Hands-On Exercises for

Android Security Internals

v. 2025.06

WARNING:
The order of the exercises does not always follow the same order of the explanations in the slides. When carrying out the exercises, carefully follow the exercise requirements. Do **NOT** blindly type the commands or code as found in the slides. Read every exercise **in its entirety** before carrying out the instructions.

These exercises created by: Karim Yaghmour

Originals at: www.opersys.com/training/android-security-internals/

Delivered and/or customized by:

Class Preparation Reference / Reminder

This section contains material that should have been used prior to class to prepare ahead of time. It's provided here as a reference/reminder.

Class Preparation for Cuttlefish

WARNING: The list of prerequisites for Android development changes over time

1. Install Android Studio:
https://developer.android.com/studio

2. Install the required packages to build Android:
See the "Installing required packages" instructions for the relevant Ubuntu version here: http://source.android.com/source/initializing.html

3. Make sure you have the "repo" tool available:
See the "Installing Repo" here: https://source.android.com/setup/develop
You may need to log out and back in if the "~/bin" directory wasn't already in your path.

4. Fetch the AOSP:
We'll be using 15.0.0_r5 Note that the new lines that start with ">" are printed by the shell. **Adding a ">" to the input you type will break the following commands.** Assuming you have a directory called "android" in your home directory:

```
$ mkdir -p ~/android/android-15.0.0_r5
$ cd ~/android/android-15.0.0_r5
$ repo init -u https://android.googlesource.com/platform/manifest \
> -b android-15.0.0_r5
$ repo sync
```

Note:

- Make sure the path depth is short. For reasons we do not control, the Cuttlefish emulator will refuse to start if the path length (in terms of characters) is too long. It's our understanding that Google is working on fixing this. In the mean time, make sure you keep the path relatively short in length. Generally our experience has been that it should be less than around 100 characters.
- Fetching the sources is a fairly lengthy process.

5. Check that you got the right AOSP version:

```
$ cat .repo/manifests/default.xml | grep revision
  <default revision="refs/tags/android-15.0.0_r5"
```

Make sure it says “android-15.0.0_r5” in this output.

6. Download the Android GKI kernel:

Note that we’re just downloading the kernel for now, we aren’t building it.

a. Create directory for working on kernel – assuming this is outside your the top-level of the AOSP:

```
$ cd ~/android/
$ mkdir android15-6.6-kernel
$ cd android15-6.6-kernel
```

b. Get the kernel itself and its tools:

```
$ repo init -u https://android.googlesource.com/kernel/manifest \
> -b common-android15-6.6
$ repo sync
```

7. Download and setup the Cuttlefish tools (Original instruction at https://android.googlesource.com/device/google/cuttlefish/):

Notes:

- Do NOT copy “>” symbols at the beginning of a line, they are added by the shell, you don’t type them in.
- The last instruction will reboot your computer (sudo reboot). Make sure that you can safely do that without loosing any information.

```
$ cd ~/android/
$ sudo apt install -y \
> git devscripts config-package-dev debhelper-compat golang curl
$ git clone https://github.com/google/android-cuttlefish
$ cd android-cuttlefish
$ git checkout 17ccfffd06139cd69d3b4cd4f0179b43b20aa573
$ for dir in base frontend; do
  cd $dir
  debuild -i -us -uc -b -d
  cd ..
done
$ sudo dpkg -i ./cuttlefish-base_*_*64.deb || sudo apt-get install -f
$ sudo dpkg -i ./cuttlefish-user_*_*64.deb || sudo apt-get install -f
$ sudo usermod -aG kvm,cvdnetwork,render $USER
$ sudo reboot
```

8. Build the AOSP:

Return to the AOSP directory and build it. We don’t specify any “-j” flag since on a modern AOSP it’ll be automatically figured out.

```
$ cd ~/android/android-15.0.0_r5
```

```
$ . build/envsetup.sh
$ lunch aosp_cf_x86_64_phone-trunk_staging-eng
$ make
```

You **MUST** check that the build completed successfully. Generally, you'll see this at the end of a successful build in the output (typically in green):

```
#### build completed successfully (...) ####
```

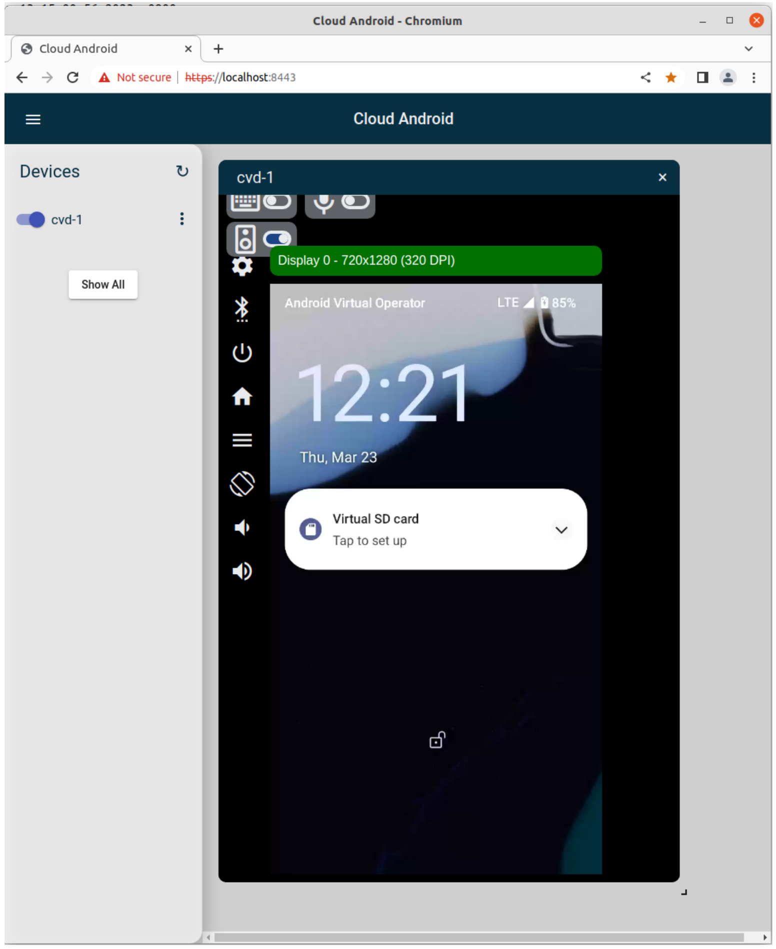
9. Give the AOSP a spin:

From the same directory where you build the AOSP, start the Cuttlefish target:

```
$ HOME=${PWD}/out launch_cvd
```

At this point, the Cuttlefish target will start with your custom-built Android running inside and print messages to the command line[1]. However Cuttlefish itself won't be visible. You'll need to use a Chrome-based browser (Firefox will NOT work) and direct it to https://localhost:8443 – do not omit the "https" or it won't work. You will likely need to accept some security exceptions to access the web page on which you'll see a device listed since it doesn't have a proper certificate. Once you click on the device you'll then see the Cuttlefish device interface.

1 If you are seeing errors regarding "kvm" then see the next section.



You can stop Cuttlefish on the command line as well:

```
$ HOME=${PWD}/out stop_cvd
```

Extended Stack Addition to AOSP

For the purpose of the exercises we will do in class, you will need to add several components to create a new system service, a new HAL and a corresponding HAL module and driver. The details of these additions are covered in Opersys' Embedded Android class material (https://www.opersys.com/training/embedded-android-training/). The archive we use for the present class is tailored for the present exercise set. Hence, while you may want to refer to the Embedded Android class material for background, we suggest you continue referring to the present exercises for the hands-on portions of the present class.

1) Backup your AOSP's existing kernel (watch out for the differences between "x86_64" and "x86-64") and remove the original kernel build artifacts from the output:

```
$ cd ~/android/android-15.0.0_r5
$ cd kernel/prebuilts/6.6/
$ mv x86_64 x86_64-orig
$ mkdir x86_64
$ git init x86_64/
$ cd ../common-modules/virtual-device/6.1
$ mv x86-64 x86-64-orig
$ mkdir x86-64
$ git init x86-64/
$ cd ~/android/android-15.0.0_r5/out/target/product/vsoc_x86_64/
$ rm kernel
```

2) Download and apply the Opersys patch for adding a system service, a HAL, corresponding HAL module and driver, and replacement kernel for Cuttlefish. Download the file here to your "~/android" directory:
https://drive.google.com/file/d/1X7HxZsJnEH9RKqellvEirhkbaQB0gkaQ/view

```
$ cd ~/android/
$ tar xvjf halext-opersys-cuttlefish-15.0.0_r5-no-se-250616.tar.bz2
$ find halext-opersys-cuttlefish-15.0.0_r5-no-se-250616 -exec touch {} \;
$ cp -r halext-opersys-cuttlefish-15.0.0_r5-no-se-250616/* android-15.0.0_r5
```

3) Rebuild your AOSP:

```
$ cd ~/android/android-15.0.0_r5
$ . build/envsetup.sh
$ lunch aosp_cf_x86_64_phone-trunk_staging-eng
$ make -j32
```

4) Restart Cuttlefish to take into account the new additions (don't forget the "--noresume" parameter):

```
$ HOME=${PWD}/out launch_cvd --noresume
```

Cuttlefish should restart with no noticeable difference in the immediate.

SEAndroid / SELinux Basics

In this section you will learn how to:

- Use the SELinux/SEAndroid tools
- Look for SELinux/SEAndroid information around the filesystem
- Check security errors in the logs

1. Familiarize yourself with the following tools and commands if you've never used the before:
    - Check the status of SELinux enforcement:
    ```
    vsoc_x86_64:/ # getenforce
    Enforcing
    ```
    - Disable SELinux enforcement – note that the value you set is not persisted to storage and will be reset to the default value upon reboot:
    ```
    vsoc_x86_64:/ # setenforce 0
    vsoc_x86_64:/ # getenforce
    Permissive
    ```
    - Enable SELinux enforcement:
    ```
    vsoc_x86_64:/ # setenforce 1
    vsoc_x86_64:/ # getenforce
    Enforcing
    ```
    - Check kernel logs for SELinux denials:
    ```
    vsoc_x86_64:/ # dmesg | grep avc
    [    9.290319] type=1400 audit(1603625582.200:4): avc: denied { execute_no_trans } for comm="init"
    path="/vendor/bin/toybox_vendor" dev="dm-1" ino=228 scontext=u:r:init:s0 tcontext=u:object_r:vendor_toolbox_exec:s0 tclass=file
    permissive=0 b/132695863
    [    9.304479] type=1400 audit(1603625582.216:5): avc: denied { execute_no_trans } for comm="init"
    path="/vendor/bin/toybox_vendor" dev="dm-1" ino=228 scontext=u:r:init:s0 tcontext=u:object_r:vendor_toolbox_exec:s0 tclass=file
    permissive=0 b/132695863
    [   22.821163] type=1107 audit(1603625595.732:6): uid=0 auid=4294967295 ses=4294967295 subj=u:r:init:s0 msg='avc: denied { set }
    for property=vendor.wlan.firmware.version pid=305 uid=1010 gid=1010 scontext=u:r:hal_wifi_default:s0
    tcontext=u:object_r:vendor_default_prop:s0 tclass=property_service permissive=0' b/131598173
    [   22.824328] type=1107 audit(1603625595.736:7): uid=0 auid=4294967295 ses=4294967295 subj=u:r:init:s0 msg='avc: denied { set }
    for property=vendor.wlan.driver.version pid=305 uid=1010 gid=1010 scontext=u:r:hal_wifi_default:s0
    tcontext=u:object_r:vendor_default_prop:s0 tclass=property_service permissive=0' b/131598173
    ...
    ```
    - Check Android logs for SELinux denials:
    ```
    vsoc_x86_64:/ # logcat | grep avc
    10-25 07:33:02.200   188   188 W init    : type=1400 audit(0.0:4): avc: denied { execute_no_trans } for
    path="/vendor/bin/toybox_vendor" dev="dm-1" ino=228 scontext=u:r:init:s0 tcontext=u:object_r:vendor_toolbox_exec:s0 tclass=file
    permissive=0 b/132695863
    10-25 07:33:02.216   190   190 W init    : type=1400 audit(0.0:5): avc: denied { execute_no_trans } for
    path="/vendor/bin/toybox_vendor" dev="dm-1" ino=228 scontext=u:r:init:s0 tcontext=u:object_r:vendor_toolbox_exec:s0 tclass=file
    permissive=0 b/132695863
    10-25 07:33:15.501   148   148 E SELinux : avc:  denied  { find } for interface=android.hardware.opersys::IOpersys
    sid=u:r:system_server:s0 pid=483 scontext=u:r:system_server:s0 tcontext=u:object_r:default_android_hwservice:s0
    tclass=hwservice_manager permissive=0
    10-25 07:33:15.501   147   147 E SELinux : avc:  denied  { add } for pid=483 uid=1000 name=opersys scontext=u:r:system_server:s0
    tcontext=u:object_r:default_android_service:s0 tclass=service_manager permissive=0
    10-25 07:33:15.732     1     1 W /system/bin/init: type=1107 audit(0.0:6): uid=0 auid=4294967295 ses=4294967295 subj=u:r:init:s0
    msg='avc: denied { set } for property=vendor.wlan.firmware.version pid=305 uid=1010 gid=1010 scontext=u:r:hal_wifi_default:s0
    tcontext=u:object_r:vendor_default_prop:s0 tclass=property_service permissive=0' b/131598173
    10-25 07:33:15.736     1     1 W /system/bin/init: type=1107 audit(0.0:7): uid=0 auid=4294967295 ses=4294967295 subj=u:r:init:s0
    msg='avc: denied { set } for property=vendor.wlan.driver.version pid=305 uid=1010 gid=1010 scontext=u:r:hal_wifi_default:s0
    tcontext=u:object_r:vendor_default_prop:s0 tclass=property_service permissive=0' b/131598173
    10-25 07:34:03.185     0     0 I selinux : avc:  received setenforce notice (enforcing=0)
    10-25 07:34:03.725   148   148 I SELinux : avc:  received setenforce notice (enforcing=0)
    10-25 07:34:05.456   147   147 I SELinux : avc:  received setenforce notice (enforcing=0)
    ...
    ```
    - Compare output from “ps” with and without security context information:
    ```
    vsoc_x86_64:/ # ps -A
    USER            PID   PPID     VSZ    RSS WCHAN            ADDR S NAME
    root              1      0 10782796  7236 do_epoll_+          0 S init
    ...
    system         4100   4039 13830920 274624 do_epoll_+         0 S system_server
    ```

```
bluetooth      4225   4039 12964824 126764 do_epoll_+         0 S com.android.bluetooth
u0_a104        4243   4039 13043784 234340 do_epoll_+         0 S com.android.systemui
webview_zyg+   4322   4040 1768136  69960 do_sys_po+          0 S webview_zygote
...
vsoc_x86_64:/ # ps -A -Z
LABEL                          USER           PID   PPID     VSZ    RSS WCHAN            ADDR S NAME
u:r:init:s0                    root             1      0 10782796  7236 do_epoll_+         0 S init
...
u:r:system_server:s0           system        4100   4039 13830968 275908 do_epoll_+         0 S system_server
u:r:bluetooth:s0               bluetooth     4225   4039 12964824 126764 do_epoll_+         0 S com.android.bluetooth
u:r:platform_app:s0:c512,c768  u0_a104       4243   4039 13043824 234604 do_epoll_+         0 S com.android.systemui
u:r:webview_zygote:s0          webview_zyg+  4322   4040 1768136  69960 do_sys_po+          0 S webview_zygote
...
```

The -Z flag enables you to see the security context associated with each process running in the system.

- Compare the output from “ls” with and without security context information:

```
vsoc_x86_64:/ # ls -al
dr-xr-xr-x  64 root   root          0 2020-10-25 07:32 acct
drwxr-xr-x  48 root   root        960 2020-10-25 07:33 apex
lrw-r--r--   1 root   root         11 2020-10-25 07:27 bin -> /system/bin
lrw-r--r--   1 root   root         50 2020-10-25 07:27 bugreports -> ... bugreports
drwxrwx---   2 system cache      4096 2020-10-25 06:59 cache
drwxr-xr-x   3 root   root          0 2020-10-25 07:32 config
lrw-r--r--   1 root   root         17 2020-10-25 07:27 d -> /sys/kernel/debug
drwxrwx--x  47 system system     4096 2020-10-25 07:35 data
drwx------   5 root   system      100 2020-10-25 07:33 data_mirror
...
vsoc_x86_64:/ # ls -al -Z
dr-xr-xr-x  64 root   root    u:object_r:cgroup:s0                    0 2020-10-25 07:32 acct
drwxr-xr-x  48 root   root    u:object_r:apex_mnt_dir:s0            960 2020-10-25 07:33 apex
lrw-r--r--   1 root   root    u:object_r:rootfs:s0                   11 2020-10-25 07:27 bin -> /system/bin
lrw-r--r--   1 root   root    u:object_r:rootfs:s0                   50 2020-10-25 07:27 bugreports -> ...
drwxrwx---   2 system cache   u:object_r:cache_file:s0             4096 2020-10-25 06:59 cache
drwxr-xr-x   3 root   root    u:object_r:configfs:s0                  0 2020-10-25 07:32 config
lrw-r--r--   1 root   root    u:object_r:rootfs:s0                   17 2020-10-25 07:27 d -> ...
drwxrwx--x  47 system system  u:object_r:system_data_root_file:s0  4096 2020-10-25 07:35 data
drwx------   5 root   system  u:object_r:mirror_data_file:s0        100 2020-10-25 07:33 data_mirror
...
```

The -Z flag enables you to see the security context associated with each file and directory in the filesystem.

- Compare the output of “id” with and without security context information:

```
vsoc_x86_64:/ # id
uid=0(root) gid=0(root)
groups=0(root),1004(input),1007(log),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin)
,3002(net_bt),3003(inet),3006(net_bw_stats),3009(readproc),3011(uhid) context=u:r:su:s0
vsoc_x86_64:/ # id -Z
u:r:su:s0
```

2. The filesystem also contains various entries that are related to SELinux and provide information about some key parts. Here are some examples for you to try:

- You can see the SELinux subsystem’s entries in sysfs:

```
vsoc_x86_64:/ # cd /sys/fs/selinux
vsoc_x86_64:/ # ls -al
total 0
-rw-rw-rw-   1 root root      0 2020-10-25 11:52 access
dr-xr-xr-x   2 root root      0 2020-10-25 11:52 avc
dr-xr-xr-x   2 root root      0 2020-10-25 11:52 booleans
-rw-r--r--   1 root root      0 2020-10-25 11:52 checkreqprot
dr-xr-xr-x 101 root root      0 2020-10-25 11:52 class
--w-------   1 root root      0 2020-10-25 11:52 commit_pending_bools
-rw-rw-rw-   1 root root      0 2020-10-25 11:52 context
```

```
-rw-rw-rw-   1 root root      0 2020-10-25 11:52 create
-r--r--r--   1 root root      0 2020-10-25 11:52 deny_unknown
--w-------   1 root root      0 2020-10-25 11:52 disable
-rw-r--r--   1 root root      0 2020-10-25 11:52 enforce
...
```

- You can see the classes of objects that are managed by SELinux in that directory along with the permissions for each class:

```
vsoc_x86_64:/ # cd class
vsoc_x86_64:/ # ls -al
...
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 alg_socket
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 appletalk_socket
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 association
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 atmpvc_socket
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 atmsvc_socket
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 ax25_socket
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 binder
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 blk_file
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 bluetooth_socket
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 bpf
dr-xr-xr-x   3 root root 0 2020-10-25 11:52 caif_socket
...
vsoc_x86_64:/sys/fs/selinux/class # ls -al binder/perms/
...
-r--r--r-- 1 root root 0 2020-10-25 11:52 call
-r--r--r-- 1 root root 0 2020-10-25 11:52 impersonate
-r--r--r-- 1 root root 0 2020-10-25 11:52 set_context_mgr
-r--r--r-- 1 root root 0 2020-10-25 11:52 transfer
vsoc_x86_64:/sys/fs/selinux/class # ls -al socket/perms
...
-r--r--r-- 1 root root 0 2020-10-25 11:52 accept
-r--r--r-- 1 root root 0 2020-10-25 11:52 append
-r--r--r-- 1 root root 0 2020-10-25 11:52 bind
-r--r--r-- 1 root root 0 2020-10-25 11:52 connect
-r--r--r-- 1 root root 0 2020-10-25 11:52 create
-r--r--r-- 1 root root 0 2020-10-25 11:52 getattr
-r--r--r-- 1 root root 0 2020-10-25 11:52 getopt
-r--r--r-- 1 root root 0 2020-10-25 11:52 ioctl
...
```

  By exploring this directory, you can see all the types of classes managed by SELinux/SEAndroid in Android and what types of permissions are associated with each.

- You can also see the initial set of security contexts which are statically defined as part of the kernel sources and which are used to kickstart the system:

```
vsoc_x86_64:/ # cd ../initial_contexts
vsoc_x86_64:/ # ls -al
...
-r--r--r-- 1 root root 0 2020-10-25 11:52 any_socket
-r--r--r-- 1 root root 0 2020-10-25 11:52 devnull
-r--r--r-- 1 root root 0 2020-10-25 11:52 file
-r--r--r-- 1 root root 0 2020-10-25 11:52 kernel
-r--r--r-- 1 root root 0 2020-10-25 11:52 netif
```

```
...
```

- You can also see process-specific SELinux information under /proc:

```
vsoc_x86_64:/ # ps -A | grep system_server
system          3279   3219 13834392 275112 do_epoll_wait      0 S system_server
vsoc_x86_64:/ # ls /proc/3279/attr/ -al
...
-rw-rw-rw- 1 system system 0 2020-10-25 15:11 current
-rw-rw-rw- 1 system system 0 2020-10-25 15:11 exec
-rw-rw-rw- 1 system system 0 2020-10-25 15:11 fscreate
-rw-rw-rw- 1 system system 0 2020-10-25 15:11 keycreate
-r--r--r-- 1 system system 0 2020-10-25 15:11 prev
-rw-rw-rw- 1 system system 0 2020-10-25 15:11 sockcreate
...
```

3. If you look into the logcat logs, you'll see that there are several security denial errors related to the Opersys components that were added by the patches earlier:

```
vsoc_x86_64:/ # logcat | grep -i opersys
10-25 19:14:27.285     0     0 I init    : starting service 'vendor.opersys-cuttlefish'...
10-25 19:14:27.310     0     0 I init    : ... started service 'vendor.opersys-cuttlefish' has pid 866
10-25 19:14:27.291   866   866 I OPERSYS : Starting opersys HAL.
10-25 19:14:27.296   866   866 W android.hardwar: type=1300 audit(0.0:134): arch=c000003e syscall=257 success=no exit=-13 a0=ffffff9c a1=77c14e1e56f7 a2=2 a3=0 items=0 ppid=1 auid=4294967295 uid=1000 gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=(none) ses=4294967295 exe="/apex/com.google.cf.opersys/bin/hw/android.hardware.opersys-service.cuttlefish" subj=u:r:hal_light_cuttlefish:s0 key=(null)
10-25 19:14:27.296   211   211 W auditd  : type=1327 audit(0.0:134): proctitle="/apex/com.google.cf.opersys/bin/hw/android.hardware.opersys-service.cuttlefish"
10-25 19:14:27.299   866   866 D opersyshw_hal_module: FAILED to open /dev/circchar
10-25 19:14:27.299   866   866 D opersyshw_hal_module: OPERSYS HW with name " opersyshw " has been initialized
10-25 19:14:27.299   866   866 I OPERSYS : Registering opersys HAL AIDL to ServiceManager.
10-25 19:14:27.335     0     0 E SELinux : avc:  denied  { add } for pid=866 uid=1000 name=android.hardware.opersys.IOpersys/default scontext=u:r:hal_light_cuttlefish:s0 tcontext=u:object_r:default_android_service:s0 tclass=service_manager permissive=0
...
```

Also, if you check to see if the system service is present you'll see that it isn't:

```
vsoc_x86_64:/ # service list | grep -i opersys
1|vsoc_x86_64:/ #
```

The SELinux rules won't permit this system service to operate. There's therefore no way for it to start.

4. Compare the output of "dmesg | grep -i opersys" to the output of the "logcat | grep -i opersys" above. You'll notice that the errors reported aren't the same. This is consistent with the fact that the SEAndroid functionality is deeply integrated into the Android user-space and isn't limited to just kernel checks.

Security Context Rule Basics

In this section you will learn how to:
- Investigate and follow SELinux/SEAndroid security contexts
- Understand how security contexts and rules are written
- Create and work with SELinux/SEAndroid rules in a variety of scenarios

1. In the previous exercises we saw that the Opersys system service and relevant components were'nt getting started due to SELinux issues. Let's try getting them working by "cheating". Let's disable the SELinux policy checking and restart the framework:

```
vsoc_x86_64:/ # stop
vsoc_x86_64:/ # setenforce 0
vsoc_x86_64:/ # start
```

Now let's check what the logs say and check if the system service was sarted:

```
vsoc_x86_64:/ # logcat | grep -i opersys
...
10-25 15:18:19.468  2995  2995 I OPERSYS : Starting opersys HAL.
10-25 15:18:19.470  2995  2995 I OPERSYS : Registering opersys HAL AIDL to ServiceManager.
10-25 15:18:19.468 2995 2995 W android.hardwar: type=1300 audit(0.0:306): arch=c000003e syscall=257 success=yes exit=5 a0=ffffff9c a1=7e15626246f7 a2=2 a3=0 items=0 ppid=1 auid=4294967295 uid=1000 gid=1000 euid=1000 suid=1000 fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=(none) ses=4294967295 exe="/apex/com.google.cf.opersys/bin/hw/android.hardware.opersys-service.cuttlefish" subj=u:r:hal_light_cuttlefish:s0 key=(null)
10-25 15:18:19.468 213 213 W auditd : type=1327 audit(0.0:306): proctitle="/apex/com.google.cf.opersys/bin/hw/android.hardware.opersys-service.cuttlefish"
10-25 15:18:19.508 0 0 E SELinux : avc: denied { add } for pid=2995 uid=1000 name=android.hardware.opersys.IOpersys/default scontext=u:r:hal_light_cuttlefish:s0 tcontext=u:object_r:default_android_service:s0 tclass=service_manager permissive=1
10-25 15:18:19.509 0 0 I servicemanager: Caller(pid=2995,uid=1000,sid=u:r:hal_light_cuttlefish:s0) Found android.hardware.opersys.IOpersys/default in device VINTF manifest.
10-25 15:18:19.476 2995 2995 I OPERSYS : Success registering opersys HAL AIDL to ServiceManager.
...
6-16 15:18:51.876  3168  3168 I SystemServer: Opersys Service
10-25 15:18:51.876  3168  3168 I OpersysService: About to get AIDL HAL
10-25 15:18:51.878  3168  3168 I OpersysService: Opersys AIDL HAL sucessfully retrieved
10-25 15:18:51.878  3168  3168 I OpersysService: System service initialized
10-25 15:18:51.878 3168 3168 I HidlServiceManagement: Trying to get transport of android.hardware.opersys@2.0::IOpersys/default without hwservicemanager
10-25 15:18:51.878  3168  3168 E OpersysService: Unable to get IOpersys interface.
10-25 15:18:51.878  3168  3168 I OpersysService: test() returns 0
10-25 15:18:51.878  2995  2995 E OPERSYS : Inside write
10-25 15:18:51.878  2995  2995 E OPERSYS : Writing: Hello
10-25 15:18:51.878  2995  2995 D opersyshw_hal_module: OPERSYS HW - write()for 5 bytes called
10-25 15:18:51.879  2995  2995 E OPERSYS : Write returned 5
10-25 15:18:51.911 0 0 E SELinux : avc: denied { find } for pid=3168 uid=1000 name=android.hardware.opersys.IOpersys/default scontext=u:r:system_server:s0 tcontext=u:object_r:default_android_service:s0 tclass=service_manager permissive=1
10-25 15:18:51.911 0 0 I servicemanager: Caller(pid=3168,uid=1000,sid=u:r:system_server:s0) Found android.hardware.opersys.IOpersys/default in device VINTF manifest.
10-25 15:18:51.880  2995  2995 E OPERSYS : Inside read
10-25 15:18:51.880  2995  2995 D opersyshw_hal_module: OPERSYS HW - read()for 50 bytes called
10-25 15:18:51.880  2995  2995 E OPERSYS : Read returned 5
10-25 15:18:51.880  3168  3168 I OpersysService: read() returns Hello
10-25 15:18:51.914 0 0 E SELinux : avc: denied { add } for pid=3168 uid=1000 name=opersys scontext=u:r:system_server:s0 tcontext=u:object_r:default_android_service:s0 tclass=service_manager permissive=1
```

```
...
vsoc_x86_64:/ # service list | grep -i opersys
113   opersys: [android.os.IOpersysService]
```

Clearly SELinux isn't happy and complains quite loudly in the logs, but the system service does actually come up. Still, this isn't a viable approach. We've now disabled the entirety of SELinux/SEAndroid and this won't help us much.

2. Let's try another approach. Now that we've tried putting the system in permissive mode, let's try to retrieve some of the information we gathered in the logs to create SELinux rules that attempt to fix the problem. First, let's try to dump the SELinux errors we get into a text file on the device:

```
vsoc_x86_64:/ # logcat | grep -i opersys | grep avc > \
> /data/local/tmp/selinux-errors
```

Now, let's go back to the host, in the AOSP project and try to see how we can use the "audit2allow" tool to create rules that fix the problem:

```
$ adb pull /sys/fs/selinux/policy
$ adb pull /data/local/tmp/selinux-errors
$ audit2allow -i selinux-errors -p policy


#============= hal_light_cuttlefish ==============
allow hal_light_cuttlefish default_android_service:service_manager add;

#============= system_server ==============
allow system_server default_android_service:service_manager { add find };
```

As you can see, audit2allow seems able to generate rules to take care of the issues. Now, try feeding that into your build system and see what happens:

```
$ audit2allow -i selinux-errors -p policy > \
> device/google/cuttlefish/shared/sepolicy/vendor/my-rules.te
$ make
```

3. As you will have noticed in the previous exercise, the existing ruleset in Android won't allow you to just arbitrarily add rules to fix your problems. The "neverallow" rules included by default by Google preclude blanket additions such as those suggested to you by "audit2allow". You need to be more judicious in your changes and look at the errors more closely to see what needs to be changed. Try the following changes instead:

- Remove the the file you added in the earlier exercise:
  ```
  $ rm device/google/cuttlefish/shared/sepolicy/vendor/my-rules.te
  ```
- Add this snippet to the tail end of "device/google/cuttlefish/shared/sepolicy/vendor/file_contexts":
  ```
  # Opersys HAL/HIDL
  /vendor/lib(64)?/hw/android\.hardware\.opersys@2\.0-impl\.so         u:object_r:same_process_hal_file:s0
  /vendor/lib64/hw/opersyshw.default.so    u:object_r:same_process_hal_file:s0
  /dev/circchar            u:object_r:serial_device:s0
  ```
- Add this snippet to the tail end of "device/google/cuttlefish/shared/sepolicy/vendor/hal_light_cuttlefish.te"
  ```
  allow hal_light_cuttlefish serial_device:chr_file rw_file_perms;
  ```
- You will also need to modify the selinux policies in order to allow your system service to be registered at startup as well as making sure your HAL can be registered. If you fail

to do so then even if all your code is included at build time, the system service will fail to register and the HAL will fail to start at startup. To allow the registrations, you need to modify 2 files under the "system/sepolicy" directory:
- "private/service_contexts"
- "prebuilts/api/202404/private/service_contexts"

Both files have to be nearly identical and must be modified to add this entry – follow convention by inserting it in the alphabetical order:

```
android.hardware.opersys.IOpersys/default     :object_r:hal_light_service:s0
...
opersys                                      u:object_r:power_service:s0
```

We suggest you copy-paste the same lines into both files at the same place. You need to modify 2 files because once a version of Android is released by Google then the core security policies are supposed to be fixed in stone. Yet, here we are messing with those rules. So we need to act as if we had amended the prebuilt definitions that Google itself would have published.

Also, note that we are tagging our system service as belonging to the power_service domain and our HAL to the hal_light_service domain. This is a hack to simplify SELinux rule definitions. If we wanted to do this properly, we'd need to define a new domains called "opersys_service" and "hal_opersys_service" and make modifications to several other SELinux files. For the moment we're using an existing definition that already works.

- Compare these additions to those suggested to you by audit2allow. You should notice that these are far more targeted than the blanket rules suggested by the latter.

Now, rebuild your AOSP, restart Cuttlefish (don't forget the --noresume) and check your logs. You shouldn't see any errors any more. Still, this isn't a full fix, we are reusing existing contexts to avoid creating our own.

4. Go back to the added snippets and rework them to define new security contexts for the driver, the system service and the HAL. Specifically, add:
- opersys_device for /dev/circchar (instead of serial_device as above)
- opersys_service for the system service (instead of using power_service as above)
- hal_opersys_hwservice for the HAL
- hal_opersys_service for the HAL daemon (instead of using hal_light_service as above)

You'll also need to add the following files – have a look at other HALs for inspiration:
- system/sepolicy/private/hal_opersys.te
- system/sepolicy/vendor/hal_opersys_default.te
- device/google/cuttlefish/shared/sepolicy/vendor/hal_opersys_cuttlefish.te (instead of using the light HAL one as above)

You'll also need to modify the "device/google/cuttlefish/guest/hals/opersys/apex_file_contexts" to use "hal_opersys_cuttlefish_exec" instead of "hal_light_cuttlefish_exec" as was provided to you in the tarball.

This is not as easy as it seems. You'll need to look at existing definitions for other devices, system services and HAL layers. Try starting from the security contexts we suggested you use in the rules we used in the snippets earlier to get started. You are likely going to need to change files under both "private/" and "public/" subdirectories under "system/sepolicy/".

Once you've added your rules, restart Cuttlefish to validate that they work and that no SELinux errors are generated by any of the Opersys components.

Security Contexts for New Processes

In this section we will expand on the learning of the previous section with a slightly more difficult problem to solve since we won't have existing files from existing processes to start from to add our own SELinux/SEAndroid rules. Instead, we'll be adding new processes and their corresponding rules. Also, the current exercises have less "hand holding" than previous ones since it's assumed that you've now got a bit of experience with the different parts of the system. Feel free to refer to the previous exercises as a reference.

1. Get the client and server programs from http://www.linuxhowtos.org/C_C++/socket.htm. Place both in vendor/ and add the relevant Android.mk files to compile them in your AOSP. For example, to compile the C server:

- Add a new subdirectory:
```
$ cd ~/android/android-15.0.0_r5
$ mkdir -p vendor/example-server/
```
- Add the server.c file to the just-created subdirectory
- In recent versions of Android (including 15), you need to "#include <strings.h>" to the C file for it to build. Note that there already is a "string.h" (singular) that's present. That's not sufficient nor does it need to be removed or replaced. You really need to add "strings.h" (plural).
- Make sure "server" is added to a "PRODUCT_PACKAGES" list in [aosp]/device/google/cuttlefish/shared/phone/device_vendor.mk.
- Make sure you have an appropriate Android.bp within the "example-server" directory to get your app to build. Here's a sample:
```
cc_binary {
    name: "server",

    srcs: ["server.c"],

    shared_libs: [
        "libcutils",
    ],

    proprietary: true,
}
```

Follow a similar technique to add the client binary. For reference, once you rebuild your AOSP and restart your device, you should in principle be able to have the client and server interact in the following way:

To start the server (from the device's command line):
```
# server 4444 &
```
To connect to the server from the client:
```
# client localhost 4444
```
The client will prompt you for a message to send to the server and the server will ack reception.

2. Add an .rc file to the server build to have it get started by init at startup. Rebuild your AOSP

and check if the server does indeed start. If it doesn't, check the logs to see what is happening.

3. You should have noticed in the previous exercise that the init process refuses to start your server. The reason for this is that there isn't a security context defined for it and init won't allow a process to be spawned directly from it without a security context. In short, we don't want any child of init to get the same security context as initi itself. To fix this problem, you'll need to add the relevant rules in device/google/cuttlefish/shared/sepolicy/ to:

- Define example_server and example_client domains
- Use an domain_auto_trans() rule (or better yet, an init_daemon_domain()) to transition from init to example_server when the server is run.
- Enable example_server and example_client domains to communicate over sockets

4. (EXPERIMENTAL) Once your server is running, you can take this a step further and investigate how a server process can retrieve the client's SE context and operate on it. In the present case, modify the server you just added to use libselinux to retrieve the client's SE context and print it out when client connects.

Security Contexts for Special Apps

This is a fairly advanced exercise, but it's representative of what you may need to do in real life. Because it's an advanced exercse, this section also provides a lot less hand-holding that many of the previous exercises.

1. Modify Opersys stack (see the halext-opersys-cuttlefish-15.0.0_r5-no-se-250616.tar.bz2 contents downloaded above) to start as separate platform-signed persistent APK with its own user ID:
   - put under top-level "vendor/" directory
   - android:persistent = true (see packages/services/Telephony)
   - Shared USER ID == one in the OEM range (look for "OEM" in system/core/include/cutils/android_filesystem_config.h)

2. Change SELinux rules to match the persistent app just added instead of a built-in system service as in the previous exercise.
   - Move all non-hidl definitions to device/google/cuttlefish/shared/sepolicy files
   - Use seapp_contexts to set domain based on user ID and app package name

SELinux in the Kernel

This section involves driver and kernel development. It may or may not apply to the type of task you need to do. In order to carry out this exercise, you will need to get the kernel sources for your AOSP along with the sources for the module for the Opersys service that we provided to you in binary form as part of the package you downloaded at the beginning of this class to conduct the previous exercises. Refer to the exercises document from our Embedded Android class (https://www.opersys.com/training/embedded-android-training/) and specifically to the "Kernel Basics" and "Linux Device Driver" exercise sections.

1. Modify the circular character driver to print the security context SID of the process opening it. Have a look at:

- struct file's f_security field
- security/selinux/hooks.c
- file_security_struct in security/selinux/include/objsec.h

2. (optional) There's a security_sid_to_context() in security/selinux/include/security.h and security/selinux/ss/services.c, but it's not exported for modules to use. Try exposing it using EXPORT_SYMBOL() and use it in your module to convert the SID to a string context and print it out.

AOSP User-Space (extra)

The exercises are extra exercises which may or may not be relevant within the context of your instance of this run of the class. To be discussed with instructor.

1. Generate a pair of adb keys using “adb keygen”

2. Create your own cert for signing the system service APK created in the previous exercises and integrate that new cert into your AOSP build and SELinux rules. There is some “googling” involved here.

3. Generate your own release keys for slide 343

4. Generate a full OTA zip file (see slide 345)

5. Make a modification to your AOSP (your choice) and rebuild the AOSP

6. Generate an incremental OTA zip file (see slide 346)

7. Modify your service and client commands from the earlier section to make them updatable as an Apex.