SELinux - Redhat / CentOS 7

Introduction

Security-Enhanced Linux (SELinux) ist ein obligatorischer Zugriffskontrollmechanismus - *mandatory access control* (MAC), der im Kernel implementiert ist. SELinux wurde zuerst in CentOS 4 / Red Hat 4 eingeführt und in späteren Versionen erheblich verbessert.

Some of the Problems

In order to better understand why SELinux is important and what it can do for you, it is easiest to look at some examples. Without SELinux enabled, only traditional discretionary access control (DAC) methods such as file permissions or access control lists (ACLs) are used to control the file access of users. Users and programs alike are allowed to grant insecure file permissions to others or, conversely, to gain access to parts of the system that should not otherwise be necessary for normal operation. For example:

- `Administrators have no way to control users: A user could set world readable permissions on sensitive files such as ssh keys and the directory containing such keys, customarily: ~/.ssh/`
- `Processes can change security properties: A user's mail files should be readable only by that user, but the mail client software has the ability to change them to be world readable`
- `Processes inherit user's rights: Firefox, if compromised by a` **`trojaned`** `version, could read a user's private ssh keys even though it has no reason to do so.`

Essentially under the traditional DAC model, there are two privilege levels, root and user, and no easy way to enforce a model of least-privilege. Many processes that are launched by root later drop their rights to run as a restricted user and some processes may be run in a chroot jail but all of these security methods are discretionary.

The Solution

SELinux follows the model of least-privilege more closely. By default under a strict enforcing setting, everything is denied and then a series of exceptions policies are written that give each element of the system (a service, program or user) only the access required to function. If a service, program or user subsequently tries to access or modify a file or resource not necessary for it to function, then access

is denied and the action is logged.

Because SELinux is implemented within the kernel, individual applications do not need to be especially written or modified to work under SELinux although, of course, if written to watch for the error codes which SELinux returns, vide infra, might work better afterwards. If SELinux blocks an action, this is reported to the underlying application as a normal (or, at least, conventional) “access denied” type error to the application. Many applications, however, do not test all return codes on system calls and may return no message explaining the issue or may return in a misleading fashion.

Please note, however, that the hypothetical examples posed to provide possible greater safety of e.g., constraining programs authorized to a limited set of programs permitted to read a user's ~/.ssh/ directory, preventing a Mail Delivery Agent from tampering with group owernship or setting on group or other file read permissions, or a web browser being constrained from reading the user's home directory have not been implemented in SELinux policies accompanying any version of CentOS up to version 6. CentOS 6 and 7 have limited support for confining user programs as described above, but doesn't have as much coverage over user programs as targeted system daemons. If an admin wishes to change from the default unconfined login configuration, they can see the section below on Role-Based Access Control.

SELinux Modes

SELinux has three basic modes of operation, of which **Enforcing** is set as the installation default mode. There is, however, an additional qualifier of **targeted** or mls which control how pervasive SELinux rules are applied, with targeted being the less stringent level.

- **Enforcing**: The default mode which will enable and enforce the SELinux security policy on the system, denying access and logging actions
- **Permissive**: In Permissive mode, SELinux is enabled but will not enforce the security policy, only warn and log actions. Permissive mode is useful for troubleshooting SELinux issues
- **Disabled**: SELinux is turned off

The SELinux mode can be viewed and changed by using the SELinux Management GUI tool available on the Administration menu or from the command line by running 'system-config-selinux' (the SELinux Management GUI tool is part of the policycoreutils-gui package and is not installed by default).

Users who prefer the command line may use the “sestatus” command to view the current SELinux status:

```
# sestatus
```

```
[root@vbulli ~]# sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
```

```
Current mode:                   permissive
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Max kernel policy version:      28
```

The **setenforce** command may be used to switch between **1** →Enforcing and **0** → Permissive modes on the fly but note that these changes do not persist through a system reboot.

To make changes persistent through a system reboot, edit the 'SELINUX=' line in `/etc/selinux/config` for either 'enforcing', 'permissive', *or* 'disabled'. For example: 'SELINUX=permissive'

Note: When switching from Disabled to either Permissive or Enforcing mode, it is highly recommended that the system be rebooted and the filesystem relabeled.

RPM dependencies to manage SELinux

Throughout this text, we already saw programs such as **semanage** from the rpm **policecoreutils-python** package **to manage our SELinux environment**. If you missed installing it, we will begin this recipe by doing so (skip step 1 if you have already done this before):

1. Log in as root and install the following basic toolkit to work with SELinux:

```
# yum install policycoreutils-python
```

2. Now, we need some additional tools that will also be needed later in the SELinux Debugging:

```
# yum install setools setools-console setroubleshoot*
```

3. Next, install and configure the SELinux manual pages as they are not available by default on CentOS 7, but are important for getting detailed information about specific policies, security contexts, and SELinux Booleans later. First, we need to install another package:

```
# yum install policycoreutils-devel
```

4. Afterwards, let's generate all the man pages for all SELinux security context policies currently available on the system, and then update the manual pages database afterwards:

```
# sepolicy manpage -a -p /usr/share/man/man8; mandb
```

SELinux Policy

As noted, SELinux follows the model of least-privilege; by default everything is denied and then a policy is written that gives each element of the system only the access required to function. This description best describes the **strict policy**. However, such a policy is difficult to write that would be suitable in the wide range of circumstances that a product such as Enterprise Linux is likely to be used. The end result is that SELinux is likely to cause problems for system administrators and end users and rather than resolve these issues, system administrators may just disable SELinux thereby defeating the built-in protections.

By design, SELinux allows different policies to be written that are interchangeable. The default policy in CentOS is the **targeted policy** which “targets” and confines selected system processes. In CentOS 4 only 15 defined targets existed (including httpd, named, dhcpd, mysqld). Later, in CentOS 5 this number had risen to over 200 targets.

All other system processes and all remaining userspace programs, as well as any in-house applications, that is everything else on the system, runs in an **unconfined** domain and is not covered by the SELinux protection model.

One goal might be for every process that is installed and, by default, running at boot should be run in a confined domain. The **targeted policy** is designed to protect as many key processes as possible without adversely affecting the end user experience and most users should be totally unaware that SELinux is even running.

SELinux Access Control

The **targeted** SELinux policy on Redhat/CentOS ships with **4 forms of access control**:

- **Type Enforcement (TE):** Type Enforcement is the primary mechanism of access control used in the targeted policy
- **Role-Based Access Control (RBAC):** Based around SELinux users (not necessarily the same as the Linux user), but not used in the default configuration of the targeted policy
- **Multi-Level Security (MLS):** Not commonly used and often hidden in the default targeted policy.
- **Multi-Category Security(MCS):** An extension of Multi-Level Security, used in the targeted policy to implement compartmentalization of virtual machines and containers through sVirt.

All processes and files have an SELinux security context! Let's see these in action by looking at the SELinux security context of the Apache homepage: '/var/www/html/index.html'

```
# ls -Z /var/www/html/index.html
```

```
-rw-r--r--  username username system_u:object_r:httpd_sys_content_t
```

```
/var/www/html/index.html
```

In addition to the standard file permissions and ownership, we can see the SELinux security context fields: `system_u:object_r:httpd_sys_content_t`.

This is based upon user:role:type:mls. In our example above, user:role:type fields are displayed and mls is hidden. Within the default targeted policy, type is the important field used to implement Type Enforcement, in this case `httpd_sys_content_t`.

Now consider the SELinux security context of the Apache web server process: 'httpd'

```
# ps axZ | grep httpd
```

```
system_u:system_r:httpd_t        3234 ?        Ss     0:00 /usr/sbin/httpd
```

Here we see the from the type field that Apache is running under the `httpd_t` type domain.

Finally, let's look at the SELinux security context of a file in our home directory:

```
# ls -Z /home/username/myfile.txt
```

```
-rw-r--r--  username username user_u:object_r:user_home_t
/home/username/myfile.txt
```

here we see the type is user_home_t, the default type for files in a user's home directory.

Access is only allowed between similar types, so Apache running as `httpd_t` can read /var/www/html/index.html of type `httpd_sys_content_t`. Because Apache runs in the httpd_t domain and does not have the userid:username, it can not access **/home/username/myfile.txt** even though this file is world readable because **/home/username/myfile.txt** SELinux security context is not of type httpd_t. If Apache were to be exploited, assuming for the sake of this example that the **root** account right needed to effect a SELinux re-labeling into another context were not obtained, it would not be able to start any process not in the httpd_t domain (which prevents escalation of privileges) or access any file not in an httpd_t related domain.

Role-Based Access Control (RBAC)

Although the default configuration of the targeted policy is to use unconfined logins, the administrator can quite easily switch to the **Role-Based Access Control** model. This model also switches to 'strict' mode for user domains, to allow targeting each program individually. To enable this, use **semanage-login** to add a login mapping for your user.

```
# semanage login -a -s "staff_u" -r "s0-s0:c0.c1023" <username>
```

The semanage-login command maps a Linux username to an SELinux user named “staff_u”, with an MLS/MCS range of “s0-s0:c0.c1023”. After this, logging in will result in **`id -Z`** returning `staff_u:staff_r:staff_t:s0-s0:c0.c1023` opposed to unconfined_u:unconfined_r:unconfined_t:s0. Though staff_r is not a role meant for administration, it is a role that allows the user to change to other roles. When an admin would like to do system administration tasks they should switch to the sysadm_r role using the -r flag in **sudo**,

```
# sudo -r sysadm_r -i
```

This can be automated by adding a configuration file under /etc/sudoers.d/, to map the user to a default admin role.

```
%wheel  ALL=(ALL)       TYPE=sysadm_t   ROLE=sysadm_r   ALL
```

It is still possible to login as an unconfined user or switch to the unconfined role via **newrole**, although the benefits of confined user domains are then lost. It is also possible to remove the ability to do this by creating a new SELinux user associated with only a select set of roles,

```
# semanage user -a -R "staff_r sysadm_r system_r -r "s0-s0:c0.c1023"
my_staff_u
```

Then substituting staff_u for my_staff_u in the semanage-login command. Now attempting to switch to the unconfined_r role will result in an **AVC** and **SELINUX_ERR** message. If the admin wishes to remove the ability to login as an unconfined user completely, they should remap the default login to a more suitable SELinux user, again using semanage-login.

```
# semanage login -m -s "user_u" -r "s0" __default__
```

If a user wishes to login as a role other than their default it is up to the login program to provide this functionality. SSH allows logging in with an alternative SELinux role by specifying it as part of the login identifier (e.g., as a staff user logging in as unconfined_r).

```
# ssh <username>/unconfined_r@hostname.net
```

The strict model that comes with Role-Based Access Control isn't perfect from a perspective of least privilege; running a quick search using policy analysis tools we can see that several confined programs can still read a users private SSH keys.

```
# sesearch -ACS -t ssh_home_t -c file -p read
```

```
Found 132 semantic av rules:
   allow snapperd_t file_type : file { ioctl read getattr lock open } ;
   allow oddjob_mkhomedir_t user_home_type : file { ioctl read write create
getattr setattr lock append unlink link rename open } ;
   allow mplayer_t non_security_file_type : file { ioctl read getattr lock
```

```
open } ;
   allow sendmail_t user_home_type : file { ioctl read getattr lock open } ;
   allow systemd_tmpfiles_t non_auth_file_type : file { ioctl read write
create getattr setattr lock relabelfrom relabelto append unlink link rename
open } ;
   allow login_pgm ssh_home_t : file { ioctl read getattr lock open } ;
   allow ssh_keygen_t ssh_home_t : file { ioctl read write create getattr
setattr lock append unlink link rename open } ;
   allow colord_t user_home_type : file { read getattr } ;
   ... snip ...
```

`mplayer_t` likely doesn't need to read SSH private keys, but it is granted access to that transiently by being allowed to read types of content associated with `non_security_file_type` that is allowing mplayer to read any content that isn't security related so the user can play multimedia from anywhere on the filesystem. This could be further constrained in the base SELinux policy, but as previously mentioned is not a major focus for the Upstream Vendor.

Beyond the strict model, Role-Based Access Control also provides a mechanism for limiting the scope of what a user can do when they use **sudo** to switch to root. It is often desirable to enforce least privilege on users with specific roles like DBAs or auditors and the targeted policy includes several user roles for purposes like those, with documentation in their respective manual pages as mentioned in Policy Documentation.

```
# seinfo -r
```

```
Roles: 14
   auditadm_r
   dbadm_r
   guest_r
   staff_r
   user_r
   logadm_r
   object_r
   secadm_r
   sysadm_r
   system_r
   webadm_r
   xguest_r
   nx_server_r
   unconfined_r
```

To map a user to one of these admin roles, the same semanage-user command is used as before to create a new SELinux user associated with the desired roles, and then semanage-login to associate the Linux login with the SELinux user. If the user should also be able to start system daemons they administrate from their user domain (i.e., to start mysql as dbadm_r for debugging from a shell) the system_r role should be included in their list of associated roles.

```
# semanage user -a -R "staff_r system_r auditadm_r" -r "s0-s0:c0.c1023"
auditor_u
# semanage login -a -s "auditor_u" -r "s0-s0:c0.c1023" <username>
```

Multi-Category Security (MCS)

Multi-Category Security provides a way to associate a set or range of **compartments** with SELinux contexts. The targeted policy model implements compartmentalization of types associated with mcs_constrained_type. To understand how this works, it's required to know how to inspect the MLS part of security contexts. This is the part after the user:role:type section, and includes a range, which indicates a low and high security level.

```
system_u:system_r:httpd_t:s0           -        s0:c0.c5
                          ▼                     ▼
                  Low security level,    High security level, also
                  associated with no     associated with compartments
                  compartments.          c0, c1, c2, c3, c4 and c5.
```

One thing that is noticeable above is the lack of compartments on the low security level, as well as both security levels being the same. The first point is an implementation detail of the MCS model in the targeted policy. When an access vector is computed for a process that is associated with mcs_constrained_type, only the MCS compartments of the high level are compared. The second point is due to the fact that MLS is not in use.

The compartment part of the above security context is a **category range**, but can also be a set of categories separated by commas. A range of categories results in the context being associated with an inclusive set of categories in that range. Understanding how access is computed for two processes with a set of categories requires looking at the **dominance** rules for SELinux security levels (access is only allowed if the source type's high security level **dominates** the target type's high security level). Those rules are as follows (only accounting for categories, and not MLS security levels)

- Source dominates the target if the categories in the source context are the same as or a superset of those in the target context.
- Source is dominated by the target if the categories in the source context are a subset of the categories of the target context.
- Source and target are equal and dominate each other if the set of categories are the same in each context.

With that in mind, we know that a context with a category set of c0.c5 will be granted access to a context with a category set of c0,c3, but not a category set of c0,c6, or c0.c1023. This rule is the reason that sVirt generates a random set of categories, so there will be no overlap where one virt domain will dominate another. The Android project also does the same thing, to put applications in isolated domains.

An example use of Multi-Category Security could be using NGINX with multiple vhosts that connect to backend servers that are also running as httpd domains (e.g., PHP-FPM). Normally these instances of the backend servers would be able to modify and manage each others domains simply due to type-enforcement rules. If they're associated with categories, they can only do so if one backend server

dominates the other. Since NGINX itself is a HTTPD domain, it should dominate all backend servers, so if we have categories c0 through c5 available for HTTPD domains we would want to run NGINX as `system_u:system_r:httpd_t:s0-s0:c0.c5`, so it could connect to the upstream servers. Each backend server would run with a single category within c0-c5, and a context such as `system_u:system_r:httpd_t:s0-s0:c1`.

There are a couple of presequities to achieving this. First, `httpd_t` must be associated with the `mcs_constrained_type` attribute that is currently only associated with the following types on CentOS 7:

```
# seinfo -xamcs_constrained_type
```

```
   mcs_constrained_type
      netlabel_peer_t
      openshift_t
      openshift_app_t
      sandbox_min_t
      sandbox_x_t
      sandbox_web_t
      sandbox_net_t
      svirt_t
      svirt_tcg_t
      svirt_lxc_net_t
      svirt_qemu_net_t
      svirt_kvm_net_t
```

To add to this list of types it is necessary to create a local policy module that associates the desired type with the attribute. This is done using the typeattribute statement, and can be done like so:

```
policy_module(httpd_mcs, 1.0)
gen_require(`
    type httpd_t;
    attribute mcs_constrained_type;
')

typeattribute httpd_t mcs_constrained_type;
```

See 

Customizing Local Policy for instructions on building policy modules.

Once the type is associated with mcs_constrained_type each backend server must have their content relabeled to include the respective categories in their file context specifications. This can be achieved by adding the file types to /etc/selinux/targeted/contexts/customizable_types, but this can potentially break on a policy upgrade. An alternative is to add new file contexts specifications that include categories using **semanage-fcontext**:

```
# semanage fcontext -a -t httpd_sys_content_t -r "s0-s0:c1"
"/srv/backend1(/.*)?"
# semanage fcontext -a -t httpd_sys_content_t -r "s0-s0:c2"
"/srv/backend2(/.*)?"
```

The next step is making sure the backend servers are started with the correct security context. On CentOS 7 with systemd this can be achieved with the **SELinuxContext=** directive in the unit file, and in previous versions can be achieved using the **runcon** command.

```
# runcon "system_u:system_r:httpd_t:s0-s0:c1" "/usr/local/bin/backend-
server"
```

Or in the systemd unit:

```
SELinuxContext=system_u:system_r:httpd_t:s0-s0:c1
```

Now each backend server should be isolated from the other, while allowing NGINX access to manage and send messages to both of them.

---

https://wiki.centos.org/HowTos/SELinux

SELinux Troubleshooting

Um SELinux denied Meldungen ausfindig zu machen, mach man am einfachsten ein “cat” auf das SELinux audit.log mit der Kombination von “grep”. Beispiel:

```
# cat /var/log/audit/audit.log | grep avc
```

```
type=AVC msg=audit(1505717571.241:949): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717571.241:950): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717571.250:951): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717571.250:952): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
```

```
type=AVC msg=audit(1505717573.862:953): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717573.862:954): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717573.863:955): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717573.863:956): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717592.480:962): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717592.480:963): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717592.481:964): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717592.481:965): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=AVC msg=audit(1505717672.128:996): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
type=USER_AVC msg=audit(1505717701.127:1000): pid=1 uid=0 auid=4294967295
ses=4294967295 subj=system_u:system_r:init_t:s0 msg='avc:  received
setenforce notice (enforcing=0)  exe="/usr/lib/systemd/systemd" sauid=0
hostname=? addr=? terminal=?'
type=AVC msg=audit(1505717714.165:1003): avc:  denied  { name_connect } for
pid=1217 comm="java" dest=3306 scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket
```

Wenn man nun jedoch vor komplizierteren SELinux Problemen steht, empfielt es sich mit den Setools für SELinux zu arbeiten.

Verwenden von Setools / Setroubleshoot

- Installation von den Troubleshoot Packages *(Falls nicht schon vorhanden)*:

  ```
  # yum install setroubleshoot setools -y
  ```

- Um nun die Fehler aus unserem Audit.log automatisiert auszuwerten folgenden Befehl ausführen:

  ```
  # sealert -a /var/log/audit/audit.log
  ```

  Jeder hier generierte Report, beschreibt zuerst den Fehler, und erklärt danach möglichst genau,

wie das Problem behoben werden kann. Ausgabe des Befehls:

```
[root@admin-server ~]# sealert -a /var/log/audit/audit.log
100% done
found 1 alerts in /var/log/audit/audit.log
--------------------------------------------------------------------------------

SELinux is preventing /usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.141-1.b16.el7_3.x86_64/jre/bin/java (deleted) from
name_connect access on the tcp_socket port 3306.

*****  Plugin catchall (100. confidence) suggests
**************************

If you believe that java (deleted) should be allowed name_connect
access on the port 3306 tcp_socket by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# ausearch -c 'java' --raw | audit2allow -M my-java
# semodule -i my-java.pp


Additional Information:
Source Context                system_u:system_r:tomcat_t:s0
Target Context                system_u:object_r:mysqld_port_t:s0
Target Objects                port 3306 [ tcp_socket ]
Source                        java
Source Path                   /usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.141-1.b16.el
                              7_3.x86_64/jre/bin/java (deleted)
Port                          3306
Host                          <Unknown>
Source RPM Packages           java-1.8.0-openjdk-
                              headless-1.8.0.144-0.b01.el7_4.x86_64
Target RPM Packages
Policy RPM                    selinux-policy-3.13.1-166.el7_4.4.noarch
Selinux Enabled               True
Policy Type                   targeted
Enforcing Mode                Permissive
Host Name                     admin-server.blacknet
Platform                      Linux admin-server.blacknet
                              3.10.0-693.2.2.el7.x86_64 #1 SMP Tue Sep
12
                              22:26:13 UTC 2017 x86_64 x86_64
Alert Count                   16
First Seen                    2017-09-17 13:33:21 CEST
```

```
Last Seen                       2017-09-18 08:55:14 CEST
Local ID                        42523e63-a9ef-438e-8a07-7e8d128d669b

Raw Audit Messages
type=AVC msg=audit(1505717714.165:1003): avc:  denied  { name_connect }
for  pid=1217 comm="java" dest=3306
scontext=system_u:system_r:tomcat_t:s0
tcontext=system_u:object_r:mysqld_port_t:s0 tclass=tcp_socket


type=SYSCALL msg=audit(1505717714.165:1003): arch=x86_64
syscall=connect success=yes exit=0 a0=77 a1=7f4cb79f6380 a2=1c a3=504
items=0 ppid=1 pid=1217 auid=4294967295 uid=91 gid=91 euid=91 suid=91
fsuid=91 egid=91 sgid=91 fsgid=91 tty=(none) ses=4294967295 comm=java
exe=/usr/lib/jvm/java-1.8.0-
openjdk-1.8.0.144-0.b01.el7_4.x86_64/jre/bin/java
subj=system_u:system_r:tomcat_t:s0 key=(null)

Hash: java,tomcat_t,mysqld_port_t,tcp_socket,name_connect
```

- Wie oben ersichtlich, wird zu unserem Problem eine **Lösung durch Eingabe von folgenden zwei Befehlen** empfohlen:

```
# ausearch -c 'java' --raw | audit2allow -M guacamole-java
# semodule -i guacamole-java.pp
```

  - **Der erste Befehl, erstellt im aktuellen Verzeichnis eine neue SELinux Regel** im ***.te Format (Text)** und **kompiliert** sie anschliessend in ein ***.pp Format.** *(Der in der Ausgabe verwendete Namen "my-java" kann beliebig festgelet werden! z.B. wie bei mir: guacamole-java)*
  - **Mit dem zweiten Befehl, wird die Regel dann permanent aktiviert!**

- http://www.serverlab.ca/tutorials/linux/administration-linux/troubleshooting-selinux-centos-red-hat/
- https://serverfault.com/questions/321301/how-do-i-view-the-contents-of-a-selinux-policy-package
- https://www.centos.org/docs/5/html/Deployment_Guide-en-US/sec-sel-building-policy-module.html
- https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security-Enhanced_Linux/sect-Security-Enhanced_Linux-Troubleshooting-Top_Three_Causes_of_Problems.html

---

Redhat Dokumentation zum Thema

red_hat_enterprise_linux-7-selinux_users_and_administrators_guide-en-us.pdf

Last update: **2018/05/08 11:22**