MANDIANT®

A FireEye® Company

Navigating SEAndroid Trust Relationships

Exploitation Techniques for Modern Android Devices

SECURITY CONSULTING

Who Am I

Jake Valletta

- Manager, Professional Services
- Joined Mandiant in 2011
  - 6 years in Information Security
  - Mobile security, penetration testing, forensics/IR, education
- Android enthusiast & beer drinker
- San Francisco, CA
- @jake_valletta

Agenda

- Introduction to SEAndroid
- Threat Landscape prior to SEAndroid
- Platform Exploitation Techniques on Modern Devices
- Q & A

Goals, Hopes & Dreams

- Help people understand SEAndroid
  - How it works fundamentally
  - How it's helping secure your Android device
- Provide a starting point for interested people to get exploring
- Prove that Android security isn't terrible.

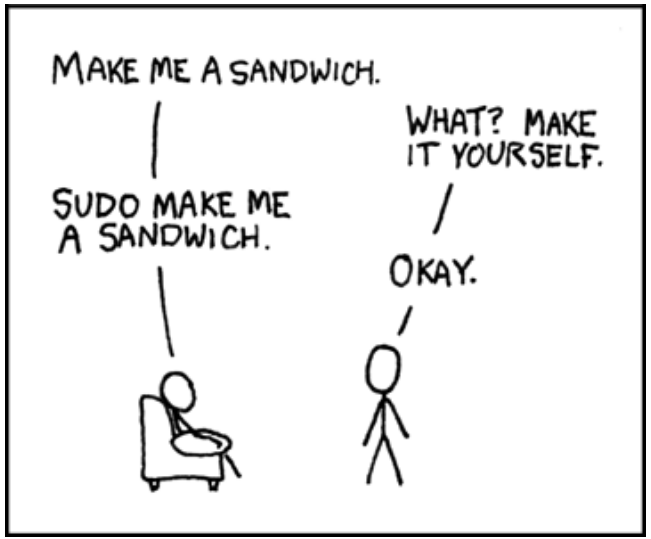
Introduction to SEAndroid

Access Controls – DAC vs. MAC

- Access Controls restrict a **subject's** access to an **object**
- Discretionary Access Control ("DAC")
  - Subject can pass controls to another subject
    - Accidentally or intentionally
  - Examples: Linux file permissions (read/write/execute)
- Mandatory Access Control ("MAC")
  - Security is governed by a central security policy administrator (i.e. the kernel)
  - Owners are unable to override policy
  - Examples: SEAndroid, TrustedBSD (Apple iOS)

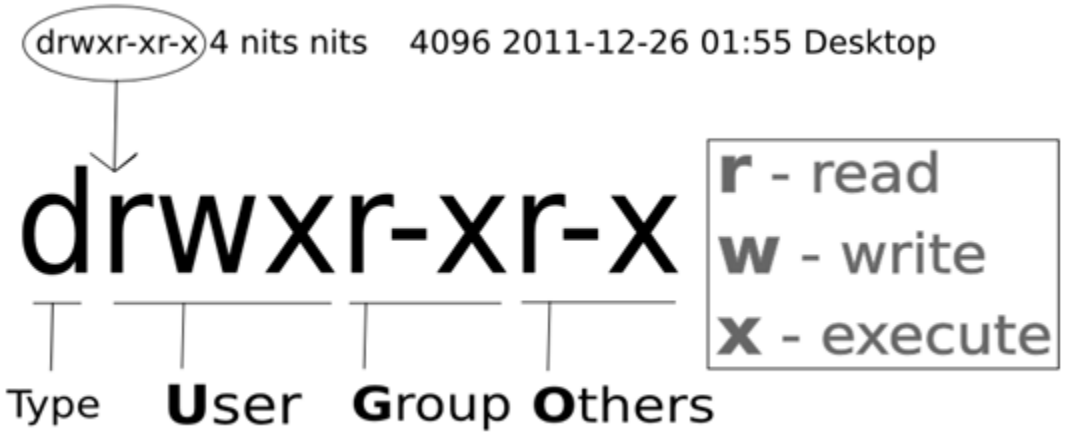
Access Controls – Linux DAC Weaknesses

- "root" user is **too powerful**
- Access to objects is entirely at discretion of owner
- Certain objects lack checks (or can be overridden)
  - Sockets, "ioctl" calls
  - "sudo" users
- Doesn't allow fine-grained control
  - Relies on Linux user/group



Access Controls – Linux DAC Example

- Linux file permissions include 9-bits for manipulating file access



Access Controls – Linux DAC Example (cont.)

1. Alice creates a file to store sensitive information
   - By default, Alice owns this file
2. Alice wants to share the file so that Bob can access it
   - Since Alice owns this file, she is allowed to alter the permissions
3. Alice sets the “other” permission bits of the file so that other users can read the file
   - Alice doesn’t realize **all** users can now view this file
4. Eve finds and accesses Alice’s file
   - Alice is sad

SELinux + SEAndroid

- Security Enhancements for Linux (“SELinux”)
  - Originally developed by NSA in 1989
- SELinux ported to Android as SE for Android (“SEAndroid”)
- Introduced by Google into Android Open-Source Project (“AOSP”) in 2012
  - Added to tree in 4.2 (Q4 2012)
  - Enabled in **permissive mode** by default in 4.3 (Q3 2013)
  - Enabled in **enforce mode** by default in 5.0 (Q4 2014)

SEAndroid Modes

- Permissive Mode
  - Log all violations
  - Not really doing anything
- Enforcing Mode
  - Blocks (and optionally logs) violations
  - **Required!**

SEAndroid Goals

- Restrict high value targets on Android
  - System daemons
  - Devices, sockets, sysfs/procfs
- Further isolate/sandbox applications
  - Linux DAC + SEAndroid
- Neuter the “root” user

Labels & Rule Format

- All objects receive a label
- **Domain –** Label for a process(es)
- **Type –** Label for an object (“myfile”, “testbin_socket”)
- **Class –** The category of object (file, socket, property_service)
- **Permission** – The operation/action being performed (read, write, open, ioctl)

```
allow domains types:classes permissions;
```

myfirstrule.jpg

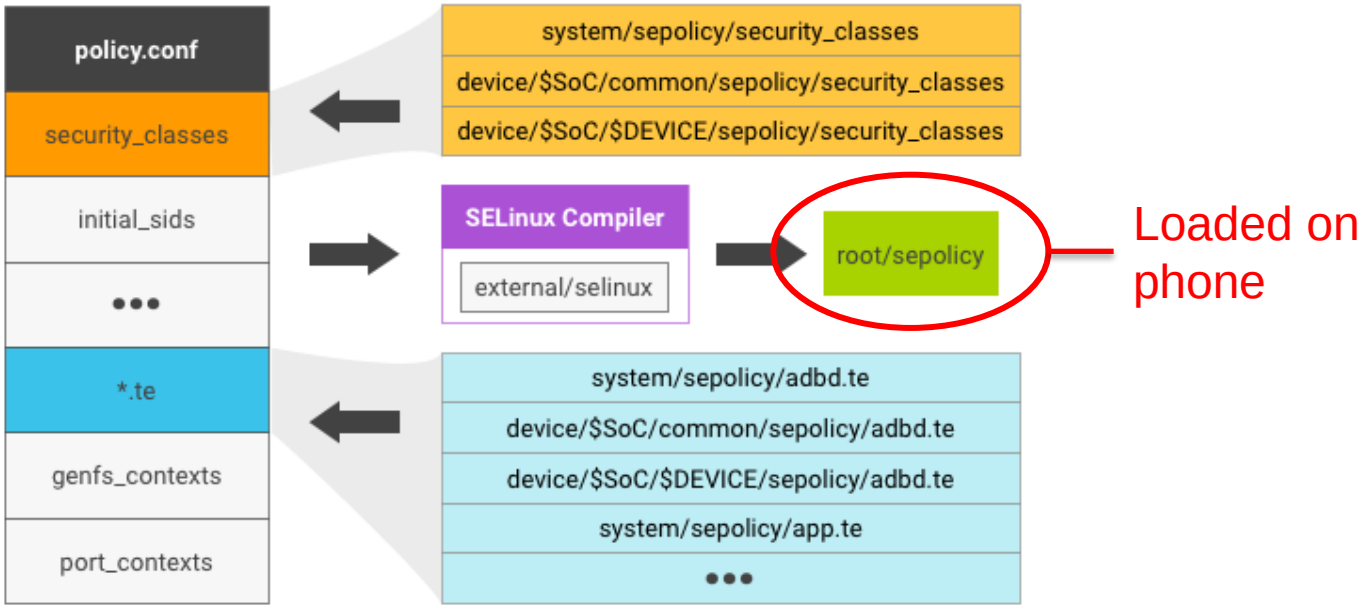
- New binary “/system/bin/mytool” needs to read from “/system/etc/mytool.config”
  - “/system/bin/mytool” is labeled as “mytool_file”
  - “/system/bin/mytool.config” is labeled as “mytool_conf”
- Add a rule to create the mapping:

```
allow mytool_file mytool_conf:file { read };
```

Rules, Rules, Rules

- Rules are defined in “*.te” files
- Google provides a working starting point for OEMs
  - Macros
  - Dozens of “*.te” files
  - Class definitions
- OEM/ODM/SoC vendors will need to add and change content per device
  - We’ll be focusing on this later

Compiling Rules



Files of Interest

- **/sepolicy** – Binary policy containing rules
- **/file_contexts(.bin)** – Labeling rules for files
- **/service_contexts** – Labeling rules for system services
- **/property_contexts** – Labeling rules for properties
- **/system/etc/security/mac_permissions.xml** – Labeling rules for applications

Attacking Android without SEAndroid

Darker Days: Race to “root”

- Attacking Android used to be a race to “root”
  - Many core Android daemons run as “root”
  - “root” can load kernel modules
- Usually required 1-2 exploits to compromise a device

Darker Days: GingerBreak

- CVE-2011-1823 / “GingerBreak”
- Exploits volume manager daemon (“vold”) which runs as “root”
- Third-party application → “root” user (1 exploit)

Darker Days: GingerBreak

1. Obtain PID of “vold” process => Read “/proc/pid/cmdline”
2. Obtain addresses of “vold” => Walk Global Offset Table
3. Send netlink message to “vold” => Open AF_NETLINK socket
4. Trigger exploit code to run => sendmsg() + non-system binary executed
5. Exploit code remounts drive => remount + chmod()/chown()
6. Execute final priv. esc. payload => execute “setuid” binary

Darker Days: GingerBreak vs. SEAndroid

1. Obtain PID of “vold” process => ~~**Read “/proc/pid/cmdline”**~~ **[Blocked]**
2. Obtain addresses of “vold” => ~~**Walk Global Offset Table**~~ **[Blocked]**
3. Send netlink message to “vold” => ~~**Open AF_NETLINK socket**~~ **[Blocked]**
4. Trigger exploit code to run => ~~**sendmsg() + non-system binary executed**~~ **[Blocked]**
5. Exploit code remounts drive => ~~**remount + chmod()/chown()**~~ **[Blocked]**
6. Execute final priv. esc. payload => execute “setuid” binary **Allowed? But still running same context…**

Darker Days: WeakSauce

- Local privilege escalation to “root” through unprotected UNIX socket + daemon
- Introduced by HTC-branded devices
- Found and reported by @jcase as “WeakSauce” in 2014
- Third-party application (with INTERNET permission) → “root” code execution (1 exploit)

Darker Days: WeakSauce vs. SEAndroid

- Local privilege escalation to “root” through unprotected UNIX socket + daemon
- Introduced by HTC-branded devices
- Found and reported by @jcase as “WeakSauce” in 2014
- Third-party application (with INTERNET permission) → “root” code execution (1 exploit)

Socket interactions in “/dev/socket/” restricted by default SEAndroid policy

Darker Days: CVE-2016-2060

- Local privilege escalation in “netd” daemon to “radio” user on Qualcomm SoC devices
  - Hundreds of devices
  - Android 2.3 – 5.0 (5 years+)
- Reported to Qualcomm by Mandiant in May 2015
- Third-party application → “radio” user (1 exploit)

Darker Days: CVE-2016-2060

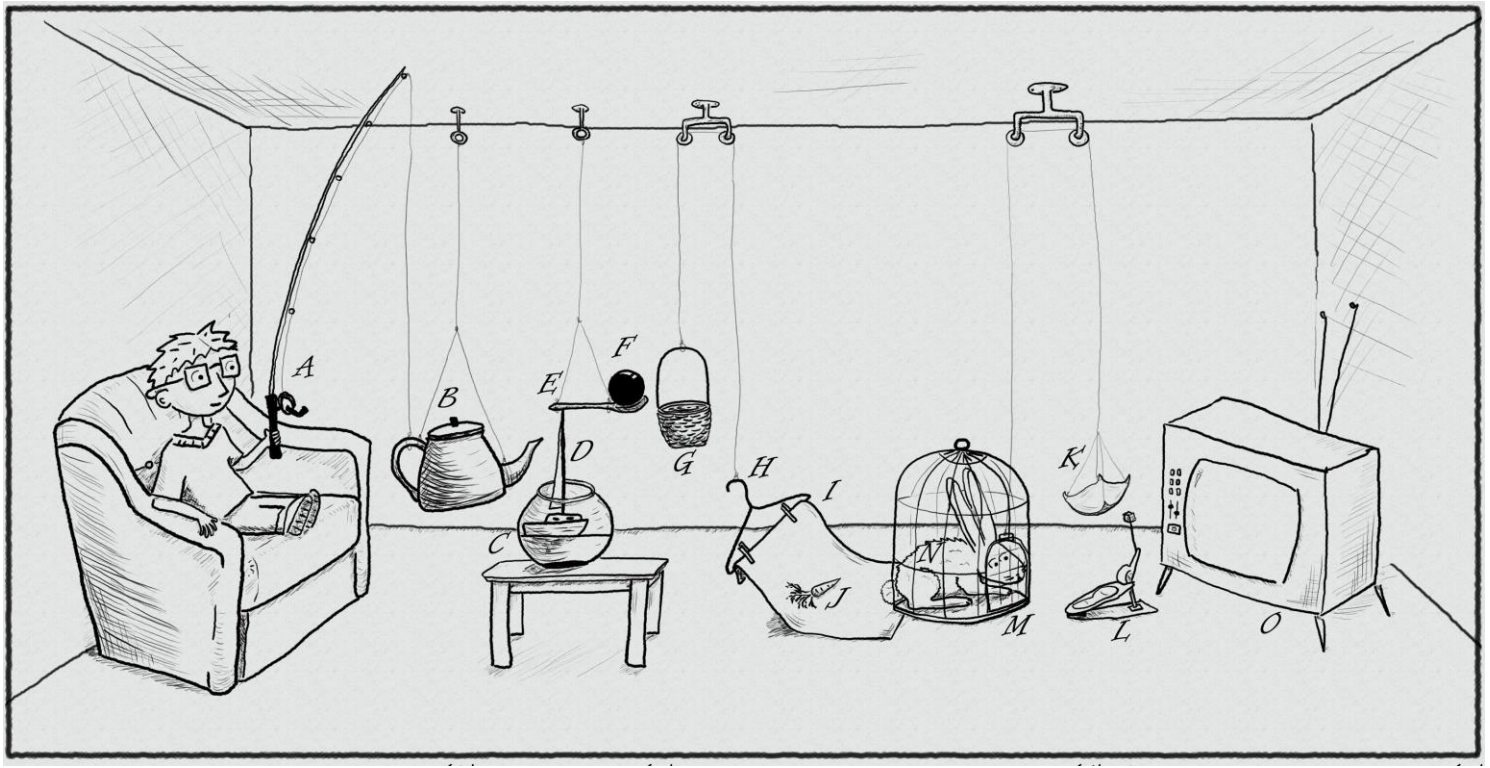
- CVE-2016-2060 allows…
  - Access to contact and SMS/MMS DB
  - Access to radio baseband devices
  - Privileged filesystem access
  - Alter/influence network stack
  - Ability to modified privileged system properties

Darker Days: CVE-2016-2060 vs. SEAndroid

- CVE-2016-2060 allows…
  - ~~**Access to contact and SMS/MMS DB**~~
  - **Limited** access to radio baseband devices
  - ~~**Privileged filesystem access**~~
  - Alter/influence network stack
  - **Limited** ability to modified privileged system properties

Exploitation Techniques for Modern Devices

Rube-Goldberg Reality



Target the Context, not the User

- Key questions to ask:
  - *What's our objective?*
  - *What have the OEMs added?*
  - *Has AOSP policy been altered?*

Step 1 – Obtain SEAndroid Files

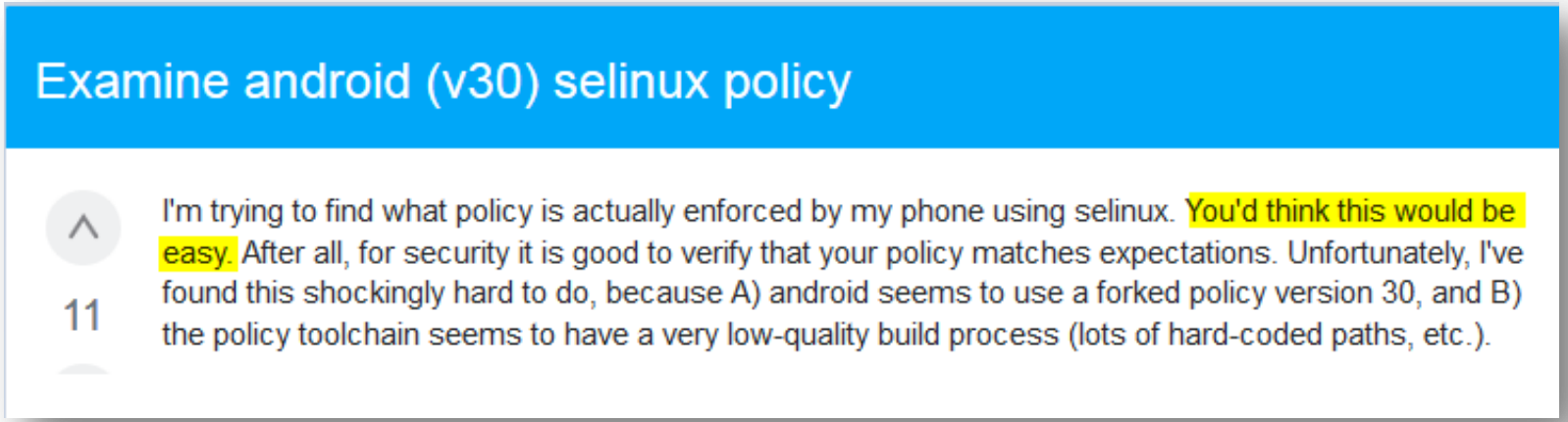
- Use Android Debugging Bridge (“adb”)
  - “shell” user is able to pull policy files
- Most files are cleartext
  - “file_contexts” was converted to binary format in Nougat
  - https://github.com/jakev/sefcontext-parser
- “sepolicy” file is another story…

Step 2 – Pick a Target

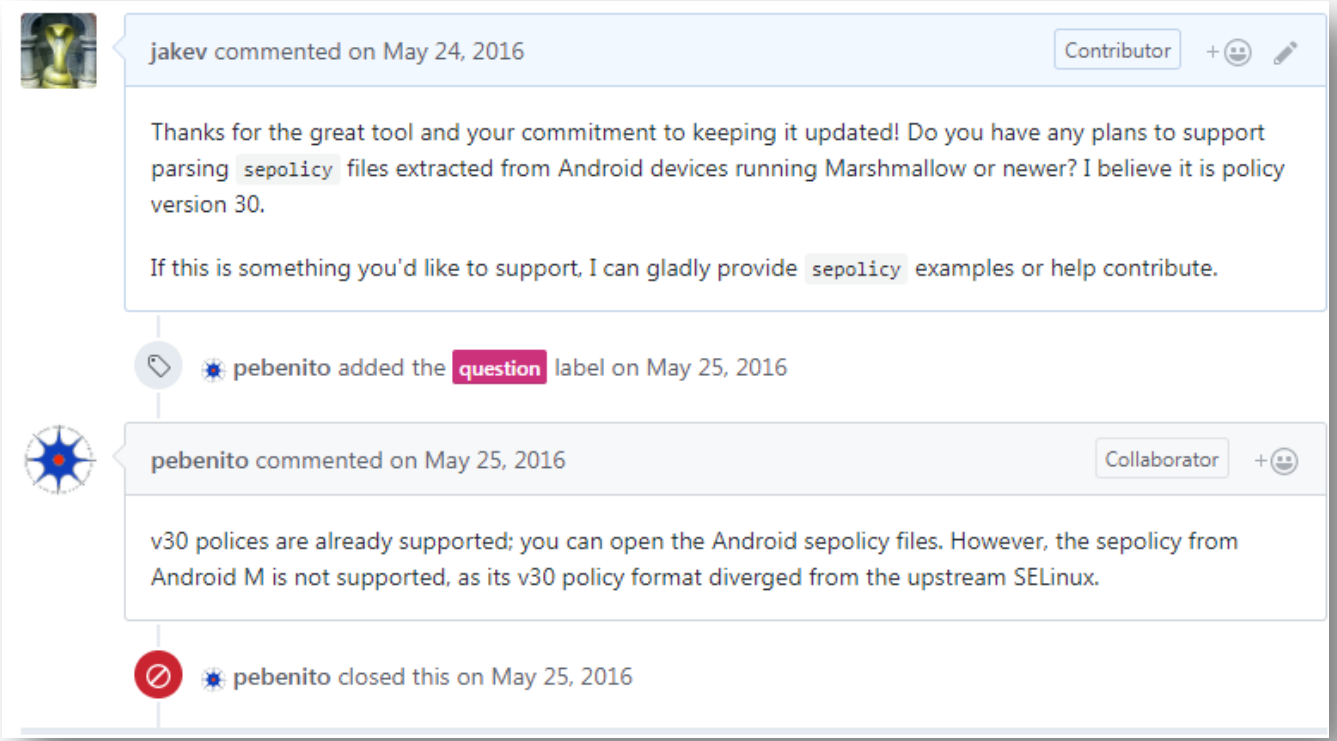
```
03:12:32 /Testing/Honor6X_7.0.0$ adb shell
bullhead:/ $ id
uid=2000(shell) gid=2000(shell) groups=2000(shell),1004(input),1007(log
),1011(adb),1015(sdcard_rw),1028(sdcard_r),3001(net_bt_admin),3002(net_
bt),3003(inet),3006(net_bw_stats),3009(readproc) context=u:r:shell:s0
```

Step 3 – Parse “sepolicy”

- Goal is to use `sesearch` to run queries against “sepolicy”
- Parsing “sepolicy” rules file is surprisingly difficult
  - Google has deviated from SELinux project and **is not merging upstream**
  - There is no clear build chain for building Google’s port of libsepol



Step 3 – Parse “sepolicy” (cont.)



jakev commented on May 24, 2016 — Contributor

Thanks for the great tool and your commitment to keeping it updated! Do you have any plans to support parsing `sepolicy` files extracted from Android devices running Marshmallow or newer? I believe it is policy version 30.

If this is something you'd like to support, I can gladly provide `sepolicy` examples or help contribute.

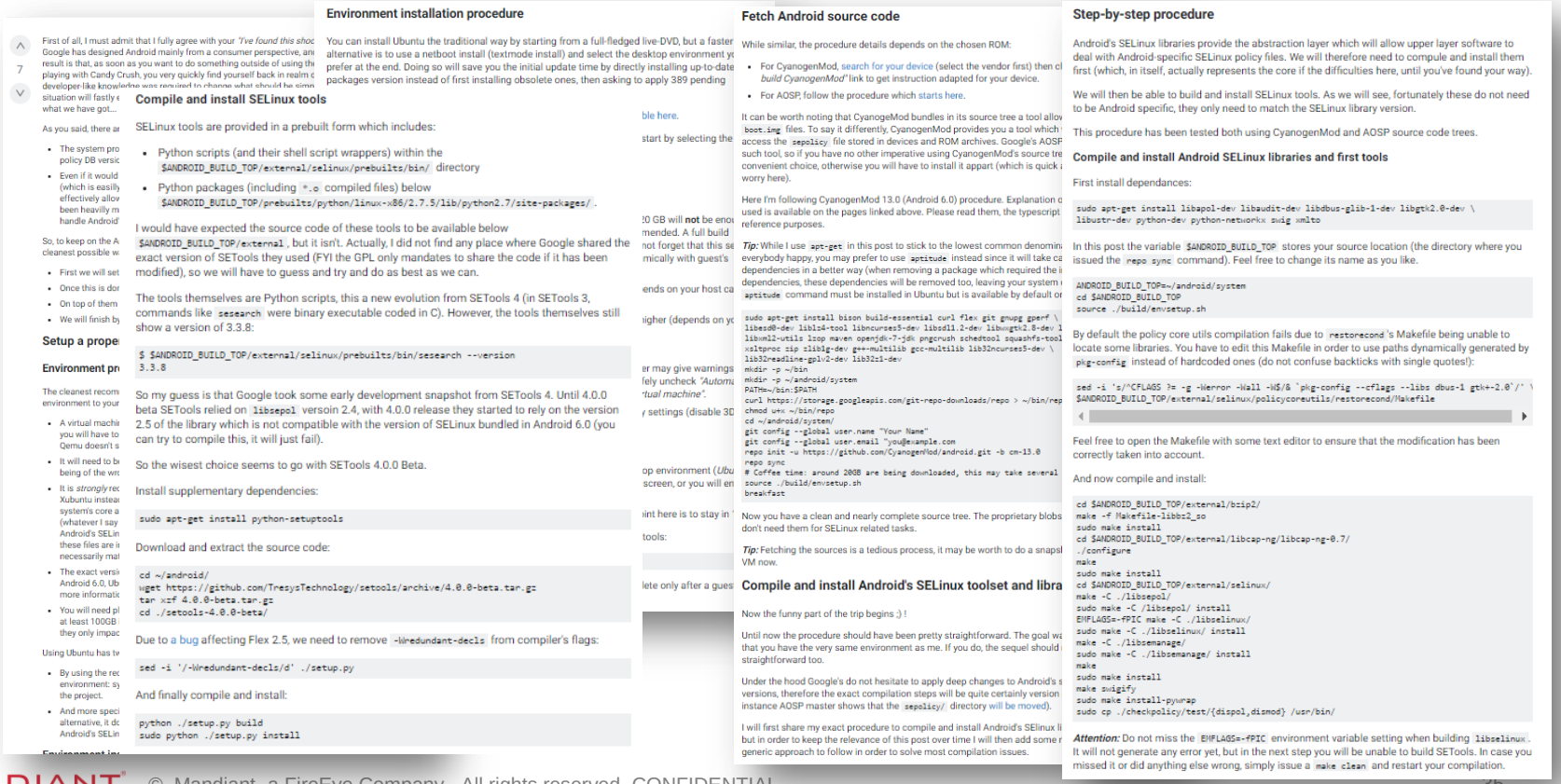
pebenito added the **question** label on May 25, 2016

pebenito commented on May 25, 2016 — Collaborator

v30 polices are already supported; you can open the Android sepolicy files. However, the sepolicy from Android M is not supported, as its v30 policy format diverged from the upstream SELinux.

pebenito closed this on May 25, 2016

Step 3 – Parse “sepolicy” (cont.)



Compile and install SELinux tools

SELinux tools are provided in a prebuilt form which includes:

- Python scripts (and their shell script wrappers) within the `$ANDROID_BUILD_TOP/external/selinux/prebuilts/bin/` directory
- Python packages (including `*.o` compiled files) below `$ANDROID_BUILD_TOP/prebuilts/python/linux-x86/2.7.5/lib/python2.7/site-packages/`.

I would have expected the source code of these tools to be available below `$ANDROID_BUILD_TOP/external`, but it isn't. Actually, I did not find any place where Google shared the exact version of SETools they used (FYI the GPL only mandates to share the code if it has been modified), so we will have to guess and try and do as best as we can.

The tools themselves are Python scripts, this a new evolution from SETools 4 (in SETools 3, commands like `sesearch` were binary executable coded in C). However, the tools themselves still show a version of 3.3.8:

```
$ $ANDROID_BUILD_TOP/external/selinux/prebuilts/bin/sesearch --version
3.3.8
```

So my guess is that Google took some early development snapshot from SETools 4. Until 4.0.0 beta SETools relied on `libsepol` versoin 2.4, with 4.0.0 release they started to rely on the version 2.5 of the library which is not compatible with the version of SELinux bundled in Android 6.0 (you can try to compile this, it will just fail).

So the wisest choice seems to go with SETools 4.0.0 Beta.

Install supplementary dependencies:

```
sudo apt-get install python-setuptools
```

Download and extract the source code:

```
cd ~/android/
wget https://github.com/TresysTechnology/setools/archive/4.0.0-beta.tar.gz
tar xzf 4.0.0-beta.tar.gz
cd ./setools-4.0.0-beta/
```

Due to a bug affecting Flex 2.5, we need to remove `-Wredundant-decls` from compiler's flags:

```
sed -i '/-Wredundant-decls/d' ./setup.py
```

And finally compile and install:

```
python ./setup.py build
sudo python ./setup.py install
```

Environment installation procedure

You can install Ubuntu the traditional way by starting from a full-fledged live-DVD, but a faster alternative is to use a netboot install (textmode install) and select the desktop environment you prefer at the end. Doing so will save you the initial update time by directly installing up-to-date packages version instead of first installing obsolete ones, then asking to apply 389 pending

Fetch Android source code

While similar, the procedure details depends on the chosen ROM:

- For CyanogenMod, search for your device (select the vendor first) then cl *build CyanogenMod* link to get instruction adapted for your device.
- For AOSP, follow the procedure which starts here.

It can be worth noting that CyanogeMod bundles in its source tree a tool allo `boot.img` files. To say it differently, CyanogenMod provides you a tool which access the `sepolicy` file stored in devices and ROM archives. Google's AOSP such tool, so if you have no other imperative using CyanogenMod's source tre convenient choice, otherwise you will have to install it appart (which is quick worry here).

Here I'm following CyanogenMod 13.0 (Android 6.0) procedure. Explanation used is available on the pages linked above. Please read them, the typescript reference purposes.

Tip: While I use `apt-get` in this post to stick to the lowest common denomin everybody happy, you may prefer to use `aptitude` instead since it will take ca dependencies in a better way (when removing a package which required the i dependencies, these dependencies will be removed too, leaving your system `aptitude` command must be installed in Ubuntu but is available by default o

```
sudo apt-get install bison build-essential curl flex git gnupg gperf \
libesd0-dev liblz4-tool libncurses5-dev libsdl1.2-dev libwxgtk2.8-dev l
libxml2-utils lzop maven openjdk-7-jdk pngcrush schedtool squashfs-tool
xsltproc zip zlib1g-dev g++-multilib gcc-multilib lib32ncurses5-dev \
lib32readline-gplv2-dev lib32z1-dev
mkdir -p ~/bin
mkdir -p ~/android/system
PATH=~/bin:$PATH
curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/rep
chmod u+x ~/bin/repo
cd ~/android/system/
git config --global user.name "Your Name"
git config --global user.email "you@example.com
repo init -u https://github.com/CyanogenMod/android.git -b cm-13.0
repo sync
# Coffee time: around 20GB are being downloaded, this may take several
source ./build/envsetup.sh
breakfast
```

Now you have a clean and nearly complete source tree. The proprietary blobs don't need them for SELinux related tasks.

Tip: Fetching the sources is a tedious process, it may be worth to do a snapsh VM now.

Compile and install Android's SELinux toolset and libra

Now the funny part of the trip begins ;) !

Until now the procedure should have been pretty straightforward. The goal wa that you have the very same environment as me. If you do, the sequel should straightforward too.

Under the hood Google's do not hesitate to apply deep changes to Android's versions, therefore the exact compilation steps will be quite certainly version instance AOSP master shows that the `sepolicy/` directory will be moved).

I will first share my exact procedure to compile and install Android's SELinux li but in order to keep the relevance of this post over time I will then add some generic approach to follow in order to solve most compilation issues.

Step-by-step procedure

Android's SELinux libraries provide the abstraction layer which will allow upper layer software to deal with Android-specific SELinux policy files. We will therefore need to compile and install them first (which, in itself, actually represents the core if the difficulties here, until you've found your way).

We will then be able to build and install SELinux tools. As we will see, fortunately these do not need to be Android specific, they only need to match the SELinux library version.

This procedure has been tested both using CyanogenMod and AOSP source code trees.

Compile and install Android SELinux libraries and first tools

First install dependances:

```
sudo apt-get install libapol-dev libaudit-dev libdbus-glib-1-dev libgtk2.0-dev \
libustr-dev python-dev python-networkx swig xmlto
```

In this post the variable `$ANDROID_BUILD_TOP` stores your source location (the directory where you issued the `repo sync` command). Feel free to change its name as you like.

```
ANDROID_BUILD_TOP=~/android/system
cd $ANDROID_BUILD_TOP
source ./build/envsetup.sh
```

By default the policy core utils compilation fails due to `restorecond`'s Makefile being unable to locate some libraries. You have to edit this Makefile in order to use paths dynamically generated by `pkg-config` instead of hardcoded ones (do not confuse backticks with single quotes!):

```
sed -i 's/^CFLAGS ?= -g -Werror -Wall -W$/& `pkg-config --cflags --libs dbus-1 gtk+-2.0`/' \
$ANDROID_BUILD_TOP/external/selinux/policycoreutils/restorecond/Makefile
```

Feel free to open the Makefile with some text editor to ensure that the modification has been correctly taken into account.

And now compile and install:

```
cd $ANDROID_BUILD_TOP/external/bzip2/
make -f Makefile-libbz2_so
sudo make install
cd $ANDROID_BUILD_TOP/external/libcap-ng/libcap-ng-0.7/
./configure
make
sudo make install
cd $ANDROID_BUILD_TOP/external/selinux/
make -C ./libsepol/
sudo make -C /libsepol/ install
EXTRA_CFLAGS=-fPIC make -C ./libselinux/
sudo make -C ./libselinux/ install
make -C ./libsemanage/
sudo make -C ./libsemanage/ install
make
sudo make install
make swigify
sudo make install-pywrap
sudo cp ./checkpolicy/test/{dispol,dismod} /usr/bin/
```

Attention: Do not miss the `EXTRA_CFLAGS=-fPIC` environment variable setting when building `libselinux`. It will not generate any error yet, but in the next step you will be unable to build SETools. In case you missed it or did anything else wrong, simply issue a `make clean` and restart your compilation.

Step 3 – Parse “sepolicy” (cont.)

- Finally can use `sesearch` utility against rule file to run queries

```
analyst@aosp-7:~/setools$ sesearch --allow -s shell -c file -p write sepolicy
allow appdomain app_fuse_file:file { read write getattr append };
allow appdomain backup_data_file:file { read write getattr };
allow appdomain cache_backup_file:file { read write getattr };
allow appdomain cgroup:file { read lock getattr write ioctl open append };
allow appdomain fuse:file { rename setattr read lock create getattr write ioctl unlink open append };
allow appdomain qtaguid_proc:file { read lock getattr write ioctl open append };
allow appdomain radio_data_file:file { read write getattr };
allow appdomain ringtone_file:file { read write getattr };
allow appdomain sdcardfs:file { rename setattr read lock create getattr write ioctl unlink open append };
allow appdomain selinuxfs:file { read lock getattr write ioctl open append };
allow appdomain shell_data_file:file { write getattr };
allow appdomain user_profile_data_file:file { rename setattr read lock create getattr write ioctl unlink open append };
allow appdomain vfat:file { read lock getattr write ioctl open append };
allow appdomain wallpaper_file:file { read write getattr };
allow domain cgroup:file { write lock open append };
allow domain debugfs_trace_marker:file { write lock open append };
allow domain debugfs_tracing:file { write lock open append };
allow shell app_data_file:file { rename setattr read lock create getattr write ioctl unlink open append };
allow shell bootchart_data_file:file { rename setattr read lock create getattr write ioctl unlink open append };
allow shell debugfs_tracing:file { read lock getattr write ioctl open append };
allow shell media_rw_data_file:file { rename setattr read lock create getattr write ioctl unlink open append };
allow shell shell:file { read lock getattr write ioctl open append };
allow shell shell_data_file:file { rename execute setattr read lock create getattr execute_no_trans write ioctl unlink open append };
allow shell shell_tmpfs:file { read write execute };
allow shell sysfs_huawei_sensor:file { read write getattr open };
```

Step 4a – Map to a File/Service/Property

```
01:44:30 /Testing/Honor6X_7.0.0/seandroid$ grep -r sysfs_huawei_sensor file_contexts |\
> awk '{print $1}'
/sys/class/sensors/acc_sensor/calibrate
/sys/class/sensors/acc_sensor/calibrate_timeout
/sys/class/sensors/acc_sensor/info
/sys/class/sensors/acc_sensor/self_test
/sys/class/sensors/acc_sensor/self_test_timeout
/sys/class/sensors/airpress_sensor/read_airpress
/sys/class/sensors/airpress_sensor/set_calidata
/sys/class/sensors/gyro_sensor/self_test
/sys/class/sensors/gyro_sensor/self_test_timeout
/sys/class/sensors/handpress_sensor/calibrate
/sys/class/sensors/handpress_sensor/calibrate_timeout
/sys/class/sensors/handpress_sensor/info
/sys/class/sensors/handpress_sensor/self_test
/sys/class/sensors/handpress_sensor/self_test_timeout
/sys/class/sensors/mag_sensor/info
/sys/class/sensors/mag_sensor/self_test
/sys/class/sensors/mag_sensor/self_test_timeout
/sys/class/sensors/ois_sensor/ois_ctrl
/sys/class/sensors/ps_sensor/calibrate
/sys/class/sensors/ps_sensor/calibrate_timeout
/sys/devices/platform/huawei_sensor/acc_calibrate
/sys/devices/platform/huawei_sensor/acc_info
/sys/devices/platform/huawei_sensor/als_info
/sys/devices/platform/huawei_sensor/gyro_selfTest
/sys/devices/platform/huawei_sensor/handpress_calibrate
/sys/devices/platform/huawei_sensor/handpress_info
/sys/devices/platform/huawei_sensor/handpress_selfTest
/sys/devices/platform/huawei_sensor/mag_info
/sys/devices/platform/huawei_sensor/mag_selfTest
/sys/devices/platform/huawei_sensor/ois_ctrl
```

Step 4a – Map to a File/Service/Property

- Properties can lead to privilege escalation
  - “default_ctl_prop” labeled properties can start/stop “init” services
  - “system_prop” labeled properties can alter a lot of things normal apps shouldn’t be able to
  - Look for non-AOSP properties!!!

Step 4a – Map to a File/Service/Property

```
analyst@aosp-7:~/setools$ sesearch -A -s netd -c property_service -p set sepolicy
allow netd ctl_mdnsd_prop:property_service set;
allow netd system_prop:property_service set;
```

```
05:50:14 /Testing/Honor6X_7.0.0/seandroid$ grep -r system_prop property_contexts
net.                    u:object_r:system_prop:s0
dev.                    u:object_r:system_prop:s0
ro.runtime.             u:object_r:system_prop:s0
hw.                     u:object_r:system_prop:s0
ro.hw.                  u:object_r:system_prop:s0
sys.                    u:object_r:system_prop:s0
service.                u:object_r:system_prop:s0
wlan.                   u:object_r:system_prop:s0
persist.sys.            u:object_r:system_prop:s0
persist.service.        u:object_r:system_prop:s0
persist.security.       u:object_r:system_prop:s0
camera.fbcdraw.enable   u:object_r:system_prop:s0
camera.dump.raw2yuv     u:object_r:system_prop:s0
camera.tunning.dump     u:object_r:system_prop:s0
camera.debug.dual.mode  u:object_r:system_prop:s0
camera3.zsl.switch      u:object_r:system_prop:s0
dump.preview.flag       u:object_r:system_prop:s0
dump.raw.flag           u:object_r:system_prop:s0
dump.feraw.flag         u:object_r:system_prop:s0
dump.savekey.value      u:object_r:system_prop:s0
dump.interval.flag      u:object_r:system_prop:s0
ctl.inputlogcat                 u:object_r:system_prop:s0
```

ro. == “read only”

Non-AOSP!

Step 4b – Determine Privileged Contexts

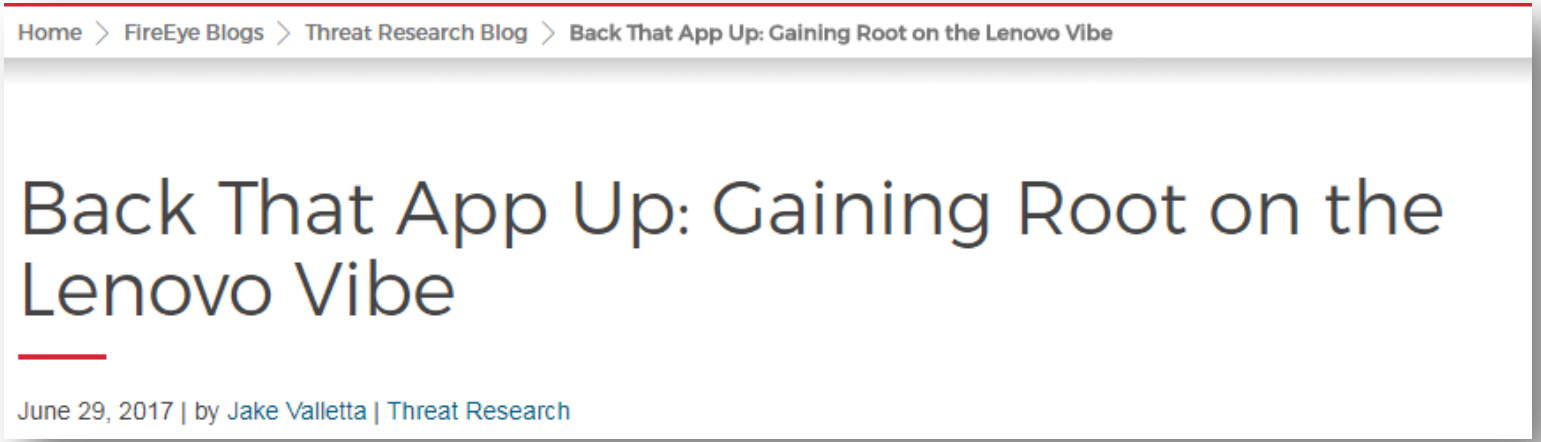
- Special type of class: “capability”
  - dac_override – Ignore DAC checks (!!)
  - dac_override_read – Ignore DAC read checks
  - sys_module – Load kernel modules (!!)
  - fowner – **5 capabilities in one** - Override all file owner requirements (e.g. for chmod, setxattr)

```
analyst@aosp-7:~/setools$ sesearch -A -c capability sepolicy |grep sys_module
allow atcmdserver atcmdserver:capability { sys_module sys_nice dac_override net_raw sys_admin fsetid net_admin fowner };
allow hi110x_daemon hi110x_daemon:capability { setuid sys_module net_raw sys_nice dac_override dac_read_search chown fsetid
setgid net_admin fowner };
allow netd netd:capability { setuid sys_module dac_override net_raw chown fsetid kill setgid net_admin fowner };
```

Full list: https://selinuxproject.org/page/ObjectClassesPerms

Case Study: CVE-2017-3748, 3749, 3750

- Series of vulnerabilities that lead code execution as “root” on (a lot of) Lenovo devices
  - “root” user *and* very powerful context
- Reported to Lenovo by Mandiant in 2016



Case Study: CVE-2017-3748, 3749, 3750

- Target: “/system/bin/nac_server”
  - Running as root and “nac_server” SEAndroid domain
  - Listens on local UNIX sockets for commands
  - Executes files in ultra powerful “root_channel” context
  - **Incorrectly validates incoming commands (CVE-2017-3748, CVE-2017-3750)**

Access to socket blocked by SEAndroid!

```
04:23:40 /$ adb shell /data/local/tmp/socat abstract-connect:supercmdlocalsocket -
2015/02/05 15:32:50 socat[15987.4706304] E connect(3, AF=1 "\0supercmdlocalsocket", 22): Permission denied
04:23:51 /$
```

Case Study: CVE-2017-3748, 3749, 3750

- Use `sesearch` to determine who can access socket
  - Only “system_app” or “platform_app”
- Need code execution in a system application
  - Local backups enabled in Lenovo “Idea Friend” application (CVE-2017-3749)

```
04:40:10 /$ adb shell ps -Z|grep ideafriend
u:r:platform_app:s0            u0_a41    4623  350   com.lenovo.ideafriend
04:40:19 /$
```

Case Study: CVE-2017-3748, 3749, 3750

- Still not a complete compromise!!
  - Can’t disable SEAndroid
  - Can’t remount file systems

```
01:03:59 /DevTesting/LenovoVibe/r21b_2$ sh root.sh
Connect phone. OK.
Pushing busybox...
[100%] /data/local/tmp/busybox
Restoring com.lenovo.security...
Now unlock your device and confirm the restore operation.
Once restore is complete, press enter.

Restoring com.lenovo.ideafriend...
Now unlock your device and confirm the restore operation.
Once restore is complete, press enter.

Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
root@passion:/ # id
uid=0(root) gid=0(root) context=u:r:root_channel:s0
root@passion:/ #
```

Closing Thoughts

Wrap Up

- If you're phone was released after 2014, you're likely being protected by SEAndroid
- SEAndroid has greatly complicated the attack surface for compromising a device
- Exploitation is no longer as "find a root process and exploit it" to fully compromise a device

Additional Resources

- “The Case for Security Enhanced (SE) Android”, 2012
  - Stephen Smalley, NSA
- “Honey, I Shrunk the Attack Surface”, 2017
  - Nick Kralevich, Google
- SEAndroid Wiki:
  - http://selinuxproject.org/page/NB_SEforAndroid_1
  - http://selinuxproject.org/page/NB_SEforAndroid_2
- Android Documentation:
  - https://source.android.com/security/selinux/

Questions & Answers

Contact Me

- **Email**
  - jake.valletta@mandiant.com
  - javallet@gmail.com
- **Twitter**
  - @jake_valletta
- **Additional Content by Me**
  - FireEye Threat Research Blog
  - https://blog.thecobraden.com/
  - https://www.thecobraden.com/
  - https://github.com/jakev/