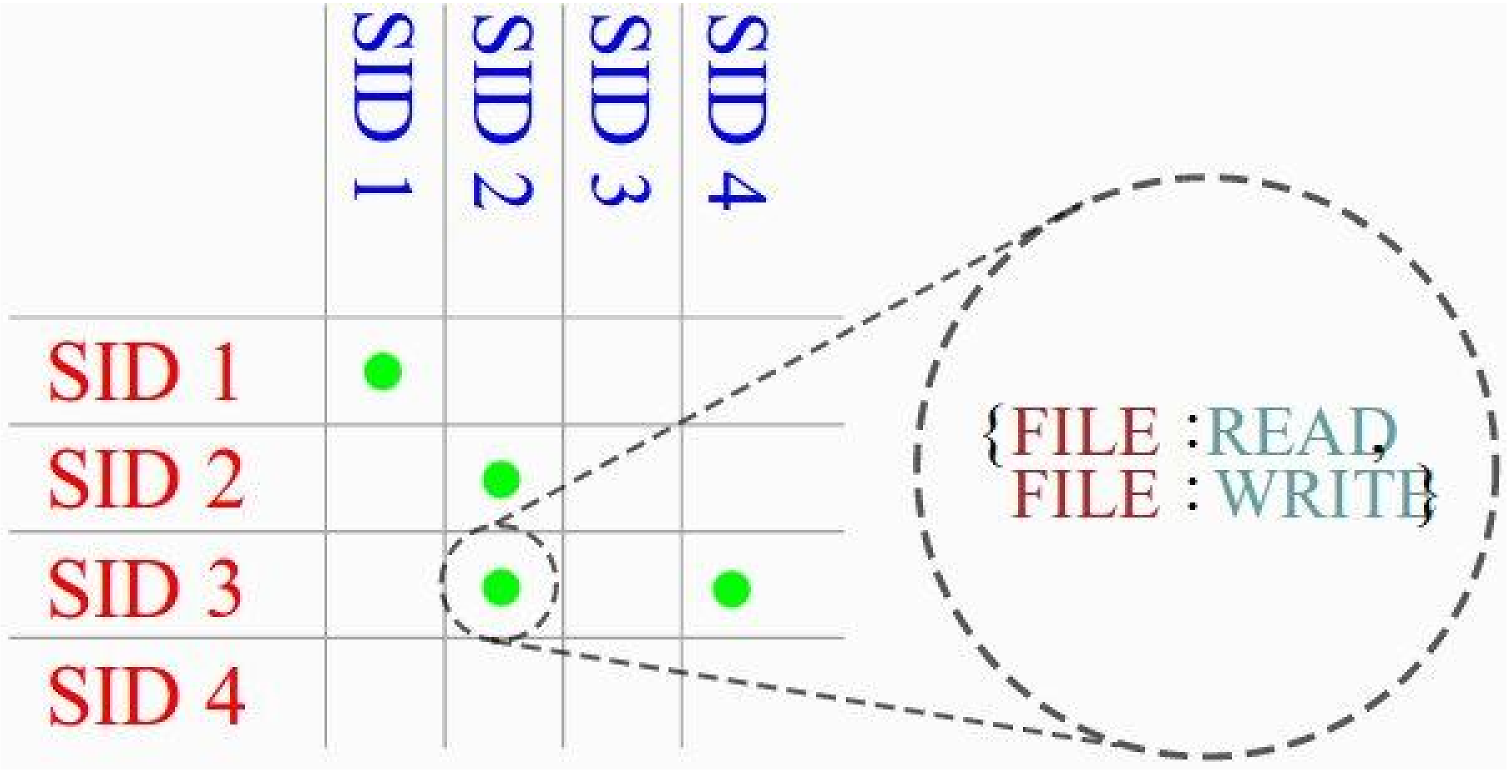
- It all started with DTMach
- After DTMach we got FLASK
    - Here TE showed up
- Then we got Linux Security Module (LSM)
- FLASK got ported to LSM "backend"
- **SELinux is a reference implementation of the FLASK security architecture**
- http://www.nsa.gov/research/selinux/faqs.shtml
- SELinux was merged into mainline

# How it works (1/11)

- http://www.imperialviolet.org/2009/07/14/selinux.html
- **May X do Y to Z?**
  - Subjects (u32 SIDs)
  - Objects (u32 SIDs)
  - Actions
    - Classes (FILE, TCP_SOCKET,...)
    - Permissions (READ, WRITE, ENTRYPOINT,...)
- Security policy

# How it works (2/11)

- http://www.imperialviolet.org/2009/07/14/selinux.html



| | SID 1 | SID 2 | SID 3 | SID 4 |
|---|---|---|---|---|
| SID 1 | ● | | | |
| SID 2 | | ● | | |
| SID 3 | | ● | | ● |
| SID 4 | | | | |

{FILE :READ,
FILE : WRITE}

# How it works (3/11)

- Access vector cache (AVC)
  - A hash map
  - From (subject, object, class)
  - To allowed permissions
  - Queried when kernel needs to make security decisions

# How it works (4/11)

- Security identifiers (SIDs)
  - Subjects and objects can be complex
  - They are reduced to an identifier (via a table)
  - That identifier is called a SID
  - A SID table maps from a SID to a matching security context (the mapping works both ways)

# How it works (5/11)

- The security server
  - Triggered if AVC does not have the required answer cached
  - Security server interprets the policy from userspace
  - Considers booleans
  - Considers constraints

# How it works (6/11)

- Booleans
  - Contained in the conditional access vector table
  - Allow runtime modifications to a security policy without having to load an new policy
  - Can be managed from userspace
  - `# getsebool`
    - http://manpg.es/getsebool
  - `# setsebool`
    - http://manpg.es/setsebool

# How it works (7/11)

- Constraints
  - http://danwalsh.livejournal.com/12333.html
  - Used to prevent people from writing bad policies
  - In case of MLS, to enforce rules governing information flow
  - `# neverallow`

# How it works (8/11)

- Users and roles
  - SELinux users are separate from {GNU/Linux, UNIX} users
  - Each user has a set of roles that he may operate under
  - User can switch to a different role if he has proper permissions to do so
    - ENTRYPOINT permission
  - `# newrole`
    - [http://manpg.es/newrole](http://manpg.es/newrole)

# How it works (9/11)

- The SELinux filesystem
    - The kernel communicates with userspace via filesystem
    - Mounted at /sys/fs/selinux or /selinux
    - `# cat /sys/fs/selinux/enforcing`
    - `# cat /sys/fs/selinux/disable`
    - `# cat /sys/fs/selinux/load`
    - Also some thingies in /proc
    - `$ cat /proc/<PID>/attr/current`

# How it works (10/11)

- User-space object managers
  - Related to objects that are managed outside the kernel
  - libselinux contains the required functions
    - Object labeling, global policy queries,...

# How it works (11/11)

- Policy files
  - http://danwalsh.livejournal.com/35127.html
  - Written in a text-based language
  - Compiled and converted to a binary blob that gets loaded into the kernel
  - libsepol implements the functions required for parsing these files

# What we learned so far or what I may have forgot to mention (1/4)

- SELinux "knows" if you are a user or an application
- Everybody gets their user, role, type (and MLS)
- We have RBAC, TE and MLS
- Userspace tools such as ps,ls,... have an additional -Z parameter that shows security contexts
- Error messages end up in /var/log/messages or /var/log/audit/audit.log

# What we learned so far or what I may have forgot to mention (2/4)

- Configuration files are stored in /etc/selinux/
- One can force the relabeling of the entire filesystem on next reboot
  - `# touch /.autorelabel`
- There is a number of predefined contexts that nobody uses :(
  - ~/.cert
  - ~/VirtualMachines

# What we learned so far or what I may have forgot to mention (3/4)

- There are some user-space tools that I feel I should mention
  - `# restorecon`
    - http://manpg.es/restorecon
  - `# secon`
    - http://manpg.es/secon
  - `# audit2allow`
    - http://manpg.es/audit2allow
  - `# setenforce`
    - http://manpg.es/setenforce

# What we learned so far or what I may have forgot to mention (4/4)

- We also have audit2why (good luck)
- There is a difference between what a user can do and what an application can do
  - Once upon a time there was a NULL pointer dereference…
  - http://eparis.livejournal.com/606.html
- Beware the m4 :)
- Running your system with SELinux disabled and then enabling it can be challenging

# Some usage examples (1/1)

- sVirt
  - Uses the MLS field for VM separation
  - Each VM gets a unique MCS label
  - http://danwalsh.livejournal.com/30565.html
- Harvard created a PaaS solution that relies on SELinux
  - http://opensource.com/education/12/8/harvard-goes-paas-selinux-sandbox
- SE Android
- SEPostgreSQL

# SE Android (1/2)

- http://selinuxproject.org/page/SEAndroid
- http://source.android.com/devices/tech/security/se-linux.html
- Since 4.3 in mainline AOSP, since 4.4 in Enforcing mode
- Middleware MAC concept
  - Install-time
  - EOPs
- Restrictions on a per-domain basis
  - root domain
  - application domain

# SE Android (2/2)

- A small number of confined daemons ATM
  - initd
  - installd
  - vold
- Reference policy in SE Android branch
- Vendors are expected to contribute
- Various benefits
  - Less need to check each application in the Google Play Store
  - Less malware
  - BYOD? (Samsung KNOX etc.)

# SEPostgreSQL (1/1)

- http://wiki.postgresql.org/wiki/SEPostgreSQL_Introduction
- SELinux for PostgreSQL DB
- Remember the part about user-space object managers?
  - This is an implementation of it
- Access can be configured on row/column level
- Each DB object gets a security context

# The end

- Thank you for listening :)
- Questions?