SELinux fun with MySQL and friends

Matthias Crauwels

Oct 21st 2020
Percona Live Online

Pythian
love your data®

Who am I?

Matthias Crauwels

- Living in Ghent, Belgium
- Bachelor Computer Science
- ~20 years Linux user / admin
- ~10 years PHP developer
- ~8 years MySQL DBA
- 3rd year at Pythian
- Currently Lead Database Consultant
- Father of Leander

Helping businesses use data to compete and win

Pythian

LOVE YOUR DATA

Pythian

Founded in 1997, Pythian is a global IT services company that helps organizations transform how they compete and win by helping them turn data into valuable insights, predictions and products. From cloud automation to machine learning, Pythian designs, implements and supports customized solutions to the toughest data challenges.

AGENDA

- What is SELinux
- MySQL and SELinux
- ProxySQL and SELinux
- Alternatives

What is SELinux

What is SELinux?

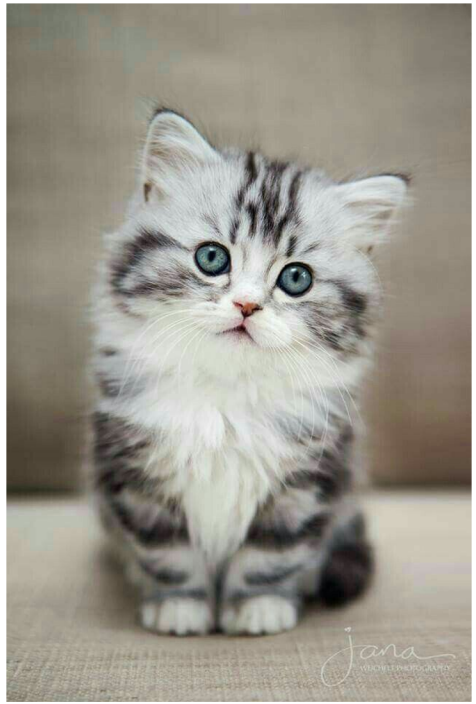
"Security-Enhanced Linux is a Linux kernel security module that provides a mechanism for supporting access control security policies, including mandatory access controls. SELinux is a set of kernel modifications and user-space tools that have been added to various Linux distributions. "
-- Wikipedia

- Originally developed by NSA and Red Hat
- Set of kernel modules to enhance security

What is SELinux?

- 3 modes
  - Enforcing
  - Permissive
  - Disabled
- Default installed on any Red Hat Enterprise Linux (RHEL) or CentOS distributions and set to "Enforcing" by default
- A wise man once said:

"Every time you disable SELinux a kitten dies!"



What is SELinux?

- Truth is - mostly the compliance / security teams will bite you if you disable SELinux
- Usually part of the security audit trails
- By default there is a **deny** policy
  - Anything you don't specifically allow will be denied.
- Useful tool to install: `policycoreutils-python`
  - This package provides a lot of tools to manage and define your SELinux policies.
- If you want to go more in depth the tool `policycoreutils-devel` might also be interesting

Check current SELinux status

- Use the tool `sestatus`

```
[root@localhost ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      31
```

- Quickly dynamically changing the status: `setenforce` / `getenforce`

```
[root@localhost ~]# getenforce
Enforcing
[root@localhost ~]# setenforce permissive
[root@localhost ~]# getenforce
Permissive
```

Remember the kittens!

SELinux: users, policies, contexts

- There is no 1:1 mapping between Linux system-users and SELinux users (however there can be).
- Use the `semanage` tool to manage the SELinux users and to map Linux system-users to SELinux users:

```
[root@localhost ~]# semanage user -l
                Labeling   MLS/       MLS/
SELinux User    Prefix     MCS Level  MCS Range                      SELinux Roles
guest_u         user       s0         s0                             guest_r
root            user       s0         s0-s0:c0.c1023                 staff_r sysadm_r system_r unconfined_r
staff_u         user       s0         s0-s0:c0.c1023                 staff_r sysadm_r system_r unconfined_r
user_u          user       s0         s0                             user_r
...
[root@localhost ~]# semanage login -a -s user_u john
[root@localhost ~]# semanage login -l
Login Name           SELinux User         MLS/MCS Range        Service
__default__          unconfined_u         s0-s0:c0.c1023       *
john                 user_u               s0                   *
...
```

SELinux: users, policies, contexts

- SELinux adds a `-Z` option to `ls` or `ps` to check the context of the process or the file.

```
[root@localhost ~]# ls -hlZ /var/lib/mysql/
...
-rw-rw----. mysql mysql system_u:object_r:mysqld_db_t:s0 ibdata1
-rw-rw----. mysql mysql system_u:object_r:mysqld_db_t:s0 ib_logfile0
-rw-rw----. mysql mysql system_u:object_r:mysqld_db_t:s0 ib_logfile1
drwx------. mysql mysql system_u:object_r:mysqld_db_t:s0 mysql
srwxrwxrwx. mysql mysql system_u:object_r:mysqld_var_run_t:s0 mysql.sock
drwx------. mysql mysql system_u:object_r:mysqld_db_t:s0 performance_schema
drwx------. mysql mysql system_u:object_r:mysqld_db_t:s0 test
...

[root@localhost ~]# ps -eZ | grep mysql
system_u:system_r:mysqld_safe_t:s0 4143 ?      00:00:00 mysqld_safe
system_u:system_r:mysqld_t:s0    4305 ?        00:00:00 mysqld
```

- Contexts are defined: `user:role:type:level`

MySQL and SELinux

MySQL and SELinux

- Out of the box experience is so that everything works!
- There is a predefined policy

```
[root@localhost ~]# semanage module -l | grep mysql
mysql                   100       pp
```

- MySQL contexts are predefined and pretty granular

```
[root@localhost ~]# semanage fcontext -l | grep mysql
/etc/mysql(/.*)?                                   all files          system_u:object_r:mysqld_etc_t:s0
/etc/my\.cnf\.d(/.*)?                              all files          system_u:object_r:mysqld_etc_t:s0
/var/log/mysql.*                                   regular file       system_u:object_r:mysqld_log_t:s0
/var/lib/mysql(-files|-keyring)?(/.*)?             all files          system_u:object_r:mysqld_db_t:s0
/var/run/mysqld(/.*)?                              all files          system_u:object_r:mysqld_var_run_t:s0
/usr/sbin/mysqld(-max)?                            regular file       system_u:object_r:mysqld_exec_t:s0
/usr/lib/systemd/system/mysqld.*                   regular file       system_u:object_r:mysqld_unit_file_t:s0
/etc/my\.cnf                                       regular file       system_u:object_r:mysqld_etc_t:s0
/root/\.my\.cnf                                    regular file       system_u:object_r:mysqld_home_t:s0
/usr/sbin/ndbd                                     regular file       system_u:object_r:mysqld_exec_t:s0
/usr/libexec/mysqld                                regular file       system_u:object_r:mysqld_exec_t:s0
/usr/bin/mysqld_safe                               regular file       system_u:object_r:mysqld_safe_exec_t:s0
/usr/bin/mysql_upgrade                             regular file       system_u:object_r:mysqld_exec_t:s0
/etc/rc\.d/init\.d/mysqld                          regular file       system_u:object_r:mysqld_initrc_exec_t:s0
/var/lib/mysql/mysql\.sock                         socket             system_u:object_r:mysqld_var_run_t:s0
/home/[^/]+/\.my\.cnf                              regular file       unconfined_u:object_r:mysqld_home_t:s0
```

Custom data directory

```
[root@localhost ~]# mkdir -p /data/mysql
[root@localhost ~]# chown mysql:mysql /data/mysql/
[root@localhost ~]# ls -hlZa /data/mysql/
drwxr-xr-x. mysql mysql unconfined_u:object_r:default_t:s0 .
drwxr-xr-x. root  root  unconfined_u:object_r:default_t:s0 ..

[root@localhost ~]# systemctl start mariadb.service
Job for mariadb.service failed because the control process exited with error code. See "systemctl status mariadb.service" and
"journalctl -xe" for details.

[root@localhost ~]# cat /var/log/audit/audit.log | audit2allow -w -a
type=AVC msg=audit(1578508437.315:664): avc:  denied  { write } for  pid=7047 comm="mysqld" name="mysql" dev="sda1" ino=1179650
scontext=system_u:system_r:mysqld_t:s0 tcontext=unconfined_u:object_r:default_t:s0 tclass=dir permissive=0
              Was caused by:
                            Missing type enforcement (TE) allow rule.

                            You can use audit2allow to generate a loadable module to allow this access.

[root@localhost ~]#
```

Custom data directory

- Finding the correct data directory context

```
[root@localhost ~]# semanage fcontext -l | grep /var/lib/mysql
/var/lib/mysql(-files|-keyring)?(/.*)?             all files          system_u:object_r:mysqld_db_t:s0
/var/lib/mysql/mysql\.sock                         socket             system_u:object_r:mysqld_var_run_t:s0
```

- Setting the context on the new directory

```
[root@localhost ~]# semanage fcontext -a -t mysqld_db_t '/data/mysql(/.*)?'
[root@localhost ~]# ls -hlZa /data/mysql
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 .
drwxr-xr-x. root root unconfined_u:object_r:default_t:s0 ..
[root@localhost ~]#
```

Wait? What!

Custom data directory

- Applying the context to the folder

```
[root@localhost ~]# restorecon -v /data/mysql/
restorecon reset /data/mysql context unconfined_u:object_r:default_t:s0->unconfined_u:object_r:mysqld_db_t:s0

[root@localhost ~]# ls -hlZa /data/mysql
drwxr-xr-x. mysql mysql unconfined_u:object_r:mysqld_db_t:s0 .
drwxr-xr-x. root  root  unconfined_u:object_r:default_t:s0 ..
```

- Starting the server

```
[root@localhost ~]# systemctl start mariadb.service
[root@localhost ~]# systemctl status mariadb.service
● mariadb.service - MariaDB database server
...
   Active: active (running) since Thu 2020-01-09 08:40:40 UTC; 30s ago
...
 Main PID: 27027 (mysqld_safe)
   CGroup: /system.slice/mariadb.service
           ├─27027 /bin/sh /usr/bin/mysqld_safe --basedir=/usr
           └─27201 /usr/libexec/mysqld --basedir=/usr --datadir=/data/mysql --plugin-
dir=/usr/lib64/mysql/plugin --log-error=/var/log/mariadb/mariadb.log --pid-file=/var/run/mariadb/mariadb.pid
--socket=/var/lib/mysql/mysql.sock
```

Custom port

- We want our MySQL instance to run on port 3307 so we add `port = 3307` to the `[mysqld]` section in `my.cnf`
- Restarting the service

```
[root@localhost ~]# systemctl start mariadb.service
Job for mariadb.service failed because the control process exited with error code. See "systemctl status
mariadb.service" and "journalctl -xe" for details.


[root@localhost ~]# systemctl status mariadb.service
● mariadb.service - MariaDB database server
...
Jan 09 08:44:52 localhost.localdomain systemd[1]: mariadb.service: control process exited, code=exited
status=1
Jan 09 08:44:52 localhost.localdomain systemd[1]: Failed to start MariaDB database server.
Jan 09 08:44:52 localhost.localdomain systemd[1]: Unit mariadb.service entered failed state.
Jan 09 08:44:52 localhost.localdomain systemd[1]: mariadb.service failed.
```

Custom port

- checking journalctl

```
-- Unit mariadb.service has begun starting up.
Jan 09 08:44:46 localhost.localdomain mariadb-prepare-db-dir[27302]: Database MariaDB is probably initialized
in /data/mysql already, nothing is done.
Jan 09 08:44:46 localhost.localdomain mariadb-prepare-db-dir[27302]: If this is not the case, make sure the
/data/mysql is empty before running mariadb-prepare-db-dir.
Jan 09 08:44:46 localhost.localdomain mysqld_safe[27336]: 200109 08:44:46 mysqld_safe Logging to
'/var/log/mariadb/mariadb.log'.
Jan 09 08:44:46 localhost.localdomain mysqld_safe[27336]: 200109 08:44:46 mysqld_safe Starting mysqld daemon
with databases from /data/mysql
Jan 09 08:44:52 localhost.localdomain systemd[1]: mariadb.service: control process exited, code=exited
status=1
Jan 09 08:44:52 localhost.localdomain systemd[1]: Failed to start MariaDB database server.-- Subject: Unit
mariadb.service has failed
```

- checking error log

```
200109  8:44:48 [Note] Server socket created on IP: '0.0.0.0'.
200109  8:44:48 [ERROR] Can't start server: Bind on TCP/IP port. Got error: 13: Permission denied
200109  8:44:48 [ERROR] Do you already have another mysqld server running on port: 3307 ?
200109  8:44:48 [ERROR] Aborting
```

Custom port

- Already in use?

```
[root@localhost ~]# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:111             0.0.0.0:*               LISTEN      1/systemd
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      2438/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      2674/master
tcp6       0      0 :::111                  :::*                    LISTEN      1/systemd
tcp6       0      0 :::22                   :::*                    LISTEN      2438/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      2674/master
[root@localhost ~]#
```

Custom port

- SELinux also defines the port(s) a service can use!

```
[root@localhost ~]# cat /var/log/audit/audit.log | audit2allow -w -a
type=AVC msg=audit(1578559488.003:936): avc:  denied  { name_bind } for  pid=27525 comm="mysqld" src=3307
scontext=system_u:system_r:mysqld_t:s0 tcontext=system_u:object_r:unreserved_port_t:s0 tclass=tcp_socket
permissive=0
               Was caused by:
               The boolean nis_enabled was set incorrectly.
               Description:
               Allow nis to enabled

               Allow access by executing:
               # setsebool -P nis_enabled 1
```

Custom port

- Setting the `nis_enabled` boolean works...

```
[root@localhost ~]# setsebool -P nis_enabled 1
[root@localhost ~]# systemctl start mariadb.service
[root@localhost ~]# netstat -lntp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:3307            0.0.0.0:*               LISTEN      27908/mysqld
...
[root@localhost ~]# systemctl stop mariadb.service
[root@localhost ~]# setsebool -P nis_enabled 0
[root@localhost ~]# systemctl start mariadb.service
Job for mariadb.service failed because the control process exited with error code. See "systemctl status
mariadb.service" and "journalctl -xe" for details.
```

- But what does `nis_enabled` do really?
  - Allows a process to bind to any port

Custom port

- ... the compliance team may or may not like that level of freedom (usually they don't)

```
[root@localhost ~]# semanage port -l | grep mysql
mysqld_port_t                  tcp      1186, 3306, 63132-63164
mysqlmanagerd_port_t           tcp      2273
[root@localhost ~]# semanage port -a -t mysqld_port_t -p tcp 3307
[root@localhost ~]# semanage port -l | grep mysql
mysqld_port_t                  tcp      3307, 1186, 3306, 63132-63164
mysqlmanagerd_port_t           tcp      2273
[root@localhost ~]# systemctl start mariadb.service
[root@localhost ~]# systemctl status mariadb.service
● mariadb.service - MariaDB database server
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; disabled; vendor preset: disabled)
   Active: active (running) since Thu 2020-01-09 09:07:34 UTC; 27s ago
  Process: 28313 ExecStartPost=/usr/libexec/mariadb-wait-ready $MAINPID (code=exited, status=0/SUCCESS)
  Process: 28278 ExecStartPre=/usr/libexec/mariadb-prepare-db-dir %n (code=exited, status=0/SUCCESS)
 Main PID: 28312 (mysqld_safe)
   CGroup: /system.slice/mariadb.service
           ├─28312 /bin/sh /usr/bin/mysqld_safe --basedir=/usr
           └─28501 /usr/libexec/mysqld --basedir=/usr --datadir=/data/mysql --plugin-
dir=/usr/lib64/mysql/plugin --log-error=/var/log/mariadb/mariadb.log --pid-file=/var/run/mariadb/mariadb.pid
--socket=/var/lib/mysql/mysql.sock --port=3307
```

ProxySQL and SELinux

ProxySQL and SELinux

- ProxySQL contexts are not defined

```
[root@localhost ~]# sestatus
SELinux status:                 enabled
...
Current mode:                   enforcing
...
[root@localhost ~]# systemctl status proxysql
● proxysql.service - High Performance Advanced Proxy for MySQL
...
   Active: active (running) since Thu 2020-01-09 09:18:12 UTC; 4s ago
...
[root@localhost ~]# ps -eZ | grep proxysql
system_u:system_r:unconfined_service_t:s0 4421 ? 00:00:00 proxysql
system_u:system_r:unconfined_service_t:s0 4422 ? 00:00:13 proxysql
[root@localhost ~]# ls -hlZa /var/lib/proxysql/
drwxr-xr-x. proxysql proxysql unconfined_u:object_r:var_lib_t:s0 .
drwxr-xr-x. root     root     system_u:object_r:var_lib_t:s0   ..
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   proxysql.db
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   proxysql.log
-rw-r--r--. proxysql proxysql system_u:object_r:var_lib_t:s0   proxysql.pid
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   proxysql_stats.db
[root@localhost ~]#
```

ProxySQL and SELinux

- It works out of the box! Why should we bother?
- Our log rotation fails!

```
[root@localhost ~]# tail -f /var/log/cron /var/log/messages
==> /var/log/messages <==
Jan  9 10:02:01 localhost systemd: Created slice User Slice of root.
Jan  9 10:02:01 localhost systemd: Started Session 12 of user root.

==> /var/log/cron <==
Jan  9 10:02:01 localhost CROND[4502]: (root) CMD (/etc/logrotate-proxysql)

==> /var/log/messages <==
Jan  9 10:02:01 localhost logrotate: ALERT exited abnormally with [1]
Jan  9 10:02:01 localhost systemd: Removed slice User Slice of root.

[root@localhost ~]# tail /var/spool/mail/root
From: "(Cron Daemon)" <root@localhost.localdomain>
To: root@localhost.localdomain
Subject: Cron <root@localhost> /etc/logrotate-proxysql
Date: Thu,  9 Jan 2020 10:02:02 +0000 (UTC)

error: error renaming /var/lib/proxysql/proxysql.log.2.gz to /var/lib/proxysql/proxysql.log.3.gz:
Permission denied

[root@localhost ~]#
```

ProxySQL log rotation

Seems to work?
... but not in crontab!

```
[root@localhost ~]# logrotate -fv /etc/logrotate.d/proxysql
reading config file /etc/logrotate.d/proxysql
Allocating hash table for state file, size 15360 B

Handling 1 logs

rotating pattern: /var/lib/proxysql/proxysql.log  forced from command line (5 rotations)
empty log files are rotated, old logs are removed
considering log /var/lib/proxysql/proxysql.log
  log needs rotating
rotating log /var/lib/proxysql/proxysql.log, log->rotateCount is 5
dateext suffix '-20200109'
glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
renaming /var/lib/proxysql/proxysql.log.5.gz to /var/lib/proxysql/proxysql.log.6.gz (rotatecount 5, logstart
1, i 5),
...
fscreate context set to system_u:object_r:var_lib_t:s0
renaming /var/lib/proxysql/proxysql.log to /var/lib/proxysql/proxysql.log.1
creating new /var/lib/proxysql/proxysql.log mode = 0600 uid = 997 gid = 993
running postrotate script
compressing log with: /bin/gzip
set default create context to system_u:object_r:var_lib_t:s0
set default create context
[root@localhost ~]#
```

To the audit log!

- Let's see what our audit log has to say about this

```
[root@localhost ~]# tail /var/log/audit/audit.log | audit2allow -w -a
type=AVC msg=audit(1578564661.266:480): avc:  denied  { rename } for  pid=4587
comm="logrotate" name="proxysql.log.3.gz" dev="sda1" ino=1835792
scontext=system_u:system_r:logrotate_t:s0-s0:c0.c1023
tcontext=system_u:object_r:var_lib_t:s0 tclass=file permissive=0
            Was caused by:
                        Missing type enforcement (TE) allow rule.

                        You can use audit2allow to generate a loadable module to allow
this access.
```

Let's create a policy for that!

```
[root@localhost ~]# tail /var/log/audit/audit.log | audit2allow -m proxysql > proxysql.te
[root@localhost ~]# cat proxysql.te

module proxysql 1.0;

require {
        type logrotate_t;
        type var_lib_t;
        class file rename;
}

#============= logrotate_t ==============
allow logrotate_t var_lib_t:file rename;

[root@localhost ~]# checkmodule -M -m -o proxysql.mod proxysql.te
checkmodule:  loading policy configuration from proxysql.te
checkmodule:  policy configuration loaded
checkmodule:  writing binary representation (version 19) to proxysql.mod
[root@localhost ~]# semodule_package -o proxysql.pp -m proxysql.mod
[root@localhost ~]# semodule -i proxysql.pp
[root@localhost ~]# semodule -l | grep proxysql
proxysql        1.0
[root@localhost ~]#
```

Still not working?

- This policy just allows for the rename

```
#============= logrotate_t ==============
allow logrotate_t var_lib_t:file rename;
```

- More operations are required to make this work.
- How to figure out all that is required for a process?
  1. Set SELinux to permissive
  2. Run the process
  3. Use audit2allow to generate policy file based on audit log
  4. Compile and load the policy
  5. Repeat 2-4 if still not working
  6. Set SELinux back to enforcing

Final policy

- After a few iterations this is the policy we ended up with:

```
module proxysql 1.0.0;

require {
        type var_lib_t;
        type logrotate_t;
        type unreserved_port_t;
        class tcp_socket name_connect;
        class file { create rename setattr unlink write };
}

#============= logrotate_t ==============
# Allow connecting to ProxySQL Admin port 6032
allow logrotate_t unreserved_port_t:tcp_socket name_connect;

# Allow modifying the log files in /var/lib/proxysql
allow logrotate_t var_lib_t:file { create rename setattr unlink write };
```

Great success!

- No more errors, and our logs are getting rotated

```
[root@localhost ~]# tail -f /var/log/cron /var/log/messages
==> /var/log/messages <==
Jan  9 10:25:01 localhost systemd: Created slice User Slice of root.
Jan  9 10:25:01 localhost systemd: Started Session 36 of user root.

==> /var/log/cron <==
Jan  9 10:25:01 localhost CROND[4808]: (root) CMD (/etc/logrotate-proxysql)

==> /var/log/messages <==
Jan  9 10:25:01 localhost systemd: Removed slice User Slice of root.

[root@localhost ~]# ls -hlZ /var/lib/proxysql/proxysql.log*
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   /var/lib/proxysql/proxysql.log
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   /var/lib/proxysql/proxysql.log.1.gz
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   /var/lib/proxysql/proxysql.log.2.gz
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   /var/lib/proxysql/proxysql.log.3.gz
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   /var/lib/proxysql/proxysql.log.4.gz
-rw-------. proxysql proxysql system_u:object_r:var_lib_t:s0   /var/lib/proxysql/proxysql.log.5.gz
```

Best solution?

- Is this the best solution? Probably not...
  - This allows logrotate in cron to delete, rename, .. any file (if it has the regular filesystem permissions to do so) in /var/lib
- What would be better?
  - We should define a ProxySQL SELinux policy
  - Define the granular contexts for all the different files the proxy creates (database files, log files, pid files, sockets, ...)
  - Ensure that the daemon runs as it's own SELinux user

Alternatives

Alternatives to SELinux

- Most well known alternative is "AppArmor"
- Default on SUSE Linux Enterprise Servers (SLES), openSUSE and Debian based platforms (including Ubuntu)
- Some key differences between SELinux and AppArmor
  - AppArmor is path based instead of inode, example creating a hardlink to a file/folder may change the accessibility of this object
  - Different methods of administration
  - SELinux supports a "remote policy server" for remote configuration of the policies
- Others are GrSecurity, RSBAC, ...