The Root of All Evil

How Dangerous is Rooting Your Android?

Dominik Gedon

May 15, 2018

Security Research Group
Department of Computer Science
Friedrich-Alexander University Erlangen-Nürnberg (FAU)

Table of contents

Motivation

Motivation

- Interesting and important parts can not be done without root
- Diverse support of Android and security updates by manufacturers
- Prolonged use of your device

Is it justified that certain apps deny service when a device is modified?

Background

SELinux

- Advanced access control mechanism
- Finely grained, systemwide security policy, which is managed by central authority
- Two modes: permissive and enforcing (default since Android 5)
- Core idea: labeling system that controls permissions based on default denial

Label is a 4-tuple string

Process: user:role:type:sr0

Object: user:object_r:type:sr0

Policy rules can be found in the compiled binary sepolicy

Policy rules

rule domains types:classes permissions;

allow vold cache_file:dir r_dir_perms;

SafetyNet Attestation

- Part of Google Play Services
- Remote device and app attestation
- Either SHA-256 of app or certificate used to sign app

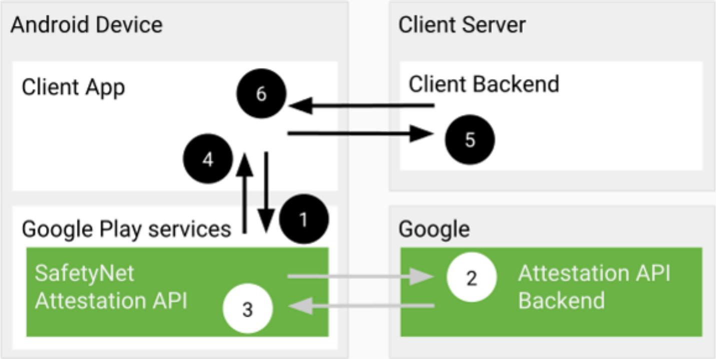

Figure 1: SafetyNet Attestation API protocol [4].

SafetyNet: Attestation results

Device Status	Value of ctsProfileMatch	Value of basicIntegrity
Certified, genuine device that passes CTS	true	true
Certified device with unlocked bootloader	false	true
Genuine but uncertified device	false	true
Device with custom ROM (not rooted)	false	true
Emulator	false	false
No device (protocol emulator script)	false	false
Signs of system integrity compromise (rooting)	false	false
Signs of other active attacks (API hooking)	false	false

Table 1: Possible SafetyNet Attestation results [4].

Verified Boot

- Guarantees integrity of the device software from the bootloader up to the operating system
- Partitions are divided into 4 KiB blocks, which are verified against a signed hash tree when read
- It is not possible anymore to roll back to an earlier OS version
- Bootloader can only be unlocked by a user physically interacting with it

Device state Locked or unlocked bootloader

Boot state Indicates the state of device integrity

Verified Boot: boot states

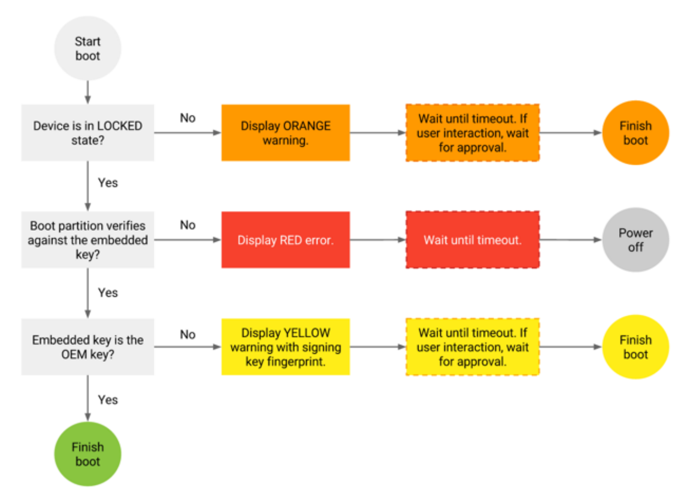

Figure 2: Verified boot flow [2].

Updates

- Diverse and sometimes short update period device manufacturers provide
  –> Popularity of custom ROMs

- New approach by Google:
  -Linux LTS kernels switch from 2 year to 6 year life cicle of support
  -Project Treble since Android 8

–> Separation of Android OS and vendor implementation

–> Faster, easier and less expensive update process

Rooting: soft root vs. hard root

Soft root

- Exploiting security vulnerabilities
- Device vulnerable to malware
- Only option for devices with unlockable bootloader

Hard root

- Su binary is flashed through a custom recovery
- or included in a custom ROM
- Systemless-ly approach: system partition untouched

LineageOS

LineageOS

- Over 1.87 million active installations on 180 different devices
- Patches several security vulnerabilities, which are not be addresses anymore by OEMs
- Device requirements, which must be met for a device to be ready to receive a LineageOS release
- Provides su addon

LineageOS: applications

- Own unique apps not found in AOSP
- Comes without Google apps, but can be flashed
- Apps have to be installed manually by the user
- Alternative app store e.g. F-Droid for FOSS apps

LineageOS: Privacy Guard

- Permission manager of applications
- Seetings can be adjusted fine grained
- Manages root access

LineageOS: su addon

- Separate su addon package
- Su binary in */system/xbin*
- Su can be turned on/off in settings (default: off)

LineageOS: su request

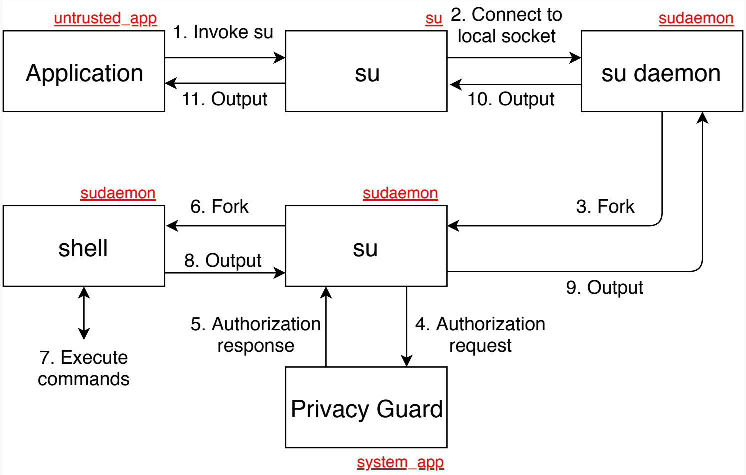

Figure 3: Procedure diagram of invoking su by an application.

LineageOS: security implications

- Unlocked bootloader, no Verified Boot (Recowvery, CVE-2016-5195)
  –> No hardware based root of trust
- Manual installation of apps

Attack possibilities against

- Su and su daemon (CVE-2013-6768, -6769, -6770, -6774, -6775)
- local socket file (0666 permission, /dev/socket/su-daemon/)
- Privacy Guard

- SELinux for su and su daemon in permissive mode

LineageOS: evaluation

Application behaviour

1. LineageOS unmodified
   –> Apps not working (due to missing Google Services)

2. LineageOS with su addon
   –> Apps not working (due to missing Google Services)

3. LineageOS with GApps (OpenGApps, *pico* and *stock* version)
   –> Apps working when installed via Google Play

4. LineageOS with GApps and *su* addon
   –> Apps working when su addon turned off

SafetyNet

```
# disabled root access
SafetyNetResponse: ...
"ctsProfileMatch":false,
"basicIntegrity":true,
"advice":"RESTORE\_TO\_FACTORY\_ROM,LOCK\_BOOTLOADER"

# enabled root access
SafetyNetResponse: ...
"ctsProfileMatch":false,
"basicIntegrity":false,
"advice":"RESTORE\_TO\_FACTORY\_ROM,LOCK\_BOOTLOADER"
```

Magisk

Magisk

- Set of tools, which establish an environment to alter Android systemless-ly
- Accomplished by only patching the boot image
- Can hide modifications from system integrity verifications like Safety
- Provides rooting solution MagiskSU

Magisk: initialization

- Init is replaced with MagiskInit and executed afterwards
- Adds own init.Magisk.rc file to init.rc
- Starts services: Magisk daemon, MagiskHide
- SELinux policy file is patched
- Files reside in /root with symlinks in /sbin (tmpfs)

MagiskSU

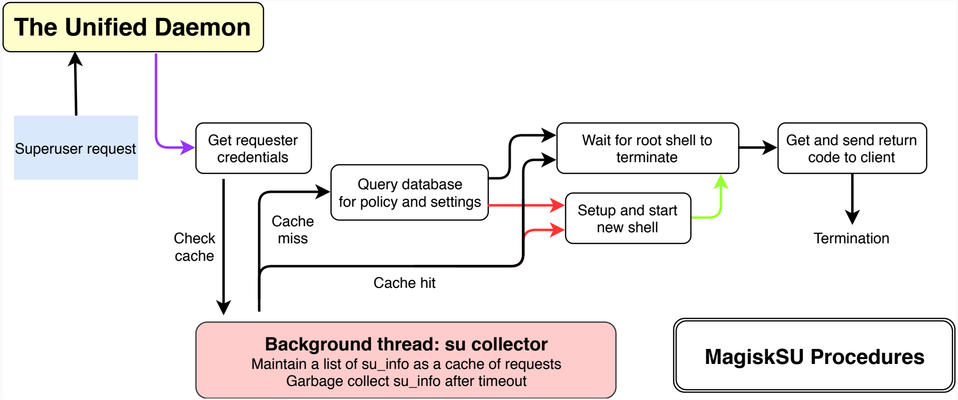

Figure 4: Procedure diagram of invoking su by an application [5].

MagiskSU <–> LineageOS su

They share the same su code base

- Su collector as cache
- Su SELinux label (u:r:su:s0) <–> su and sudaemon label
- Magisk Manager <–> Privacy Guard
- Broadcast request intent <–> AppOpsManager API

Magisk: security implications

- Unlocked bootloader, no Verified Boot (Recowvery, CVE-2016-5195)
  –> No hardware based root of trust

Attack possibilities against

- Su and su daemon (CVE-2013-6768, -6769, -6770, -6774, -6775)
- local socket file (0777 permission, /dev/.socketXXXXX)
- Su request intent
- policy database

- SELinux for su in permissive mode
- New SELinux policies for su

MagiskHide

Hides Magisk, Magisk Manager or an unlocked bootloader from apps or system integrity checks

- Keeps a list of apps to hide from (managed in Magisk Manager)
- Monitors Logcat am_proc_start events
- Target will be paused immediately (SIGSTOP), new process is forked
- Process joins mount namespace and hides sensitive properties
- *tmpfs* /sbin is unmounted
- Target is allowed to continue (SIGCONT)

MagiskHide: sensitive properties

```
ro.boot.verifiedbootstate   = green
ro.boot.flash.locked        = 1
ro.boot.veritymode          = enforcing
ro.boot.warranty_bit        = 0
ro.warranty_bit             = 0
ro.debuggable               = 0
ro.secure                   = 1
ro.build.type               = user
ro.build.tags               = release-keys
ro.build.selinux            = 0
```

Magisk: evaluation

Application behaviour

MagiskHide enabled

- Most applications work
- Some check the installed packages and recognize Magisk Manager
- Solution: Hide Magisk Manger with random package name

MagiskHide disabled

- Most applications do not work
- Error message: device does not have the necessary safety mechanisms or device is rooted

Magisk: evaluation

SafetyNet

```
# MagiskHide disabled
SafetyNetResponse: ...
"ctsProfileMatch":false,
"basicIntegrity":false,
"advice":"RESTORE\_TO\_FACTORY\_ROM"

# MagiskHide enabled
SafetyNetResponse: ...
"ctsProfileMatch":true,
"basicIntegrity":true
```

Root detection

Root detection methods

How detect apps if a device is rooted?

1. Presence of files
2. System properties
3. Directory permissions
4. Installed packages
5. Processes, Services and Tasks
6. Shell commands

Root detection: presence of files

- Check existence of files in certain direcories
  /system/xbin, /system/bin, /system/app, /sbin, /data/app

- Parsing *PATH* and appending */su* to each entry
- Using *which* combined with *su*

Root detection: system properties

Check certain entries in /system/build.prop using *getprop*

Queried entries

- *ro.build.tags =release-keys*
- *ro.build.type = user*
- *ro.debuggable = 0*
- *ro.secure = 1*

Root detection: directory permissions

Check directory permissions using common functions and the Java API

Used functions

- *access(3P)*
- *canRead()*
- *canWrite()*

Root detection: installed packages

Use PackageManger API to retrieve installed packages

Used functions

- *getInstalledPackages()*
- *getInstalledApplications()*
- *pm list packages*

Root detection: processes, services and tasks

Use ActivityManager API to retrieve information about proccesses, services and tasks

Used functions

- *get.RunningAppProcesses()*
- *get.Running.Services()*
- *get.RecentTasks()*

Root detection: shell commands

Use common shell commands to retrieve information on files and folders

Used functions

- *ls*
- *ps | grep <name>*
- *pm path <packagename>*

Jailbreaking iOS

Jailbreaking iOS

- Closed source operating system with software restrictions
- Jailbreaking <-> exploiting security vulnerabilities
- Not available for every iOS versions
- Best compared to soft rooting Android

Not recommended for devices handling sensitive information since attackers can use vulneratiblities, too!

Conclusion

Summary

- There are opportunities for attacks, but no current known attacks
  –> no less secure than other software
- no Verified Boot
- Root detection: No distinction hard <-> soft root
  –> No justification for excluding such devices from certain apps.
- Security relies on user and his decisions
- Custom ROMs often the only possibility to get updates
- Cat and mouse game between Google and the rooting community

Questions?

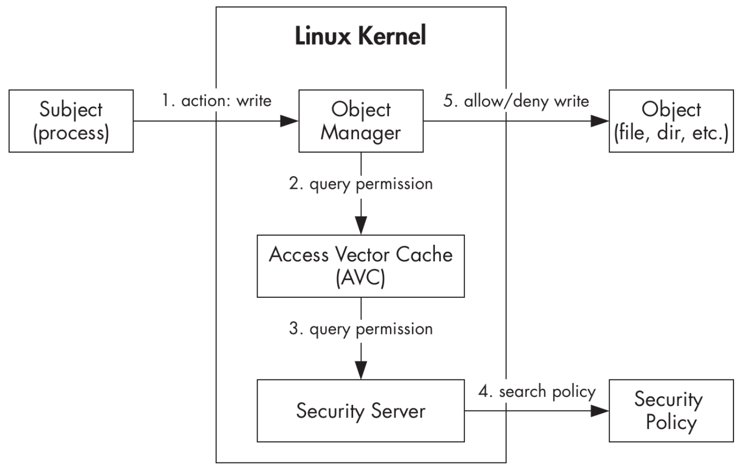

Figure 5: SELinux components [1].

SafetyNet: Attestation payload

```
"nonce": "R2Rra24fVm5xa2Mg",
"timestampMs": 9860437986543,
"apkPackageName": "com.package.name.of.requesting.app",
"apkCertificateDigestSha256": ["base64 encoded, SHA-256 hash of the
                                certificate used to sign requesting app"],
"apkDigestSha256": ["base64 encoded, SHA-256 hash of
                    the APK installed on a user's device"],
"ctsProfileMatch": true,
"basicIntegrity": true,
```

SafetyNet: additional APIs

Safe Browsing API determines if an URL has been marked as a known threat

reCAPTCHA API uses reCAPTCHA to protect apps from malicious traffic/spam

Verify Apps API protects devices against potentially harmful apps

Verified Boot: hash tree

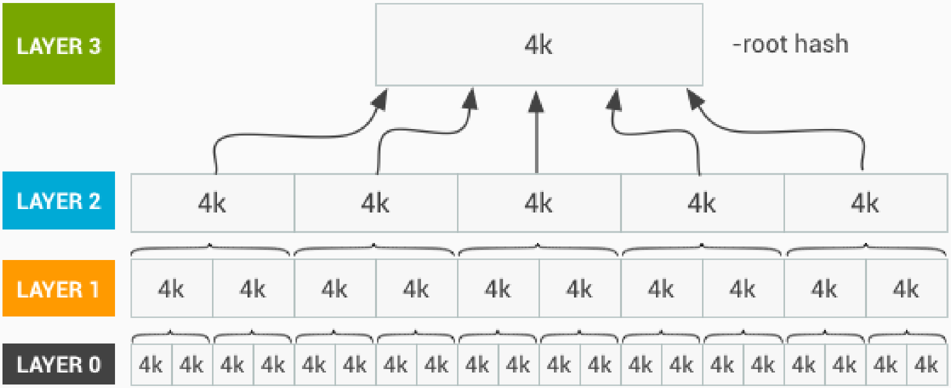

Figure 6: dm-verity hash tree [2].

Updates: Project Treble

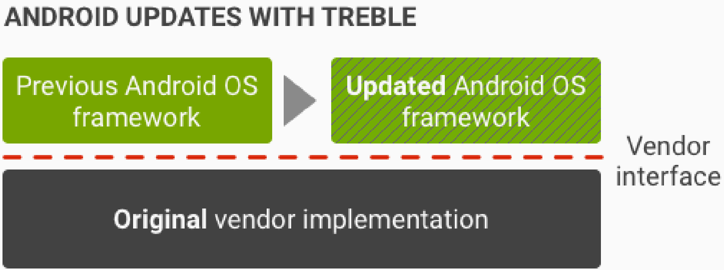

Figure 7: The update process before Project Treble [3].

LineageOS: shell escape vulnerability

```
"su -c 'COMMAND' "

...
ctx->from.uid, ctx->to.uid, get_command(&ctx->to),
policy == ALLOW ? "allow" : "deny", ctx->user.android_user_id);


get_command() would return "COMMAND", unescaped

su -c "'&touch /data/test;'"
su -c '`touch /data/test`'
su -c '$(touch /data/test)'
```

MagiskHide: procedure diagram

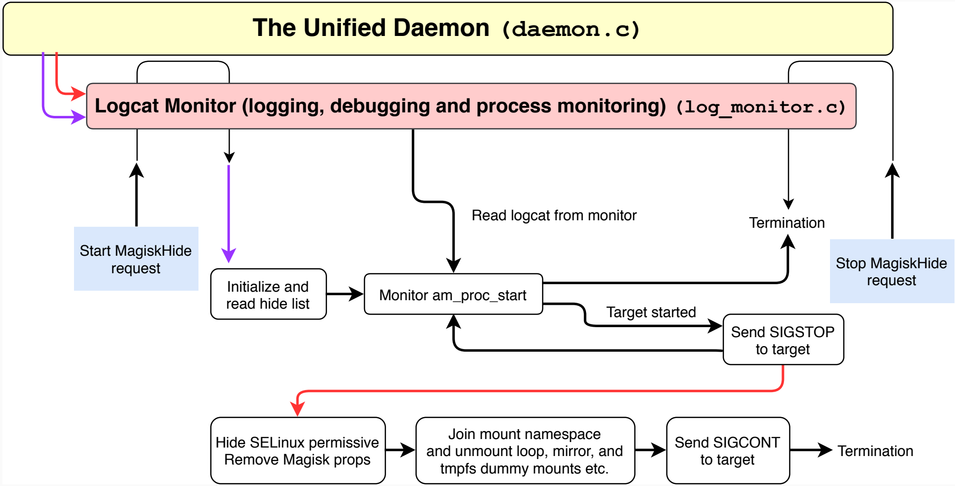

Figure 8: Procedure diagram of MagiskHide [5].

Magisk: Magisk tools

```
magiskboot                     /* binary */
magiskinit                     /* binary */
magiskpolicy -> magiskinit
supolicy -> magiskinit         /* alias of magiskpolicy */
magisk                         /* binary */
magiskhide -> magisk
resetprop -> magisk
su -> magisk
```

Magisk: structure I

LOGFILE	"/cache/magisk.log"
DISABLEFILE	"/cache/.disable_magisk"
UNINSTALLER	"/cache/magisk_uninstaller.sh"
CACHEMOUNT	"/cache/magisk_mount"
MAINIMG	"/data/adb/magisk.img"
DATABIN	"/data/adb/magisk"
MANAGERAPK	"/data/adb/magisk/magisk.apk"
DEBUG_LOG	"/data/adb/magisk_debug.log"
UNBLOCKFILE	"/dev/.magisk.unblock"
PATCHDONE	"/dev/.magisk.patch.done"
MAGISKRC	"/init.magisk.rc"
MAGISKTMP	"/sbin/.core"
MIRRDIR	"/sbin/.core/mirror"

Magisk: structure II

BBPATH	"/sbin/.core/busybox"
MOUNTPOINT	"/sbin/.core/img"
COREDIR	"/sbin/.core/img/.core"
HOSTSFILE	"/sbin/.core/img/.core/hosts"
HIDELIST	"/sbin/.core/img/.core/hidelist"

Magisk: resetprop

- created by pulling out the portion of source code managing properties from AOSP
- try to mimic what init is doing.
- Result: direct access to the data structure
- Property deletion is accomplished by detaching the target node from the tree structure, making it effectively invisible.

Analyzed applications

Banking

- Sparkasse, Sparkasse pushTAN
- VR-Banking, VR-SecureGo
- DKB-Banking, DKB-TAN2go
- Deutsche Bank Mobile
- ING-DiBa Banking to go, ING-DiBa Banking + Brokerage
- o2 Banking, Commerzbank Banking, N26

Antivirus, Root checker

- Avira, Kaspersky, Avast, McAfee, Eset Mobile Security, AVG Mobile
- Root Checker (3x), Root Check

References i

N. Elenkov.
Android Security Internals - An In-Depth Guide to Android's Security Architecture.
no starch press, 2014.

Google.
Android security: Verified boot.

Google.
Project treble.

Google.
Safetynet attestation api.

J. Wu.
Magisk documentation.