Android_Emuroot:

Abusing Google Play emulator debugging to RE non-cooperative apps as root

Anaïs Gantet

Blackhoodie'18 - November 16, 2018

AIRBUS

Android_Emuroot Outline

Introduction

Android security model around Android applications

- Linux DAC (Discretionary Access Control) for application sandboxing
  - 1 Linux user for each application (app ID)
  - 1 dedicated data directory for each application (`RW` reserved to the app ID)
- SELinux MAC (Mandatory Access Control)
  - Access to objects (file, socket, etc.) conditioned by rules defined in `sepolicy`

Android world in a nutshell

Android security model around Android applications

- Linux DAC (Discretionary Access Control) for application sandboxing
  - 1 Linux user for each application (app ID)
  - 1 dedicated data directory for each application (`RW` reserved to the app ID)
- SELinux MAC (Mandatory Access Control)
  - Access to objects (file, socket, etc.) conditioned by rules defined in `sepolicy`

Android applications

- File format: `apk` (Android PacKage) containing
  - Dalvik code (`.dex`) resulting from Java code compilation
  - Native code (`.so` libraries)
  - Resources and certificates for code signing
- Need device configuration requirements (recent kernel version, `Google Play Services`, etc.)
- Can embed additional security measures like **rooting detection mechanisms**

Examples of common rooting checks in Android apps

Check unwanted applications	`ls -l /system/app/Superuser.apk`
	`pm list packages | grep eu.chainfire.supersu`
	`pm list packages | grep magisk`

Check unwanted binaries	`ls -l /system/bin/su /system/xbin/su`
	`ls -l system/su /system/bin/.ext/.su`
	`ls -l /system/usr/we-need-root/su-backup`

Check shell permissions	`id | grep root` `ps |grep adbd | grep root`
Check file system changes (RW, etc.)	`ls -lR /system | grep -e :$ -e [r-][w-]x` `ls -laR /system | grep [r-][w-]s[-r' ']`

Check build tag, hardware/system properties	`getprop ro.secure`
	`getprop | grep ro.product.model`
	`getprop | grep ro.build.type`

Some libraries/implementations

- rootbeer, RootTools, RootManager, etc.

Reverse engineering (RE) android applications - the common way

Why?

- Search app vulnerabilities
- Check potential privacy information leak
- etc.

How?

- Decompress the apk (`apktool`)
- Decompile Java code (`JEB`, `procyon`, `dex2jar`, etc.)
- Browse the app data (via `ADB` shell)
- Debug the app step by step (`IDA debugger`)
- Hook and trace functions (`Frida for Android`)
  - `frida-server` must run on the device with root privileges

Reverse engineering (RE) android applications - the common way

Why?

- Search app vulnerabilities
- Check potential privacy information leak
- etc.

How?

- Decompress the apk (`apktool`)
- Decompile Java code (`JEB`, `procyon`, `dex2jar`, etc.)
- Browse the app data (via `ADB` shell)
- Debug the app step by step (`IDA debugger`)
- Hook and trace functions (`Frida for Android`)
  - `frida-server` must run on the device with root privileges

Important: RE often requires ROOT access on the device

Existing devices and root access

Android devices

- Physical devices (`user` build)
- Emulated devices
  - `default` (`eng` build)
  - `google-api` (`userdebug` build)
  - `google-api-playstore` **(`user` build)**

Existing devices and root access

Android devices

- Physical devices (`user` build)
- Emulated devices
  - `default` (`eng` build)
  - `google-api` (`userdebug` build)
  - `google-api-playstore` **(`user` build)**

Root shell available?

- `eng` build: root shell by default
- `userdebug` build: root shell optional but possible

Problem: easily detectable
(`/system/xbin/su` binary present)

Existing devices and root access

Android devices

- Physical devices (`user` build)
- Emulated devices
  - `default` (`eng` build)
  - `google-api` (`userdebug` build)
  - `google-api-playstore` **(`user` build)**

Root shell available?

- `eng` build: root shell by default
- `userdebug` build: root shell optional but possible
- `user` build: root access not allowed but possible by using known rooting techniques
  - Changing boot image or system image
  - Crafting custom ROM
  - Rooting via Exploits, etc.

Problem: easily detectable (`/system/xbin/su` binary present)

Problem: methods already checked by the rooting detection mechanisms

How to RE applications with ROOTED shell
without being spotted by the rooting detection?

Our approach

The main idea

- Start from a clean Android system build
- Launch a non-root shell
- Understand how shell process information is stored by the Linux kernel
- Patch the memory on the fly to change shell rights to root

Our approach

The main idea

- Start from a clean Android system build
- Launch a non-root shell
- Understand how shell process information is stored by the Linux kernel
- Patch the memory on the fly to change shell rights to root

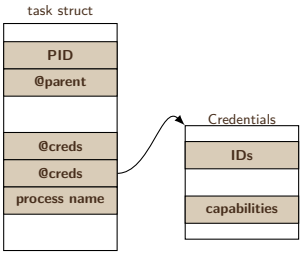
Chosen device: *Google API Playstore* emulator

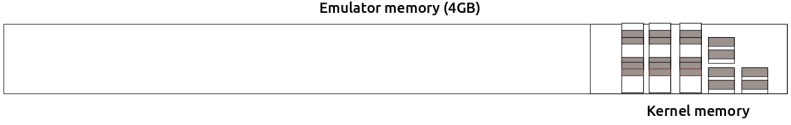
- Because it is an emulated device
  - Device memory easier to access
  - `GDB` attachable to read/write the memory (`-qemu -s`)
  - A lot of device versions testable
- Because it uses the `user` build variant
  - Shell server (`adbd`) as root disabled
  - `Google Play Services` installed

Concepts

Step 1: Understand the Android process metadata organization

- Process metadata stored in `task_struct`
- Interesting fields to identify the right `task_struct`
  - Process PID
  - Process name
- Other interesting fields
  - Pointer to parent process
  - Pointers to credential structures
    - Used by the kernel for permission checks





https://android.googlesource.com/kernel/common/+/android-3.10/include/linux/sched.h

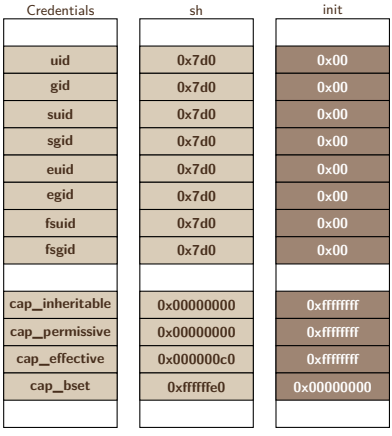
Step 2: Understand `cred` structure content

- Security context of a task defined in `cred` structure
- Interesting fields
  - Linux user identifier (UID)
  - Linux effective user identifier (EUID)
  - Set of flags enabling or disabling Linux capabilities (`CAP_CHOWN`, `CAP_DAC_OVERRIDE`, `CAP_DAC_READ_SEARCH`, etc.)
  - `security` pointer with SELinux task information
  - etc.

https://android.googlesource.com/kernel/common/+/android-3.10/include/linux/cred.h

Step 2: Understand `cred` structure content

- Security context of a task defined in `cred` structure
- Interesting fields
  - Linux user identifier (UID)
  - Linux effective user identifier (EUID)
  - Set of flags enabling or disabling Linux capabilities (`CAP_CHOWN`, `CAP_DAC_OVERRIDE`, `CAP_DAC_READ_SEARCH`, etc.)
  - `security` pointer with SELinux task information
  - etc.



https://android.googlesource.com/kernel/common/+/android-3.10/include/linux/cred.h

Step 3: From non-rooted to rooted shell

In ADB shell

- Link /system/bin/sh to a file with magic name
- Launch the created file

```
root@850e9484e78e:~# adb shell
generic_x86:/ $ ln -s /system/bin/sh /data/local/tmp/MAGICNAME
generic_x86:/ $ ./data/local/tmp/MAGICNAME
```

With GDB debugger

- Search MAGICNAME task_struct in emulator kernel memory (find 0xc0000000,+0x40000000,"MAGICNAME")
- Step through parent task_struct until finding init
- Get init cred structure pointer
- Overwrite MAGICNAME cred pointer by the init one
- Set SELinux mode to permissive

Step 3: From non-rooted to rooted shell

In ADB shell

- Link /system/bin/sh to a file with magic name
- Launch the created file

```
root@850e9484e78e:~# adb shell
generic_x86:/ $ ln -s /system/bin/sh /data/local/tmp/MAGICNAME
generic_x86:/ $ ./data/local/tmp/MAGICNAME
```

With GDB debugger

- Search MAGICNAME task_struct in emulator kernel memory
  (find 0xc0000000,+0x40000000,"MAGICNAME")
- Step through parent task_struct until finding init
- Get init cred structure pointer
- Overwrite MAGICNAME cred pointer by the init one
- Set SELinux mode to permissive

Step 3: From non-rooted to rooted shell

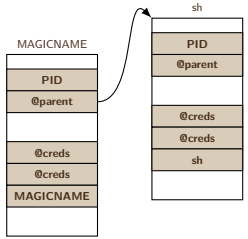
In ADB shell

- Link /system/bin/sh to a file with magic name
- Launch the created file

```
root@850e9484e78e:~# adb shell
generic_x86:/ $ ln -s /system/bin/sh /data/local/tmp/MAGICNAME
generic_x86:/ $ ./data/local/tmp/MAGICNAME
```

With GDB debugger

- Search MAGICNAME task_struct in emulator kernel memory (find 0xc0000000,+0x40000000,"MAGICNAME")
- Step through parent task_struct until finding init
- Get init cred structure pointer
- Overwrite MAGICNAME cred pointer by the init one
- Set SELinux mode to permissive



Step 3: From non-rooted to rooted shell

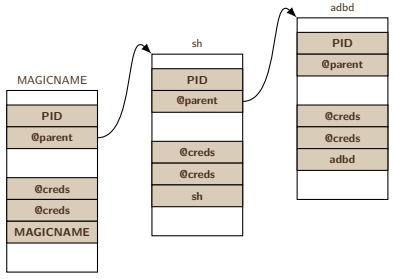
In ADB shell

- Link /system/bin/sh to a file with magic name
- Launch the created file

```
root@850e9484e78e:~# adb shell
generic_x86:/ $ ln -s /system/bin/sh /data/local/tmp/MAGICNAME
generic_x86:/ $ ./data/local/tmp/MAGICNAME
```

With GDB debugger

- Search MAGICNAME task_struct in emulator kernel memory (find 0xc0000000,+0x40000000,"MAGICNAME")
- Step through parent task_struct until finding init
- Get init cred structure pointer
- Overwrite MAGICNAME cred pointer by the init one
- Set SELinux mode to permissive



Step 3: From non-rooted to rooted shell

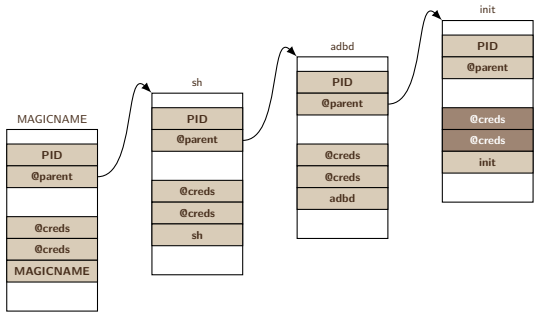
In ADB shell

- Link /system/bin/sh to a file with magic name
- Launch the created file

```
root@850e9484e78e:~# adb shell
generic_x86:/ $ ln -s /system/bin/sh /data/local/tmp/MAGICNAME
generic_x86:/ $ ./data/local/tmp/MAGICNAME
```

With GDB debugger

- Search MAGICNAME task_struct in emulator kernel memory (find 0xc0000000,+0x40000000,"MAGICNAME")
- Step through parent task_struct until finding init
- Get init cred structure pointer
- Overwrite MAGICNAME cred pointer by the init one
- Set SELinux mode to permissive



Step 3: From non-rooted to rooted shell

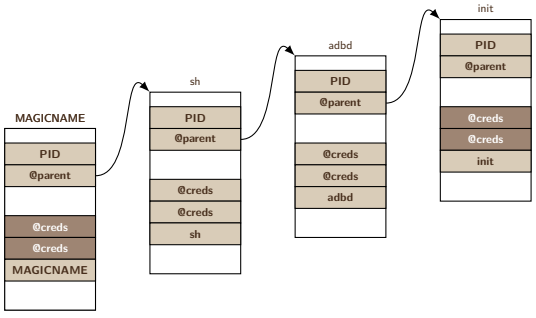
In ADB shell

- Link /system/bin/sh to a file with magic name
- Launch the created file

```
root@850e9484e78e:~# adb shell
generic_x86:/ $ ln -s /system/bin/sh /data/local/tmp/MAGICNAME
generic_x86:/ $ ./data/local/tmp/MAGICNAME
```

With GDB debugger

- Search MAGICNAME task_struct in emulator kernel memory (find 0xc0000000,+0x40000000,"MAGICNAME")
- Step through parent task_struct until finding init
- Get init cred structure pointer
- Overwrite MAGICNAME cred pointer by the init one
- Set SELinux mode to permissive



Note: Technique similar to *Token stealing* on Windows

Practice

What is Android_Emuroot?

- Tool as Python script based on open-source libraries
  - pygdbmi[1] for GDB commands
  - pure-python-adb[2] for ADB shell commands
- Features
  - Automate the memory modification
  - Give the possibility to spawn more than 1 rooted shell
  - Support of multiple kernel versions

[1] https://pypi.org/project/pygdbmi
[2] https://pypi.org/project/pure-python-adb

`single --magic-name NAME`

- Change the credentials of the shell given in parameter
- Note: the `shell_name` must run beforehand (process must exist)

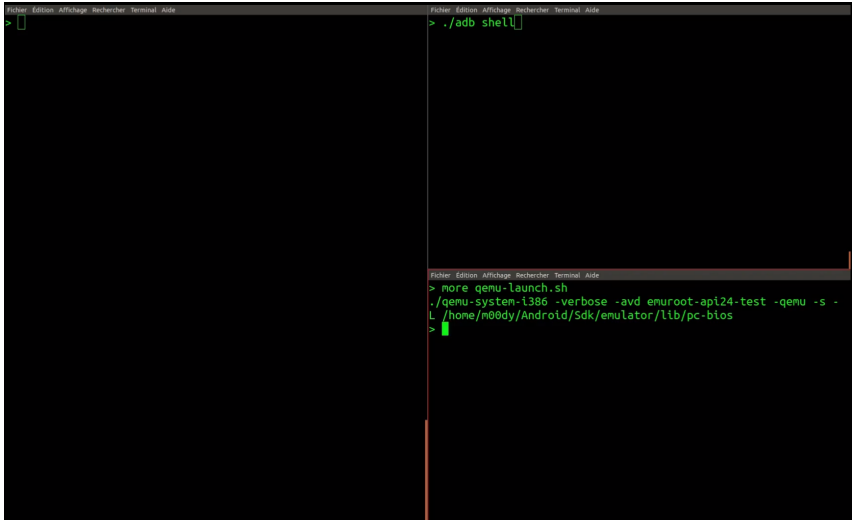
`adbd [--stealth]`

- Modify the `adb` server credentials on the fly
- `[--stealth]` additional option: keep `adbd EUID` intact (for anti-detection reasons)

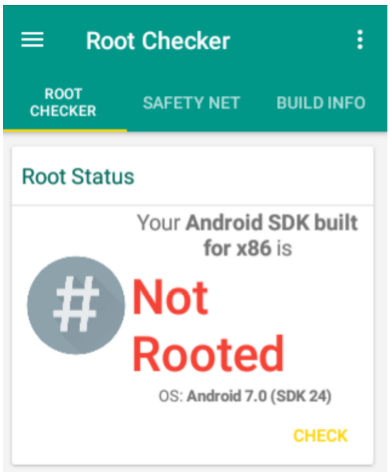
`setuid --path NAME`

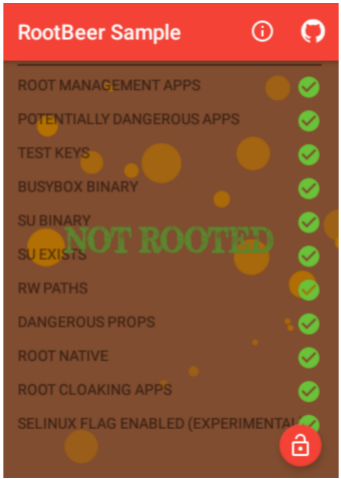
- Install a `sh` binary with setuid root in `NAME` (default: /data/local/tmp/rootsh)
- Note: the setuid binary must be launched with `-p` option

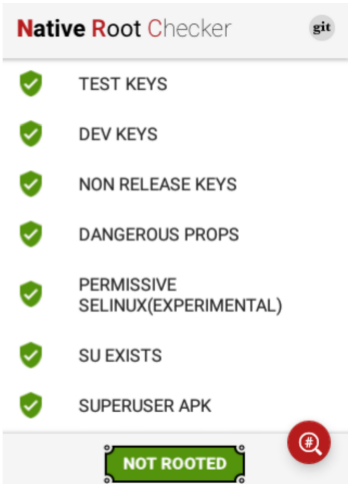
Demo


Fichier Edition Affichage Rechercher Terminal Aide
>
Fichier Edition Affichage Rechercher Terminal Aide
> ./adb shell
Fichier Edition Affichage Rechercher Terminal Aide
> more qemu-launch.sh
./qemu-system-i386 -verbose -avd emuroot-api24-test -qemu -s -
L /home/m00dy/Android/Sdk/emulator/lib/pc-bios
>

Detected?


Root Checker
ROOT CHECKER
SAFETY NET
BUILD INFO
Root Status
Your Android SDK built for x86 is
Not Rooted
OS: Android 7.0 (SDK 24)
CHECK


RootBeer Sample
ROOT MANAGEMENT APPS
POTENTIALLY DANGEROUS APPS
TEST KEYS
BUSYBOX BINARY
SU BINARY
SU EXISTS
RW PATHS
DANGEROUS PROPS
ROOT NATIVE
ROOT CLOAKING APPS
SELINUX FLAG ENABLED (EXPERIMENTAL)
NOT ROOTED


Native Root Checker
git
TEST KEYS
DEV KEYS
NON RELEASE KEYS
DANGEROUS PROPS
PERMISSIVE SELINUX(EXPERIMENTAL)
SU EXISTS
SUPERUSER APK
NOT ROOTED

Android_Emuroot single --magic-name NAME

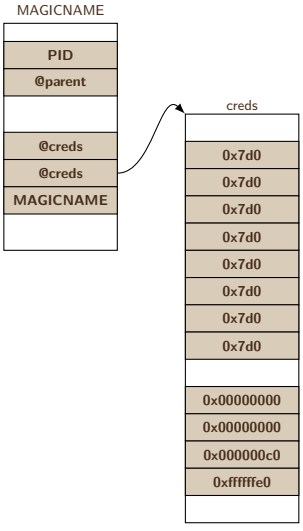
User contribution

Before rooting:

```
adb shell
$ ln -s /system/bin/sh MAGICNAME
$ ./MAGICNAME
```

Android_Emuroot contribution

- Specific shell credentials overwriting (IDs+capabilities)



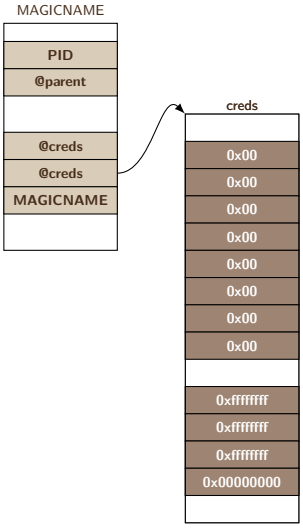
User contribution

Before rooting:

```
adb shell
$ ln -s /system/bin/sh MAGICNAME
$ ./MAGICNAME
```

Android_Emuroot contribution

- Specific shell credentials overwriting (IDs+capabilities)



Android_Emuroot contribution

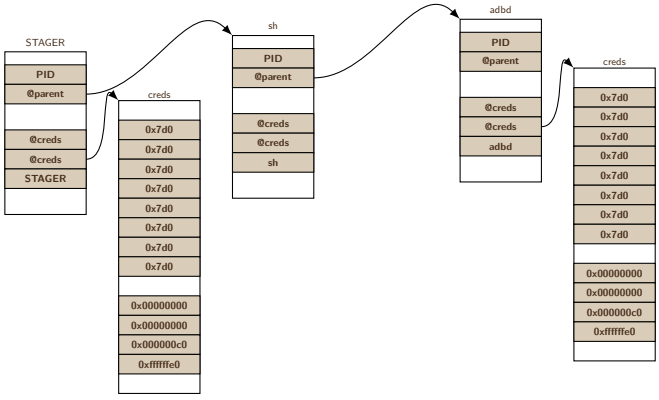
- adbd credentials modification (IDs+capabilities)

User contribution

Before rooting: None

After rooting:

```
adb shell
# echo "ok, I'm root now :)"
ok, I'm root now :)
```



Android_Emuroot contribution

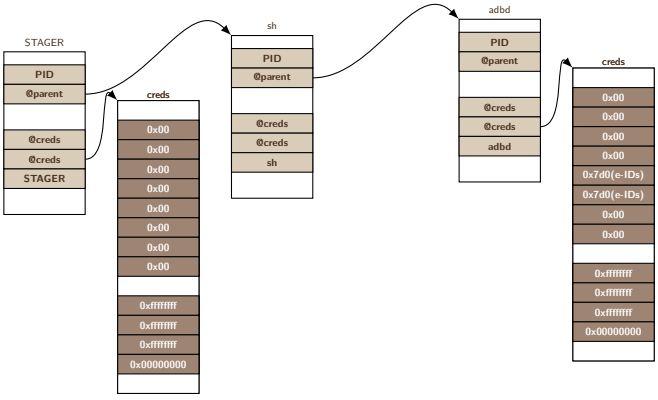
- adbd credentials modification (IDs+capabilities)

User contribution

Before rooting: None

After rooting:

```
adb shell
# echo "ok, I'm root now :)"
ok, I'm root now :)
```



Android_Emuroot setuid --path NAME

Android_Emuroot contribution

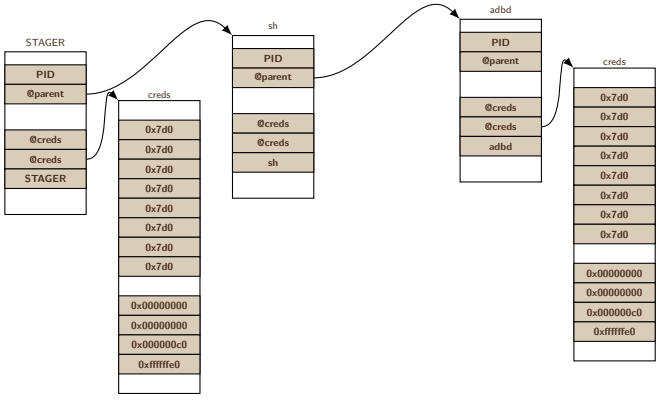
- A setuid binary on the file system
- /data remounted without nosuid
- adbd capabilities modification

User contribution

Before rooting: None

After rooting:

```
adb shell
$./data/local/tmp/rootsh -p
# echo "ok, I'm root now :)"
ok, I'm root now :)
```



Android_Emuroot contribution

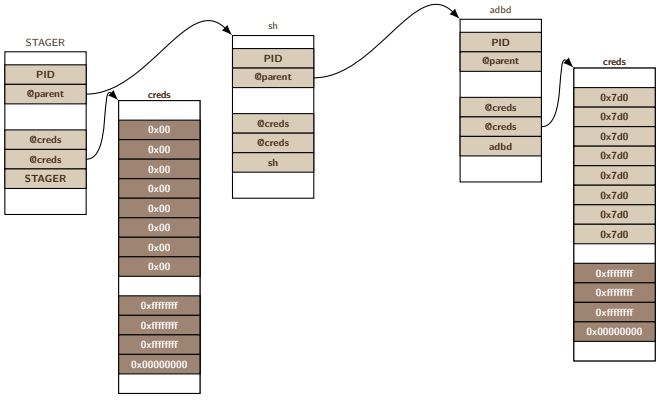
- A setuid binary on the file system
- /data remounted without nosuid
- adbd capabilities modification

User contribution

Before rooting: None

After rooting:

```
adb shell
$./data/local/tmp/rootsh -p
# echo "ok, I'm root now :)"
ok, I'm root now :)
```



Conclusion

Conclusion

`Android_Emuroot` today

- Give a rooted environment to help RE Android applications despite rooting-detection mechanisms
- Based on playing with `GDB` debugger attached to Android emulator memory
- Currently supported kernel versions: `google-api-playstore` 24 to 27, x86
- Total time spent: about 35 person-days

Tool limitations

- Technique not persistent to device reboot
- Options giving multiple root shells can be detectable
- Technique not applicable if the applications refuse to run on emulators

Outlooks

Next steps?

- Still a work in progress
- Support more kernel architectures/versions?
- Extend the rooting technique to other emulated systems having `GDB` stub (e.g. VMWare)?

Thanks for your attention! :)

https://github.com/airbus-seclab/android_emuroot

mouad.abouhali@airbus.com, anais.gantet@airbus.com

https://airbus-seclab.github.io