`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

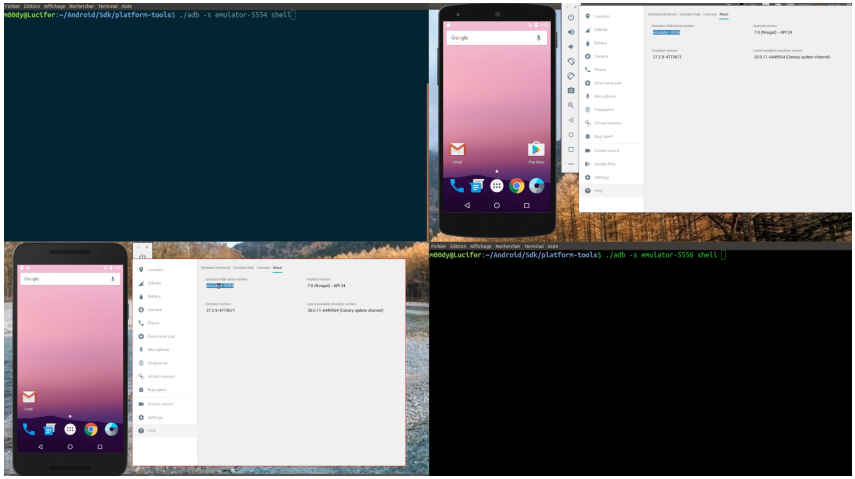
AIRBUS

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.

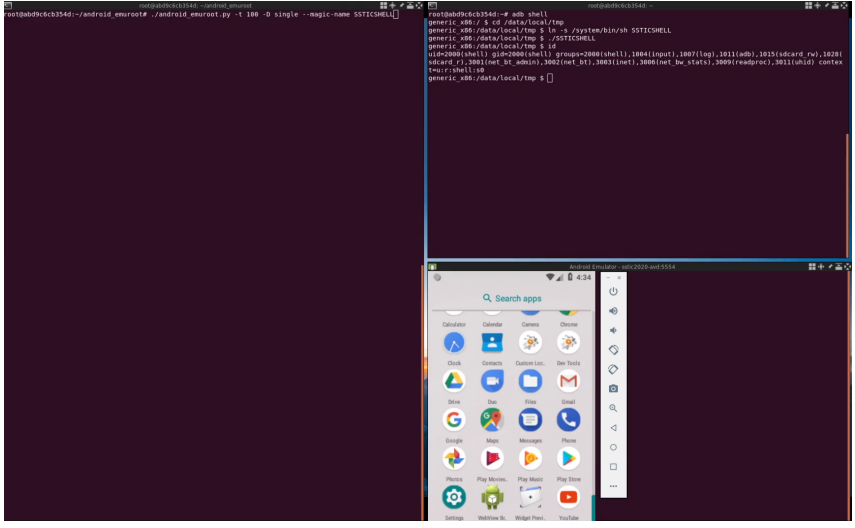
Introduction

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*


root@abd9c6cb3544: ~/android_emuroot
root@abd9c6cb3544:~/android_emuroot# ./android_emuroot.py -t 100 -D single --magic-name SSTICSHELL
root@abd9c6cb3544: ~
root@abd9c6cb3544:~# adb shell
generic_x86:/ $ cd /data/local/tmp
generic_x86:/data/local/tmp $ ln -s /system/bin/sh SSTICSHELL
generic_x86:/data/local/tmp $ ./SSTICSHELL
generic_x86:/data/local/tmp $ 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),3011(uhid) context=u:r:shell:s0
generic_x86:/data/local/tmp $
Android Emulator - nitc2020-avd:5554
4:34
Search apps
Calculator
Calendar
Camera
Chrome
Clock
Contacts
Custom Loc...
Dev Tools
Drive
Duo
Files
Gmail
Google
Maps
Messages
Phone
Photos
Play Movies...
Play Music
Play Store
Settings
WebView Br...
Widget Previ...
YouTube

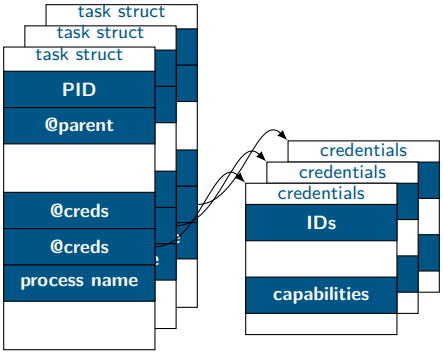
Fonctionnement interne

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

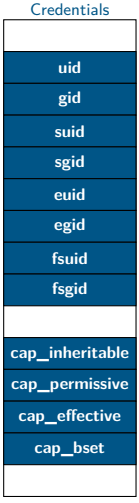


Une structure mémoire du noyau intéressante

- 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

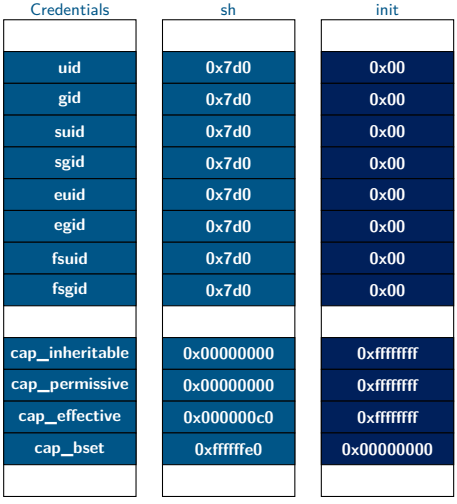


Une structure mémoire du noyau intéressante

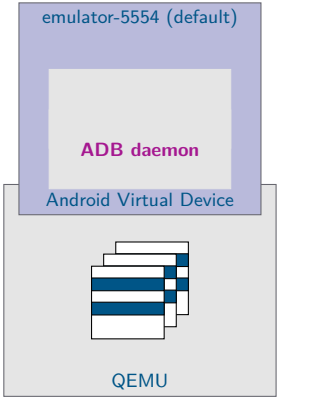
- champ `comm`
- adresse de la structure parente
- Sous-structure `creds`

Possibilité d'élévation de privilèges de `sh` ?

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

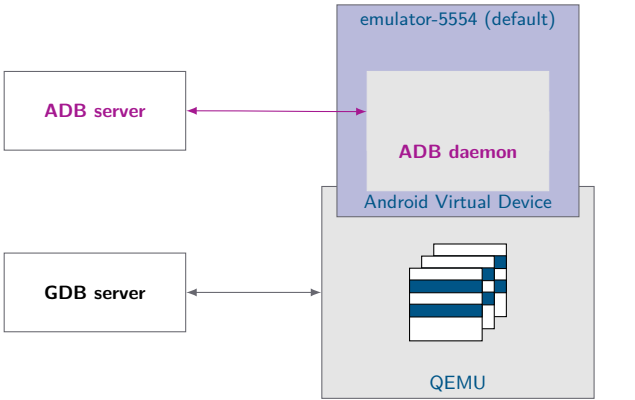


Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



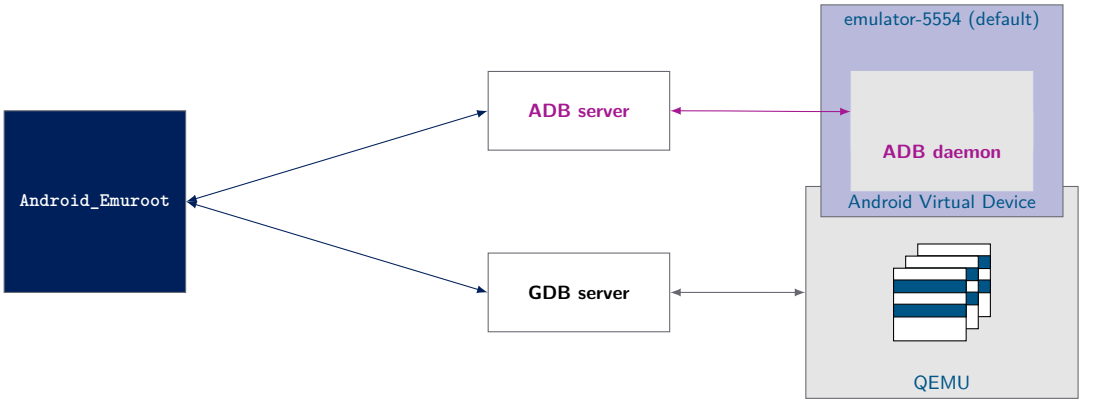
$ ./emulator -avd emulator-5554 -qemu -s

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



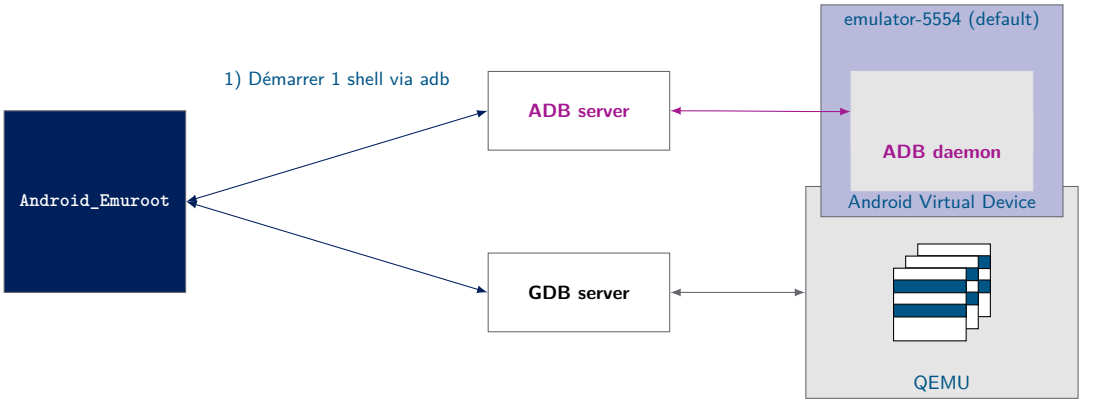

```
$ ./emulator -avd emulator-5554 -qemu -s
```

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

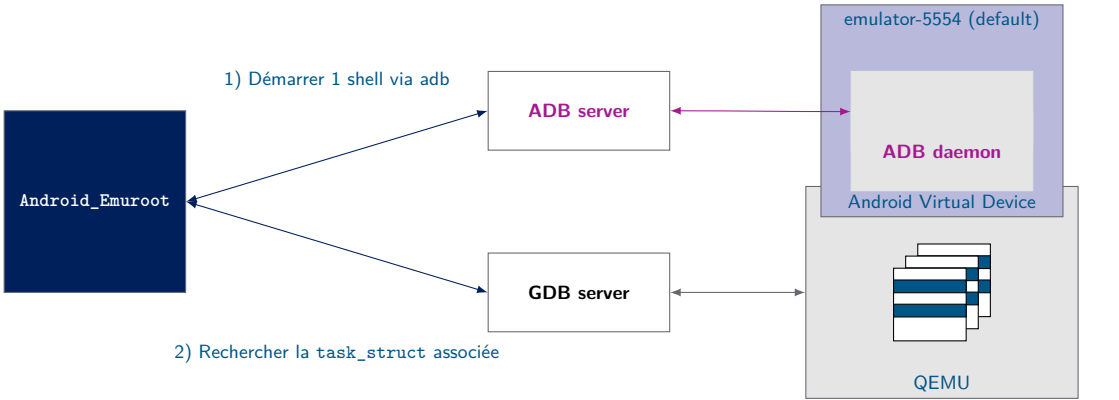



```
$ ./emulator -avd emulator-5554 -qemu -s
```

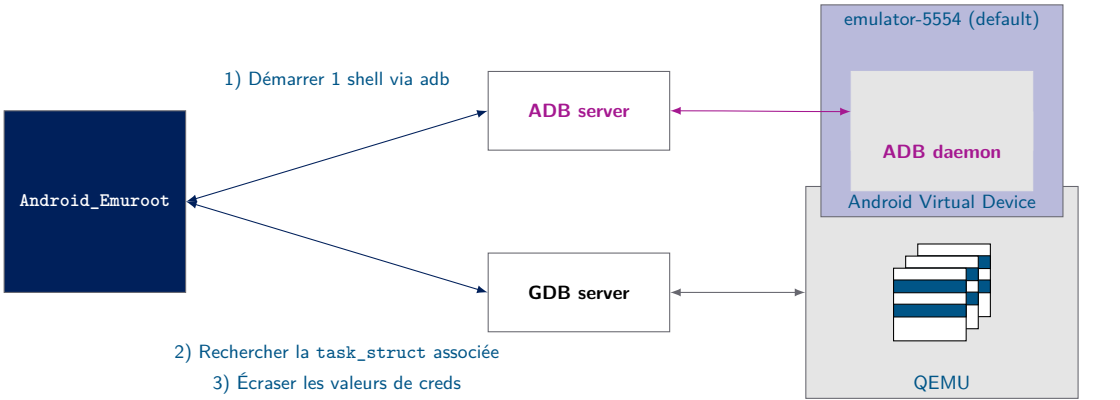
Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



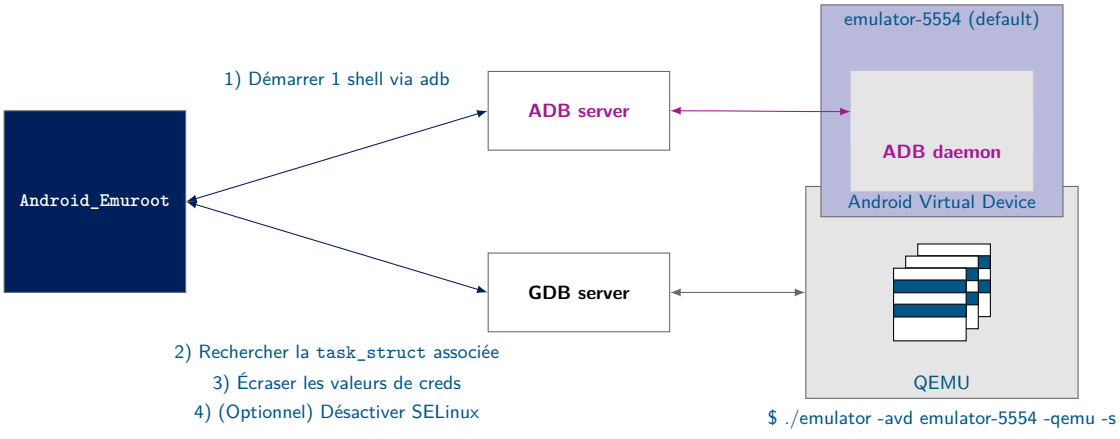
Avec l'option `qemu -s` de l'outil `emulator` (SDK Android Studio)



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



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



Stratégies de recherche mémoire ?

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

Stratégies de recherche mémoire ?

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

Étape de RE

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`

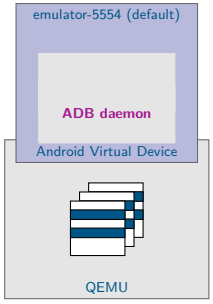
Méthodologie de recherche des offsets SELinux

- 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 boite à outils d'analyse des applications Android

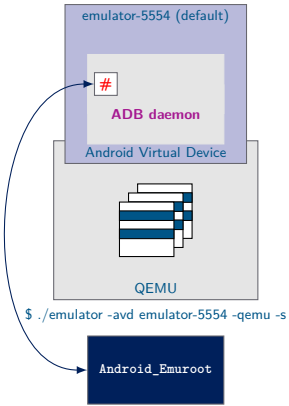
- Script Python
- Création standard des images Android pour l'émulateur
- Activation de l'option -qemu -s
- Utilisation des outils d'analyse dynamique : Frida, IDA, etc.



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

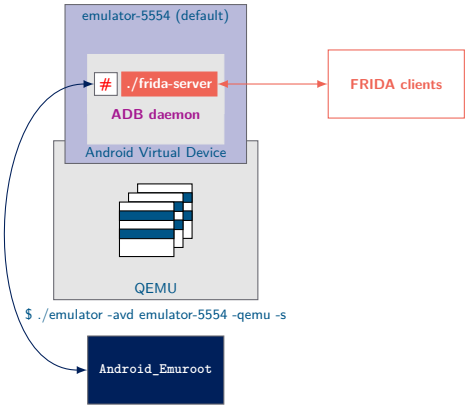
Android_Emuroot est facilement intégrable dans la boite à outils d'analyse des applications Android

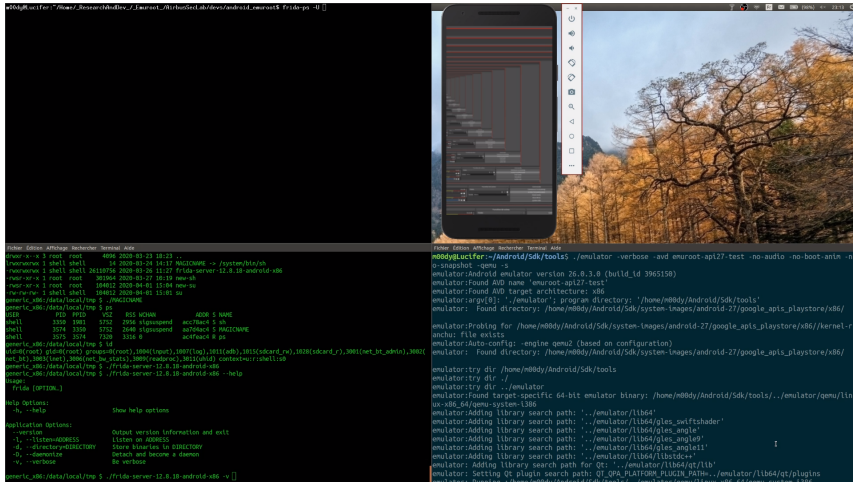
- Script Python
- Création standard des images Android pour l'émulateur
- Activation de l'option -qemu -s
- Utilisation des outils d'analyse dynamique : Frida, IDA, etc.



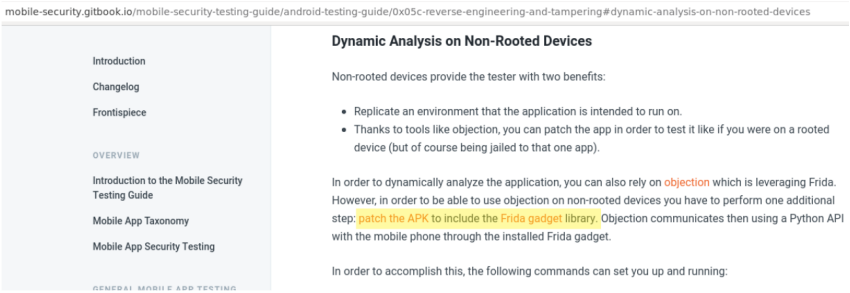
Android_Emuroot est facilement intégrable dans la boite à outils d'analyse des applications Android

- 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.

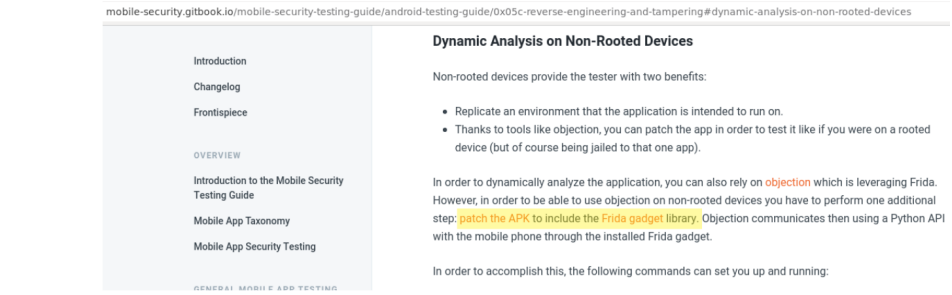




Une alternative aux limites des techniques existantes (e.g. décompilation/recompilation)



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

Questions ?

mouad.abouhali@airbus.com
anais.gantet@airbus.com
https://github.com/airbus-seclab/android_emuroot
@AirbusSecLab
https://airbus-seclab.github.io