



DroidGuard

A Deep Dive into SafetyNet

Romain Thomas

June, 2022

SSTIC 2022

Introduction

- Security engineer at **UL - La Ciotat**¹
- Working on banking app certifications (EMVCo, VISA, ...)
- Author of LIEF: <https://lief.re>
- Enjoy Android, reverse engineering and, obfuscation.



¹<http://tiny.cc/pb5quz>

- SafetyNet is a solution developed by Google to verify device's **integrity**.

- SafetyNet is a solution developed by Google to verify device's **integrity**.
- **integrity?**

- SafetyNet is a solution developed by Google to verify device's **integrity**.
- **integrity?**
 - Rooted
 - Custom firmware
 - Emulators
 - Bootloader unlocked
 - ...

- SafetyNet is a solution developed by Google to verify device's **integrity**.
- **integrity?**
 - Rooted
 - Custom firmware
 - Emulators
 - Bootloader unlocked
 - ...
- It is used by a large number of app developers who need be sure their applications do not run on a *compromised* environment (games, fintech, messaging apps, ...)

SafetyNet API

```
1 SafetyNet.getClient(this).attest(nonce, API_KEY)
2 .addOnSuccessListener(this) {
3     // Indicates communication with the service was successful.
4     // Use response.getJwtResult() to get the result data.
5 }
6 .addOnFailureListener(this) { e →
7     // An error occurred while communicating with the service.
8     if (e is ApiException) {
9         // An error with the Google Play services API contains some
10        // additional details.
11        val apiException = e as ApiException
12
13        // You can retrieve the status code using the
14        // apiException.statusCode property.
15    } else {
16        // A different, unknown type of error occurred.
17        Log.d(FragmentActivity.TAG, "Error: " + e.message)
18    }
19 }
20 }
```

→ The developer provides:

1. A **nonce** to avoid replay attack
2. An **API_KEY** to be authenticated by the Google's backend

← SafetyNet returns:

1. A JWS token² that wraps the device's integrity status
2. Or, an error

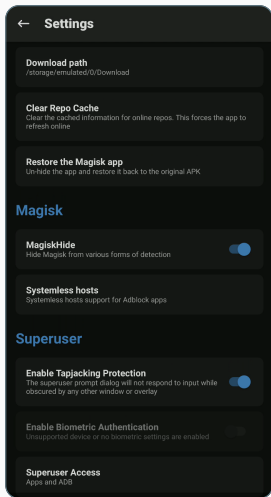
²Signed by Google's private key (in the backend)

Why this talk?

Inside Android's SafetyNet Attestation

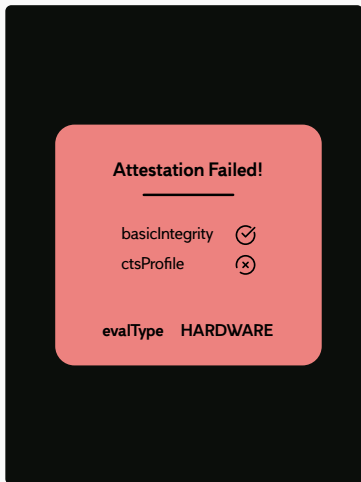
Collin Mulliner & John Kozyrakis

Magisk & Magisk Hide



Magisk & Magisk Hide

Magisk & Magisk Hide



basicIntegrity vs ctsProfileMatch

Basic Integrity

"A more lenient verdict of device integrity. If only the value of `basicIntegrity` is `true`, then the device running your app likely wasn't tampered with. However, the device hasn't necessarily passed Android compatibility testing."

CTS Profile Match

"A stricter verdict of device integrity. If the value of `ctsProfileMatch` is `true`, then the profile of the device running your app matches the profile of a device that has passed Android compatibility testing and has been approved as a Google-certified Android device."

<https://developer.android.com/training/safetynet/attestation>

basicIntegrity vs ctsProfileMatch

Basic Integrity

- Rooted device
- Emulator
- API Hooking

CTS Profile Match

- Rooted device
- Emulator
- API Hooking
- + Bootloader unlocked
- + Device with custom ROM (not rooted)
- + **Genuine but uncertified device, such as when the manufacturer doesn't apply for certification**

Magisk v24.0

MagiskHide Removal

I have lost interest in fighting this battle for quite a while;
plus, the existing MagiskHide implementation is flawed in so many ways.

Decoupling Magisk from root hiding is, in my opinion, beneficial to the community.
Ever since my announcement on Twitter months ago, highly effective "root hiding" modules
(much MUCH better than MagiskHide) has been flourishing, which again shows that people are
way more capable than I am on this subject.

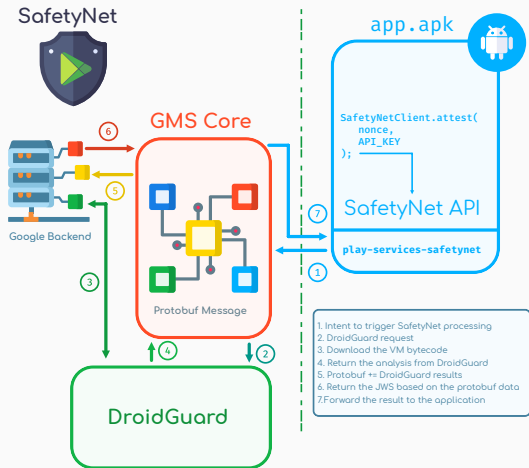
So why not give those determined their time to shine, and let me focus on improving Magisk
instead of drowning in the everlasting cat-and-mouse game

Magisk v24.0 Release Note – January 2022

```
1 SafetyNet.getClient(this).attest(nonce, API_KEY)
2 .addOnSuccessListener(this) {
3     // Indicates communication with the service was successful.
4     // Use response.getJwsResult() to get the result data.
5 }
6 .addOnFailureListener(this) { e →
7     // An error occurred while communicating with the service.
8     if (e is ApiException) {
9         // An error with the Google Play services API contains some
10        // additional details.
11        val apiException = e as ApiException
12
13        // You can retrieve the status code using the
14        // apiException.statusCode property.
15    } else {
16        // A different, unknown type of error occurred.
17        Log.d(FragmentActivity.TAG, "Error: " + e.message)
18    }
19 }
20 }
```

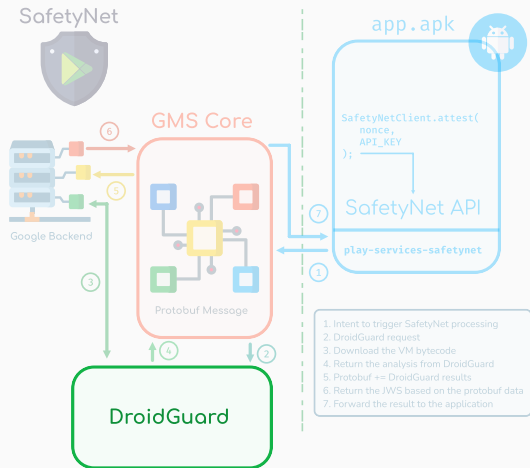

SafetyNet

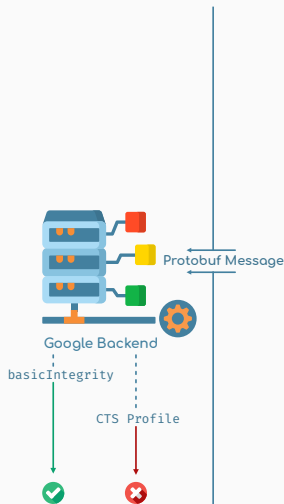
```
1 SafetyNet.getClient(this).attest(nonce, API_KEY)
2 .addOnSuccessListener(this) {
3     // Indicates communication with the service was successful.
4     // Use response.getJwsResult() to get the result data.
5 }
6 .addOnFailureListener(this) { e →
7     // An error occurred while communicating with the service.
8     if (e is ApiException) {
9         // An error with the Google Play services API contains some
10        // additional details.
11        val apiException = e as ApiException
12
13        // You can retrieve the status code using the
14        // apiException.statusCode property.
15    } else {
16        // A different, unknown type of error occurred.
17        Log.d(FragmentActivity.TAG, "Error: " + e.message)
18    }
19 }
20 }
```



SafetyNet

```
1 SafetyNet.getClient(this).attest(nonce, API_KEY)
2 .addOnSuccessListener(this) {
3     // Indicates communication with the service was successful.
4     // Use response.getJwsResult() to get the result data.
5 }
6 .addOnFailureListener(this) { e →
7     // An error occurred while communicating with the service.
8     if (e is ApiException) {
9         // An error with the Google Play services API contains some
10        // additional details.
11        val apiException = e as ApiException
12
13        // You can retrieve the status code using the
14        // apiException.statusCode property.
15    } else {
16        // A different, unknown type of error occurred.
17        Log.d(FragmentActivity.TAG, "Error: " + e.message)
18    }
19 }
20 }
```





`com.google.android.gms`

```
SafetyNetData = {  
  nonce           = [ca ee ...]  
  packageName     = "com.demo.snet"  
  signatureDigest = [66 49 ...]  
  fileDigest      = [fa 0a ...]  
  gmsVersionCode  = 213918046  
  suCandidates = {  
    fileName = "/system/bin/su"  
    digest   = [25 53 ...]  
  }  
  selinuxState = {  
    supported = true  
    enabled   = true  
  }  
  currentTimeMs = 1638672572674  
  googleCn      = false  
}
```

Code written in Java/Kotlin, lightly
obfuscated.

Code mostly written in C++
and obfuscated (VM, MBA, ...)

`/data/app/[...]/com.google.android.gms/base.apk`

`com.google.android.gms.unstable`

`DroidGuardResult = "CgZpApMYiWYSi9cB [...]"`

`/data/data/com.google.android.gms/app_dg_cache/<hash>/the.apk`

Why This Talk?

Goal of this talk

- Understand how SafetyNet works thanks to DroidGuard
- Describe the integrity's checks behind SafetyNet

Why This Talk?

Goal of this talk

- Understand how SafetyNet works thanks to DroidGuard
- Describe the integrity's checks behind SafetyNet

Non-goal of this talk

- Show methods to bypass or tricks the hardware attestation
- Promote/release a new click and play tool to replace MagiskHide

DroidGuard: The VM behind SafetyNet

```
DroidGuardResult = "CgZpApMYiWYSi9cB [..]"
```

1. How this token is generated?
2. What kind of information is stored?

How this token is generated?

`com.google.ccc.abuse.droidguard.DroidGuard`



`/data/data/com.google.android.gms/app_dg_cache/<hash>/the.apk`

1. APK updated every ~ 2 weeks from the Google's servers³
2. The Java layer is pretty small: about ~ 60 classes.
3. Embed a native library that implements an **obfuscated VM**

³not from the PlayStore

How this token is generated?

```
loadDroidGuardLibrary();  
DroidGuard(Context context,  
            String flowName,  
            byte[] program);
```

DroidGuard



Java Native Interface



Obfuscation:

- VM based
- MBA
- Anti-hooking
- Anti-debug
- Buffers encoding



Rooted?

Emulator?

Bootloader
Unlocked?

Infected with
Pegasus?

VM Bytecode dynamically downloaded
for each SafetyNet
request



DroidGuardResult = "CgZpApMYiWYSi9cB [...]"

To highlight the logic behind SafetyNet, we have to understand how the bytecode behaves within the VM and how the VM is designed.

DroidGuard VM: Registers



- 256 **typed** registers

0. **Pointer**
1. **Double**
2. **jobject** (JNI object)
3. **Int**
4. **Long**
5. **String/Buffer**
6. **None**

DroidGuard VM: Registers

- 256 **typed** registers, shuffled for each new version of the VM



0. **String/Buffer**
1. **Int**
2. **Long**
3. **Double**
4. **jobject (JNI object)**
5. **Pointer**
6. **None**

VM: How to Write a Register Value?



DroidGuard VM: The Handlers



libd23DDF14B425.so

The DroidGuard VM is composed of a set of *handlers* that have a dedicated purpose:

- Perform a syscall
- Resolve a function (`dlsym`)
- Perform an add, xor, mult, div, ...
- Read an encoded buffer
- Perform a SHA256⁴
- Call a JNI function
- ...

⁴Based on BoringSSL

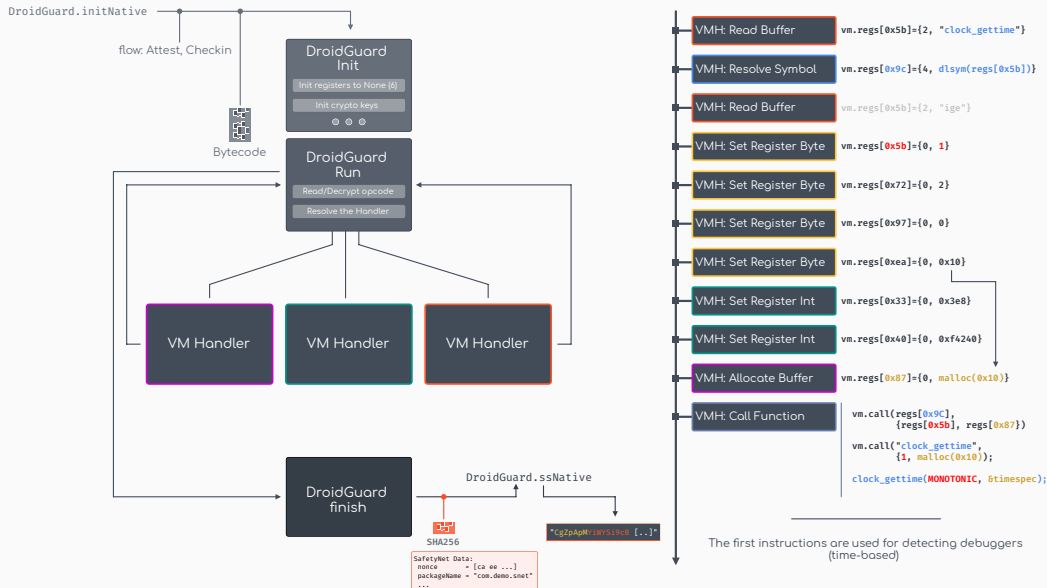
VM Handlers



libd23DDF14B425.so

```
.text:0000000000249B8 .text:0000000000249B8 vm_init_handlers
.text:0000000000249B8 .text:0000000000249B8 var_10= -0x10
.text:0000000000249B8 .text:0000000000249B8 STP X30, X19, [SP, #var_10]!
.text:0000000000249BC MOV X19, X0
.text:0000000000249C0 ADD X0, X0, #0x20 ; ' ' ; s
.text:0000000000249C4 MOV W2, #0x1000 ; n
.text:0000000000249C8 MOV W1, WZR ; c
.text:0000000000249CC BL .memset
.text:0000000000249D0 ADRL X17, VMH_01c2b0
.text:0000000000249D8 STR X17, [X19, #0xAA0]
.text:0000000000249DC ADRL X17, VMH_02d568
.text:0000000000249E4 STR X17, [X19, #0x820]
.text:0000000000249E8 ADRL X17, VMH_030288
.text:0000000000249F0 STR X17, [X19, #0x480]
.text:0000000000249F4 ADRL X17, VMH_321A8
.text:0000000000249FC STR X17, [X19, #0x690]
.text:000000000024A00 ADRL X17, VMH_JNI_CallStaticObjectMethod
.text:000000000024A08 STR X17, [X19, #0xDA0]
.text:000000000024A0C ADRL X17, VMH_0378d8
.text:000000000024A14 STR X17, [X19, #0xFB0]
.text:000000000024A18 ADRL X17, VMH_03951c
.text:000000000024A20 ADRP X0, #VMH_0297a0@PAGE
.text:000000000024A24 STR X17, [X19, #0x270]
.text:000000000024A28 ADRP X17, #VMH_03a2f0@PAGE
.text:000000000024A2C ADD X0, X0, #VMH_0297a0@PAGEOFF
.text:000000000024A30 ADD X17, X17, #VMH_03a2f0@PAGEOFF
.text:000000000024A34 STR X0, [X19, #0xF20]
.text:000000000024A38 ADRP X0, #VMH_02d754@PAGE
.text:000000000024A3C STR X17, [X19, #0x290]
.text:000000000024A40 ADRP X17, #VMH_set_uint32@PAGE
.text:000000000024A44 ADD X0, X0, #VMH_02d754@PAGEOFF
.text:000000000024A48 ADD X17, X17, #VMH_set_uint32@PAGEOFF
.text:000000000024A4C STR X0, [X19, #0x5E0]
.text:000000000024A50 ADRP X0, #VMH_xor@PAGE
```

VM Overview



*"What brings you to these parts of town?
Say hi to droidguard-hello+xxxxxxxxxxxxxxxxxx@google.com"*

*"You just keep pulling back the layers!
Say hi to droidguard-hello+xxxxxxxxxxxxxxxxxx@google.com"*

Device's Integrity Checks

With a good understanding of the VM and its handlers, we can target a few of them⁵ to highlight the integrity checks.

⁵Mostly handlers which perform syscalls, calls, JNI calls

Example

```
[016667] VMH_read_buffer() {
0x039748: vm→decode(sp!968, 0x4e2f, 0x2): 0x4e31
0x0397b8: std::string(sp!968, 0x10, 0x0)
0x0397fc: vm→decode(sp!967, 0x4e31, 0x10): 0x4e41
0x0398a4: vm→decode(sp!94c, 0x4e41, 0x1): 0x4e42
0x039910: operator_new(0x18): 0x6f08b44be0
0x03991c: std::string::copy(malloc@0x6f08b44be0, sp!968)
0x03992c: vm_set_register(0x4c, 0x2, 0x6f08b44be0): "/data/local/xbin"
}
// ...
[016674] VMH_call_function() {
0x03c75c: vm→read_byte_vector(std::vector<uint8_t>@sp!980, KEY): {0x09, 0x4c, 0x3d, 0x09}
0x03c780: vm→decode(sp!950, 0x4872, 0x1): 0x4873
0x03c7e0: vm→decode(sp!950, 0x4873, 0x1): 0x4874
0x03c824: vm→get_pointer(0xa): 0x6e6de1459c
0x03c834: vm→read_register(0x7b): 0x30
0x03c844: operator_new(0x10): 0x6ef8a9c470
0x03c87c: memcpy(0x6ef8a9c470, 0x6e84242788, 0x10)
0x03c8a0: vm→prepare_params(in: {09,4c,3d,09},
                        {&vm_syscall, 0x30, 0x0, 0x6f48d4fb61, 0x4, 0x0},
                        {"/data/local/xbin"}')
{
  faccessat("/data/local/xbin"): 0xfffffffffffffffe
}
0x03c930: vm→set_register(0xc9, 0x0, 0xfffffffffe)
}
```

- `/data/local/tmp/su`
- `/system/bin/.ext/su`
- `/system/bin/su`

Root Detection: System Properties

- `init.svc.magisk_service`
- `persist.magisk.hide`
- `ro.magisk.disable`

"Signs of active attacks, such as API hooking"

```
0x2171c VMH_read_buffer() {
  0x039748: vm→decode(sp!668, 0x3ad0, 0x2): 0x3ad2
  0x0397b8: std::string(sp!668, 0x15, 0x0)
  0x0397fc: vm→decode(sp!667, 0x3ad2, 0x15): 0x3ae7
  0x0398a4: vm→decode(sp!64c, 0x3ae7, 0x1): 0x3ae8
  0x039910: operator_new(0x18): 0x7c21437560
  0x03991c: basic_string_copy(malloc@7c21437560, sp!668): "'6V+0F`C"
  0x03992c: vm→set_register(0x6f, 0x2, 0x7c21437560): ":libriru_edxposed.so:"
}
0x2171c VMH_find_in_string() {
  0x043140: vm→decode(sp!660, 0x3ae9, 0x1): 0x3aea
  0x0431b8: vm→decode(sp!660, 0x3aea, 0x1): 0x3aeb
  0x043214: vm→decode(sp!660, 0x3aeb, 0x1): 0x3aec
  0x043270: vm→decode(sp!660, 0x3aec, 0x1): 0x3aed
  0x043304: vm→read_register(0x9): 0x0
  regs[0xac].value → 0xc8e697a25b30b3fb | ":linker64:app_process64:[vdso]:libandroid_runtime.so:libbinder.so: [ ... ]
  regs[0x6f].value → 0x239a5f05c8953bcf | ":libriru_edxposed.so:"
  // ...
}
```


"Signs of active attacks, such as API hooking"

- `frida-agent-64.so`
- `libriru_snet-tweak-riru.so`
- `libsandhook.so`

Last but not least ...

NSO's Pegasus

```
+-----+
cd 2e 5f 64 61 65 6d 6f 6e 73 75 5f cd 70 65 67
61 73 75 73 2e 61 70 6b cd 63 70 cd 63 73 6b cd
68 74 66 73 6b cd 2e 6c 73 cd 2e 6c 64 2e 6a 73
cd 69 73 75 cd 61 6e 64 72 6f 56 4d 2d 70 72 6f
70 cd 62 75 73 79 62 6f 78 cd 6d 75 cd 64 61 65
6d 6f 6e 73 75 cd 2e 63 6f 6c 64 62 6f 6f 74 5f
69 6e 69 74 cd 73 75 5f cd 2e 63 70 2e 70 6d cd
74 65 6d 70 5f 73 75 cd 69 6e 69 74 2e 6d 61 67
69 73 6b 2e 72 63 cd 62 61 73 65 72 76 69 63 65
cd 62 61 64 61 6d 6f 6e cd 2e 70 65 cd 70 70 6d
cd 2e 5f 73 75 cd 2e 5f 73 75 5f cd 64 72 6f 69
64 34 78 2d 70 72 6f 70 cd 74 74 56 4d 2d 70 72
6f 70 cd 69 67 70 69 cd 71 65 6d 75 5f 70 72 6f
70 73 cd 2e 70 72 2e 69 6f cd 2e 74 65 2e 73 74
cd 61 6d 70 6d 6c cd 69 70 6d cd 2e 74 73 cd 61
6e 6c 5f 36 34 cd 61 6e 6c cd 67 69 65 66 72 6f
6f 74 cd 72 62 6e cd 6d 69 63 72 6f 76 69 72 74
2d 70 72 6f 70 cd 73 6d 73 64 61 6d 6f 6e cd 77
61 77 cd 73 6d 73 73 65 72 76 69 63 65 cd 6c 69
62 69 6d 63 72 63 5f 36 34 2e 73 6f cd 77 6c 61
6e 64 cd 6d 69 63 72 6f 76 69 72 74 64 cd 6c 69
62 69 6e 6a 65 63 74 6f 72 2e 73 6f cd 6e 6f 78
2d 70 72 6f 70 cd 73 75 cd 73 75 32 cd 61 6d 70
6d 6c 5f 36 34 cd 2e 61 75 74 68 6f 72 cd
+-----+
.._daemonsu..peg
asus.apk.cp.csk.
htfsk..ls..ld.js
.isu.androVM-pro
p.busybox.mu.dae
monsu..coldboot_
init.su..cp.pm.
temp_su.init.mag
isk.rc.baservice
.badamon..pe.ppm
..su..su_.droi
d4x-prop.ttVM-pr
op.igpi.qemu_pro
ps..pr.io..test
.ampml.ipm..ts.a
nl_64.anl.giefro
ot.rbn.microvirt
-prop.smsdamon.w
aw.smsservice.li
bimcrc_64.so.wla
nd.microvirt.d.li
binjector.so.nox
-prop.su.su2.amp
ml_64..author.
```

.coldboot_init

csk

pegasus.apk

Pegasus for Android Technical Analysis and Findings of Chrysaor 2017

Second Pegasus Sample

This sample differs significantly from the first sample analyzed above. It has a considerably smaller code base and is clearly intended to be installed on a device that was previously rooted and already contains the `/system/csk` superuser binary.

Analysis of this sample showed that its sole purpose is to initiate a connection to a remote address, download an additional payload, save this data to the file `/data/data/com.network.android/.coldboot_init`, before copying it to `/mnt/obb/.coldboot_init` and changing the permissions on this file to 0711. The functionality to perform this download is located in the sample's only native binary, `libsgn.so`. The portion of the sample written in java is extremely minimal and exists just to load `libsgn.so`. Below is a section of code from the `libsgn.so` file that attempts to write the retrieved payload to various paths.



Telemetry

In addition to pre-defined *boolean* checks⁶. DroidGuard collects information about the device (system properties, mount information, ...).

These information are used by the Google backend to enhance the device's integrity checks.

⁶Whether a file exists, if a library is present in memory, ...

```
ro_zygote      = "zygote64_32"
pointer_info   = "7f3669240000-7f3669241000 rw-p 00000000"
cmdline        = "com.google.android.gms.unstable"
env_path       = "/product/bin:/apex/com.android.runtime/bin:/apex/com.android.art/bin:[ ... ]"
cache_dir      = "/data/user/0/com.google.android.gms/cache"

vbmeta_device_state = "locked"
vbmeta_digest       = "5c43a03e2a47d742deefb3a05c2bcd1afadedb89ddbdba7651f99fdc92438f8"
verifiedbootstate   = "green"
security_patch       = "2021-12-12"
f134                 = "com.google.android.gms" # Output of com.google.android.gms.droidguard.loader.RuntimeApi.c()
kernel_info          = "5.4.223-ga45ffa6db-74ceeb #1 SMP PREEMPT Tue Jul 21 01:52:07 UTC 2021"
flow                 = "attest"
installer            = "com.android.vending"
proc_self_stat       = "561 (id.gms.unstable) S 949 949 0 0 -1 107832 324 0 0 0 "
```

```
f242 = [  
    # List of KeyStore.getCertificateChain (Hardware attestation → CTS Profile)  
]  
mount_info = [  
    "/dev/block/loop22 /apex/com.android.art@1"          "/dev/block/loop22 /apex/com.android.art",  
    "/dev/block/loop23 /apex/com.android.i18n@1"         "/dev/block/loop23 /apex/com.android.i18n",  
    "/dev/block/loop27 /apex/com.android.vndk.v30@1"     "/dev/block/loop27 /apex/com.android.vndk.v30"  
]  
proc_self_maps_info = [  
    "/apex/com.android.art/javalib/bouncycastle.jar",  
    "/system/framework/boot-ims-common.vdex",  
    "/data/data/com.google.android.gms/app_dg_cache/1FEFB755F7DFAAFB69E71C4B872D96A200EC65BF/the.apk"  
    ...  
]  
current_class_loaders = ""  
dalvik.system.PathClassLoader[  
    DexPathList[  
        [zip file "/data/app/~*****=/com.google.android.gms-*****-*****-*A=/base.apk"]  
        nativeLibraryDirectories=[/system/lib64, /system/product/lib64]  
    ]  
]  
""
```

Telemetry & Device's Integrity Checks

DroidGuard



the.apk

```
libriru_snet-tweak-riru.so
frida-agent-64.so      proc_self_maps_info = [
libva-native.so        "/apex/com.android.art/javalib/bouncycastle.jar",
libriru_edxp.so         "/system/framework/boot-ims-common.vdex",
                        ...
                        ]
/data/local/tmp/su
/vendor/bin/su

init.svc.magisk_service
persist.magisk.hide
ro.magisk.disable

vbmeta_device_state = "locked"
vbmeta_digest       = "5c43a03e2a47d742deefb3a05 [...]"
verifiedbootstate   = "green"
security_patch       = "2021-12-12"
f134                 = "com.google.android.gms"
kernel_info          = "5.4.223-ga45ffa6db-74ceeb [...]"
flow                 = "attest"
installer             = "com.android.vending"
proc_self_stat       = "561 (id.gms.unstable) S [...]"

Pegasus

True False True Protobuf Encoded
```

```
DroidGuardResult = "CgZpApMYiWYSi9cB [...]"
```


Conclusion

Conclusion

After investigation, it seems that DroidGuard is not only used to run Google's bytecode related to SafetyNet.

Conclusion

After investigation, it seems that DroidGuard is not only used to run Google's bytecode related to SafetyNet.

It can also run *programs* which are named:

attest/full : SafetyNet checks (~ 70 KiB)

msa-f : ??? (~ 7 KiB)

checkin : for Google account enrollment? (~ 50KiB)

ad_attest : to prevent ad-frauds? (~ 50KiB)

federatedMachineLearningReduced : ??? (~ 50 KiB)

po-token-fast,hades_persephone_risk,smartsetup_2,dcs_get_verdict ...

What is the cost of such *reverse engineering*?

What is the cost of such *reverse engineering*?

1. The reverse engineering of DroidGuard is not trivial and requires tooling:

- Code lifting/emulation with QBDL and Unicorn
- Dynamic analysis with Frida Gum⁷
- Static code analysis with IDA
- MBA simplifications with msynth on the top of Miasm
- Dedicated tools to inspect the VM:
 - Dump the VM's registers
 - Decode the encoded buffers
 - ...

⁷Combined with LIEF for the runtime integrity bypass

What is the cost of such *reverse engineering*?

2. Regular updates which occur ~ 2 weeks requires to automate the process.⁸

- To have a good overview of the design: ~ 5 weeks
- To create dedicated tools: ~ 2 weeks
- In the end, a new version of the VM could be reversed⁹ in a couple of hours

⁸Or you give-up

⁹Identifying the VM handlers, the mapping of the registers types, the encodings, ...

What is the cost of such *reverse engineering*?

3. Conclusion:

- Well protected and difficult to circumvent
- The **basicIntegrity** flag can – in the end – be bypassed without Magisk Hide¹⁰

¹⁰PoC: <https://www.romainthomas.fr/projects-images/safetynet/>

What are the limits of the SafetyNet's design?

What are the limits of the SafetyNet's design?

The VM runs in a dedicated process¹¹ and the checks are done in this memory space.

⇒ They **cannot** detect local tampering in the application that performed the SafetyNet request.

¹¹com.google.android.gms.unstable

What are the limits of the SafetyNet's design?

This is why MagiskHide only had to target `com.google.android.gms.unstable`¹² to bypass SafetyNet.

¹²and, to a lesser extent `com.google.android.gms`

Hardware Attestation



```
KeyStore ks = KeyStore.getInstance("AndroidKeyStore");  
ks.load(null);  
ks.aliases(); // Iterate and check the aliases
```

```
long rndLong = (new Random()).nextLong();  
String alias = "unstable.<hash>." + rndLong.toString();
```


```
KeyGenParameterSpec spec = new KeyGenParameterSpec.Builder(alias, KeyProperties.PURPOSE_SIGN)  
    .setAlgorithmParameterSpec(new ECGenParameterSpec("secp256r1"))  
    .setDigests(KeyProperties.DIGEST_SHA512)  
    .setAttestationChallenge(<unique number>)  
    .build();
```


```
KeyGenerator keyGenerator = KeyPairGenerator.getInstance("EC", "AndroidKeyStore");  
keyGenerator.initialize(spec);  
keyGenerator.generateKeyPair();
```

```
Certificate certificates[] = keyStore.getCertificateChain(alias);
```


Thank you for your attention


Thank you for your attention

 <https://github.com/romainthomas/droidguard-samples>

 <https://www.romainthomas.fr/publication/>

Thank you for your attention

 <https://github.com/romainthomas/droidguard-samples>

 <https://www.romainthomas.fr/publication/>

Questions?