

Android_Emuroot: Outil de *rooting* d'émulateurs Android Google API Playstore

ou comment faciliter l'analyse dynamique d'applications Android protégées

Mouad Abouhali, Anaïs Gantet

SSTIC, juin 2020

AIRBUS

L'évaluation de la sécurité des applications Android requiert un environnement de travail parfois complexe à mettre en place

- Téléphones mobiles *rootés*
 - Différentes marques et plusieurs versions d'Android
- SDK et NDK Android
- Émulateurs Android
 - Émulateurs "*Google Default*" et "*Google API*"
 - Émulateur "*Google API Playstore*"
- Outils de rooting
- Outils d'analyse
 - Frida, Drozer, Objection, etc.

Introduction

Fonctionnement interne

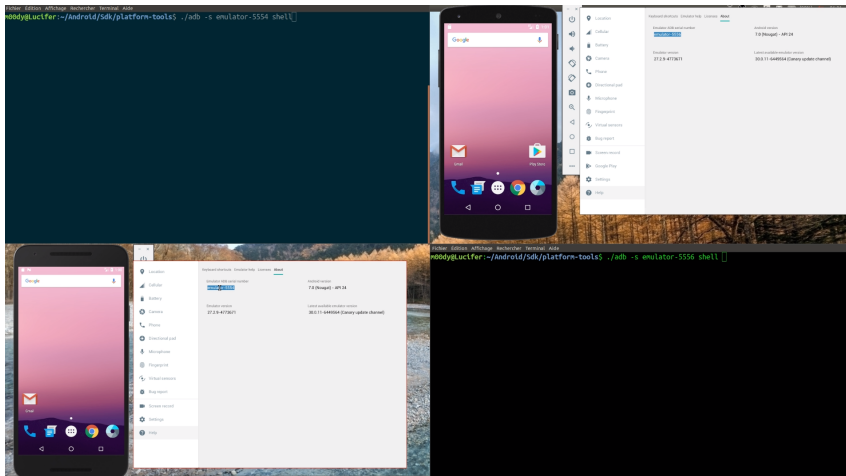
Utilisation et utilité

Conclusion

Introduction

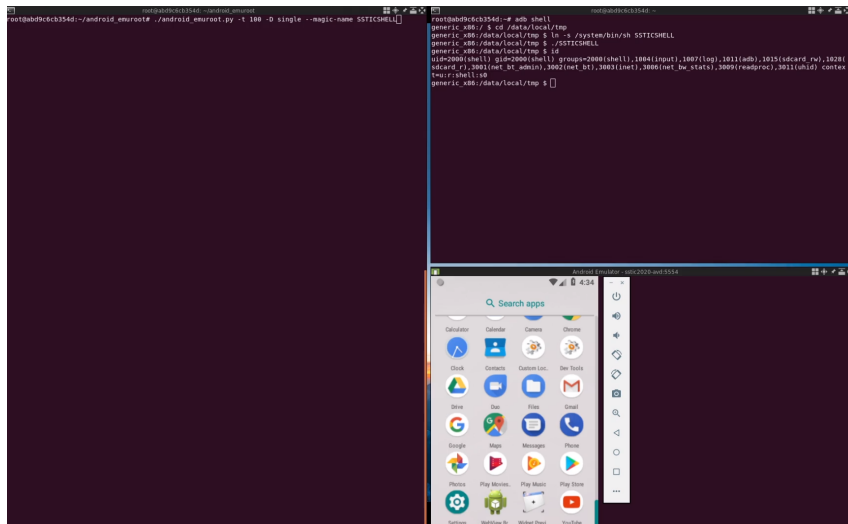


Les émulateurs Android



Besoin d'un shell sur un émulateur Google API Playstore

- Avec des droits privilégiés (*root*)
- Qui permette d'utiliser les outils d'analyse dynamique classiques (Frida, etc.)
- Qui soit difficilement détectable par les mécanismes de protection d'*anti-rooting*



Fonctionnement interne

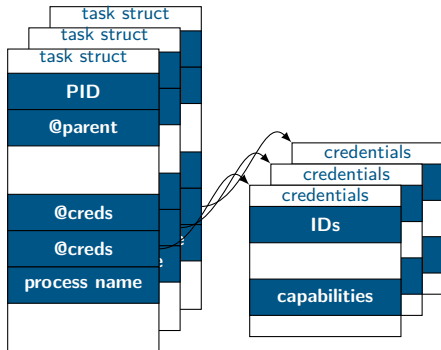


Une structure mémoire du noyau intéressante

- champ `comm`
- adresse de la structure parente
- Sous-structure `creds`

Possibilité d'élévation de privilèges de `sh` ?

- Modification en mémoire de la valeur de `creds`
- Nécessite un accès à la mémoire du noyau



Une structure mémoire du noyau intéressante

- champ `comm`
- adresse de la structure parente
- Sous-structure `creds`

Possibilité d'élévation de privilèges de `sh` ?

- Modification en mémoire de la valeur de `creds`
- Nécessite un accès à la mémoire du noyau

Credentials

uid
gid
suid
sgid
euid
egid
fsuid
fsgid
cap_inheritable
cap_permisive
cap_effective
cap_bset

Une structure mémoire du noyau intéressante

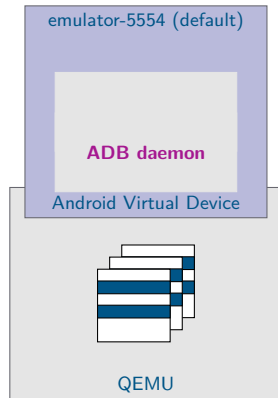
- champ `comm`
- adresse de la structure parente
- Sous-structure `creds`

Possibilité d'élévation de privilèges de `sh` ?

- Modification en mémoire de la valeur de `creds`
- Nécessite un accès à la mémoire du noyau

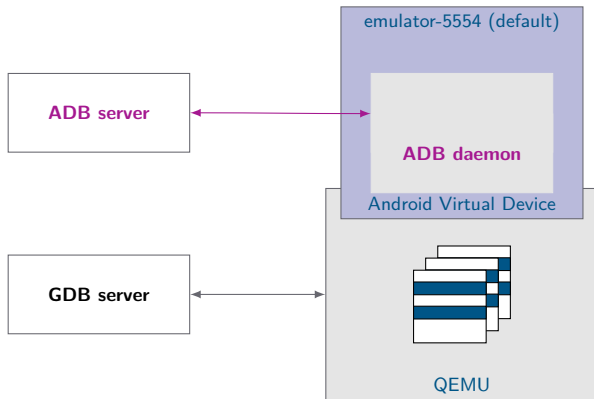
Credentials	sh	init
uid	0x7d0	0x00
gid	0x7d0	0x00
suid	0x7d0	0x00
sgid	0x7d0	0x00
euid	0x7d0	0x00
egid	0x7d0	0x00
fsuid	0x7d0	0x00
fsgid	0x7d0	0x00
cap_inheritable	0x00000000	0xffffffff
cap_permisive	0x00000000	0xffffffff
cap_effective	0x000000c0	0xffffffff
cap_bset	0xffffffffe0	0x00000000

Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



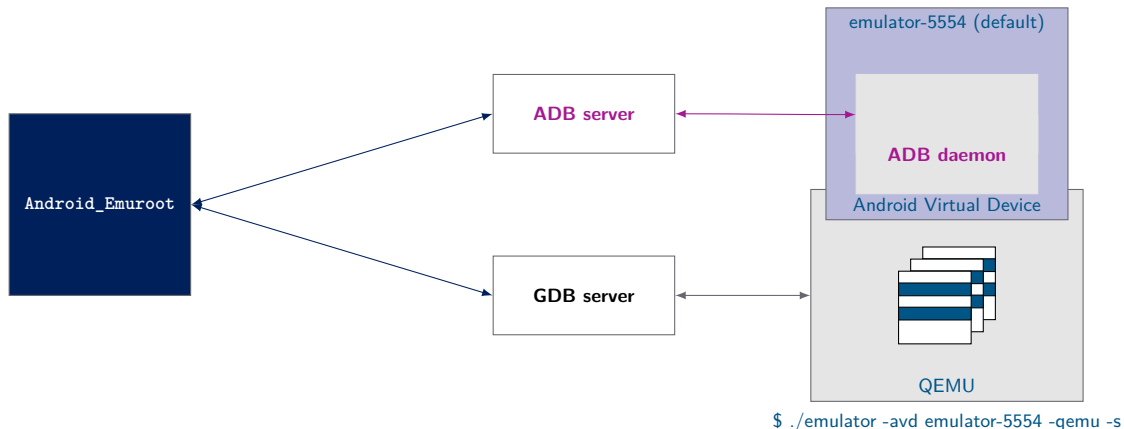
\$ `./emulator -avd emulator-5554 -qemu -s`

Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)

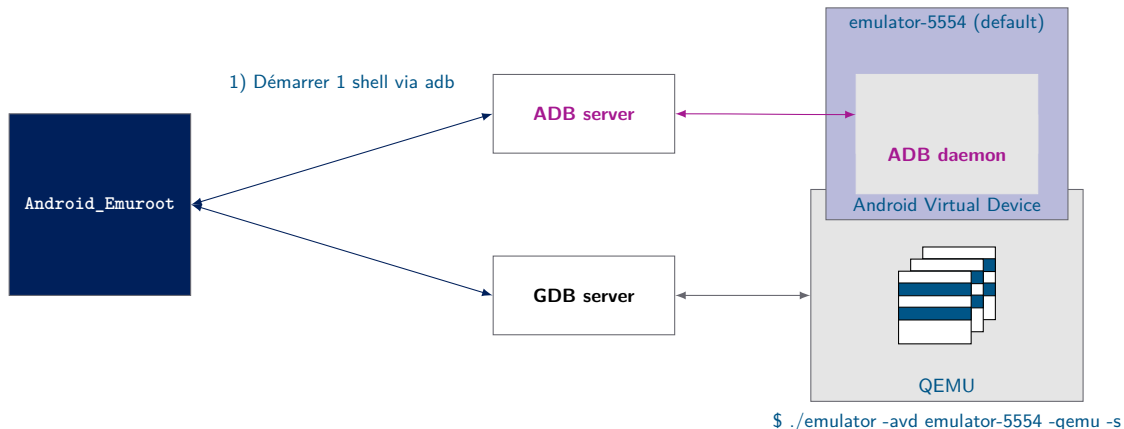


\$ `./emulator -avd emulator-5554 -qemu -s`

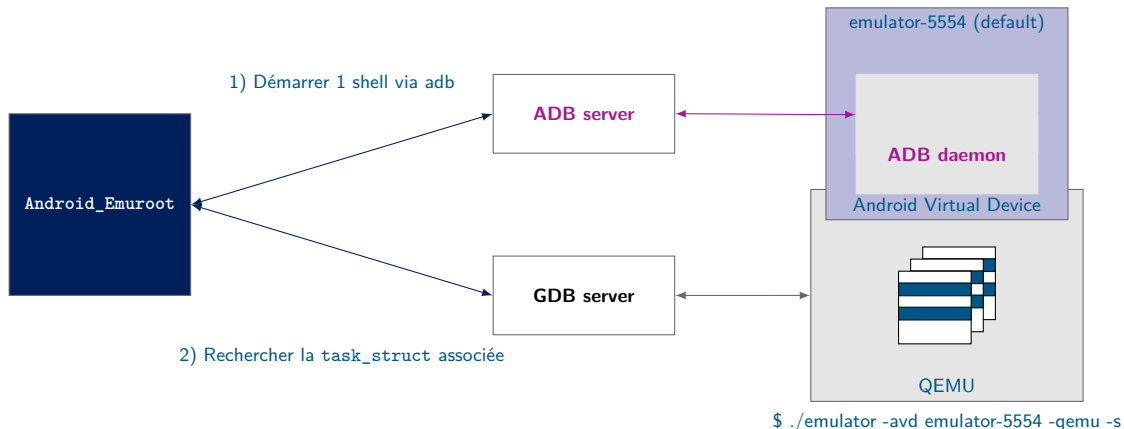
Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



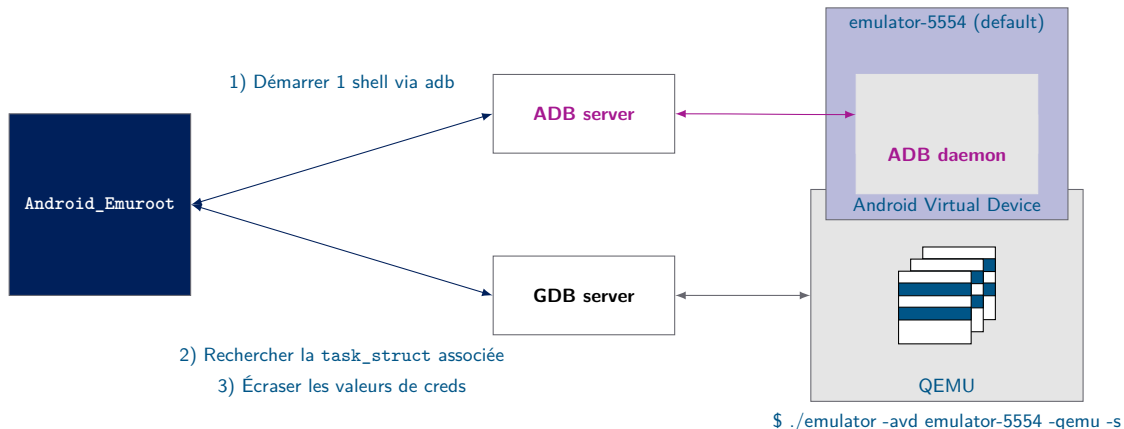
Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



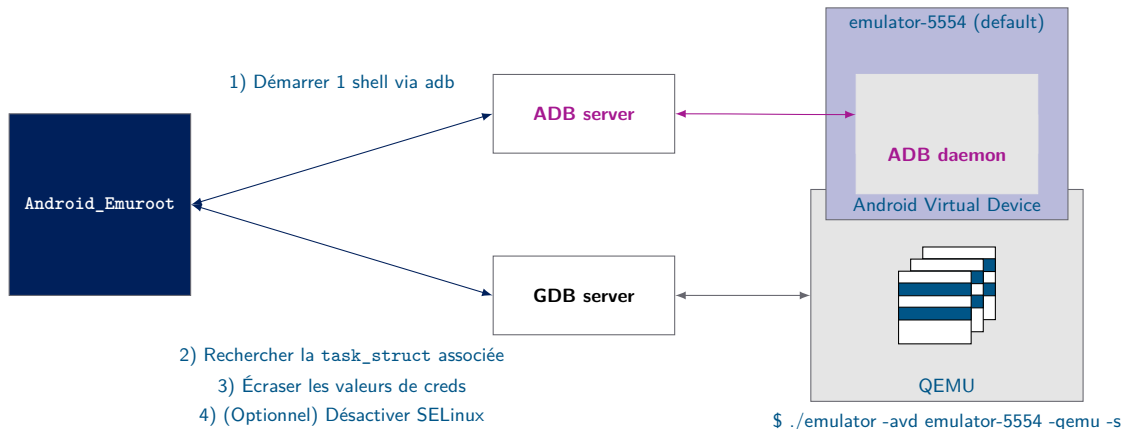
Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



Méthode "bourrin"

1. Nommage du shell avec un nom repérable
2. Recherche de ce nom dans la RAM émulée par QEMU (`gdb find`)
3. Heuristique de détermination d'appartenance des adresses trouvées à une `task_struct`

Avantages ?

- Intérêt : KASLR supporté
- Inconvénient : pas ultra rapide

Méthode basée sur le chaînage de `task_struct` entre elles (crédits R.Brechet/G.Teissier)

1. Adresse de la première `task_struct` (`init`) comme point de départ
2. Parcours de la liste des `task_struct`
3. Mise en cache de toutes les adresses de `task_struct` et leur nom associé
4. Nommage du shell avec un nom repérable
5. Recherche de ce nom dans la liste des `task_struct` trouvées

Avantages ?

- Intérêt : Recherche plus performante
- Inconvénient : Nécessite KASLR désactivé

Nécessaire pour connaître :

- Position des champs (comm, creds, etc.) dans la `task_struct`
- Adresses configuration SELinux
- Adresse de la `task_struct` du processus `init`

Versions actuellement supportées dans `Android_Emuroot` :

- Android 7.0 24 x86 3.10 google-api-playstore
- Android 7.1.1 25 x86 3.10 google-api-playstore
- Android 8.0 26 x86 3.18 google-api-playstore
- Android 8.1 27 x86 3.18 google-api-playstore

Méthodologie de recherche des offsets SELinux

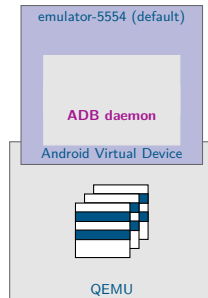
- Noyau du système Android fourni avec la SDK Android
 - Exemple pour Android 7.0 (api 24) :
`Sdk/system-images/android-24/google_apis_playstore/x86/kernel-ranchu`
- Noyau à décompresser
- Désassembler le noyau à la recherche de chaînes de caractères caractéristiques ("SELinux: Starting in enforcing mode")
- Lire le code autour de la chaîne identifiée pour repérer les adresses des variables globales SELinux

Utilisation et utilité



Android_Emuroot est facilement intégrable dans la boîte à outils d'analyse des applications Android

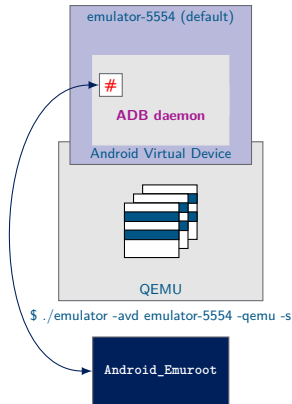
- Script Python
- Création standard des images Android pour l'émulateur
- Activation de l'option `-qemu -s`
- Utilisation des outils d'analyse dynamique : Frida, IDA, etc.



```
$ ./emulator -avd emulator-5554 -qemu -s
```

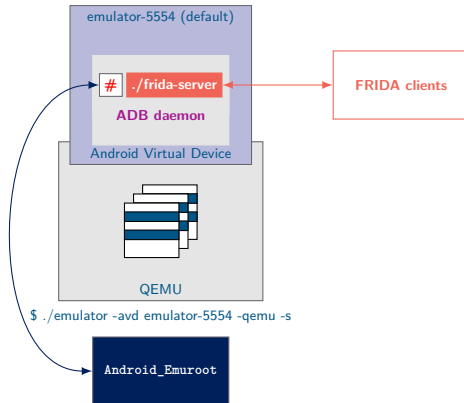

Android_Emuroot est facilement intégrable dans la boîte à outils d'analyse des applications Android

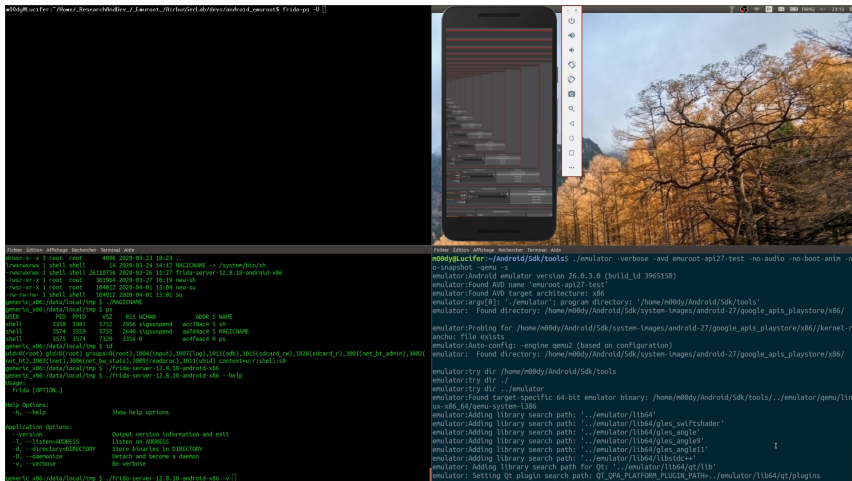
- Script Python
- Création standard des images Android pour l'émulateur
- Activation de l'option `-qemu -s`
- Utilisation des outils d'analyse dynamique : Frida, IDA, etc.



Android_Emuroot est facilement intégrable dans la boîte à outils d'analyse des applications Android

- Script Python
- Création standard des images Android pour l'émulateur
- Activation de l'option `-qemu -s`
- Utilisation des outils d'analyse dynamique : Frida, IDA, etc.





```
m08dy@lucifer:~/Research/Android_7_Jauneoot/WirbusSecLab/docs/android_emuroot$ frida ps -U
frida: ps -U: 1 shell root 4096 2020-03-23 18:23 ..
frida: ps -U: 1 shell root 14 2020-03-24 14:17 MAGICNAME -> /system/bin/sh
frida: ps -U: 1 shell root 26110756 2020-03-28 11:27 frida-server-12.8.18-android-x86
frida: ps -U: 1 root root 303504 2020-03-27 16:19 new-sh
frida: ps -U: 1 root root 104012 2020-04-01 15:04 new-su
frida: ps -U: 1 shell root 104012 2020-04-01 15:01 su
generic_x86:/data/local/tmp $ ./MAGICNAME
generic_x86:/data/local/tmp $ ls
USB PID PID VSIZE RSS NOHAN ADDR $ NAME
shell 3350 1981 5732 2936 sigsuspend acc78acc 5 sh
shell 3274 3350 5732 2646 sigsuspend acc78acc 5 MAGICNAME
shell 3075 3274 7320 3316 6 acc78acc 6 ps
generic_x86:/data/local/tmp $ ls
uid=0(root) gid=0(root) groups=0(root),1000(input),1001(log),1011(m08),1015(sdcard_rw),1020(sdcard_r),1001(net_bt_admin),1002(
net_bt),1003(net),1004(net_low_stats),1005(readonly),1011(uid) context=ui:/shell:sh
generic_x86:/data/local/tmp $ ./frida-server-12.8.18-android-x86
generic_x86:/data/local/tmp $ ./frida-server-12.8.18-android-x86 --help
Usage:
  frida [OPTION...]
Help Options:
  -h, --help            Show help options
Application Options:
  -v, --version          Output version information and exit
  -l, --listen=ADDRESS  Listen on ADDRESS
  -d, --directory=DIR    Store binaries in DIRECTORY
  -D, --daemonize        Detach and become a daemon
  -V, --verbose          Be verbose
generic_x86:/data/local/tmp $ ./frida-server-12.8.18-android-x86 --U
```

```
m08dy@lucifer:~/Android/Sdk/tools$ ./emulator -verbose -avd emuroot.apk27-test -no-audio -no-boot-anim -n
Q-snapshot -qemu -s
emulator:Android emulator version 26.0.3.0 (build_id 3965150)
emulator:Found AVD name 'emuroot-apk27-test'
emulator:Found AVD target architecture: x86
emulator:argv[0]: './emulator'; program directory: '/home/m08dy/Android/Sdk/tools'
emulator: Found directory: /home/m08dy/Android/Sdk/system-images/android-27/google_apis_playstore/x86/
emulator:Probing for /home/m08dy/Android/Sdk/system-images/android-27/google_apis_playstore/x86/kernel-r
anchu: file exists
emulator:Auto-config: -engine qemu2 (based on configuration)
emulator: Found directory: /home/m08dy/Android/Sdk/system-images/android-27/google_apis_playstore/x86/
emulator:try dir /home/m08dy/Android/Sdk/tools
emulator:try dir ./
emulator:try dir ../emulator
emulator:Found target-specific 64-bit emulator binary: /home/m08dy/Android/Sdk/tools/./emulator/qemu/libn
ux-x86_64/qemu-system-x86_64
emulator:Adding library search path: './emulator/lib64'
emulator:Adding library search path: './emulator/lib64/gles_swiftshader'
emulator:Adding library search path: './emulator/lib64/gles_angle'
emulator:Adding library search path: './emulator/lib64/gles_angle9'
emulator:Adding library search path: './emulator/lib64/gles_angle11'
emulator:Adding library search path: './emulator/lib64/libstdc++'
emulator:Adding library search path for Qt: './emulator/lib64/qt/lib'
emulator: Setting Qt plugin search path: QT_QPA_PLATFORM_PLUGIN_PATH=./emulator/lib64/qt/plugins
```

Une alternative aux limites des techniques existantes (e.g. décompilation/recompilation)

mobile-security.gitbook.io/mobile-security-testing-guide/android-testing-guide/0x05c-reverse-engineering-and-tampering#dynamic-analysis-on-non-rooted-devices

Introduction

Changelog

Frontispiece

OVERVIEW

Introduction to the Mobile Security
Testing Guide

Mobile App Taxonomy

Mobile App Security Testing

GENERAL MOBILE APP TESTING

Dynamic Analysis on Non-Rooted Devices

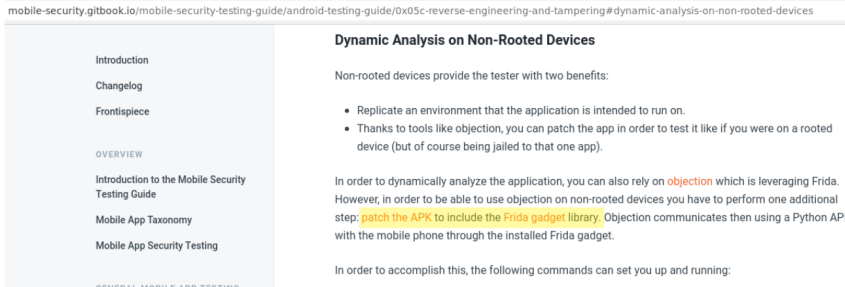
Non-rooted devices provide the tester with two benefits:

- Replicate an environment that the application is intended to run on.
- Thanks to tools like objection, you can patch the app in order to test it like if you were on a rooted device (but of course being jailed to that one app).

In order to dynamically analyze the application, you can also rely on **objection** which is leveraging Frida. However, in order to be able to use objection on non-rooted devices you have to perform one additional step: **patch the APK to include the Frida gadget library**. Objection communicates then using a Python API with the mobile phone through the installed Frida gadget.

In order to accomplish this, the following commands can set you up and running:

Une alternative aux limites des techniques existantes (e.g. décompilation/recompilation)



Rend possible l'analyse dynamique d'applications Android

- Qui implémentent des mécanismes de détection de rooting
- Qui ne fonctionnent pas sur d'autres émulateurs que Google API Playstore
- Dont la décompilation/recompilation est rendue difficile par d'autres mécanismes de protection

Conclusion

Android_Emuroot est un outil

- Permettant de disposer d'un accès root sur un émulateur *Google API Playstore*
- Utile pour l'analyse dynamique d'applications protégées
- Disponible sur github et ouvert aux contributions



`mouad.abouhali@airbus.com`



`anais.gantet@airbus.com`



`https://github.com/airbus-seclab/android\_emuroot`



`@AirbusSecLab`



`https://airbus-seclab.github.io`