SYNACKTIV

The Android Security Model

THCON 2023

2023/04/21

Agenda

- Introduction
- Security Model
- Android Permissions
- Hardening and Mitigations
- Conclusion

Presentation

SYNACKTIV

- **Jean-Baptiste Cayrou**
  - Security researcher @Synacktiv
  - Vulnerability research & exploitation
- **Synacktiv**
  - Offensive security company
  - Based in France
  - ~140 Ninjas
  - We are hiring!!!

Introduction

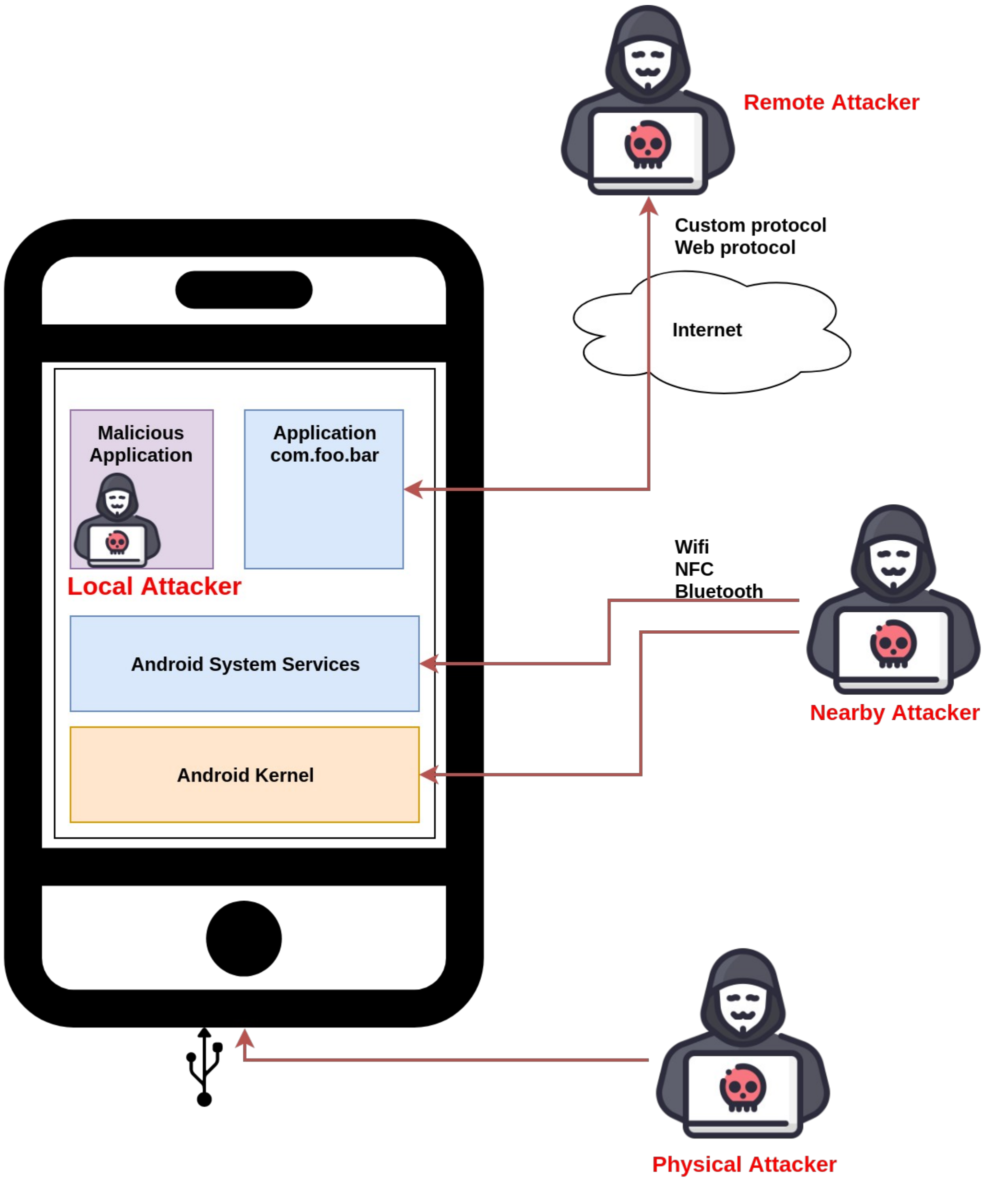
- **Android is an open-source project led by Google**
  - Lastest version is Android 13
  - ~70% mobile devices worldwide use Android
- **It is based on a Linux kernel with the “binder” driver enabled for process interactions**
- **In userland, applications are Java packages that run in a specific JVM**

Introduction

SYNACKTIV

- **Our smartphones contain a lot of sensitive data**
  - Emails and conversations
  - Photos and videos
- **And they have many sensors**
  - Camera
  - Microphone
  - GPS
- **Access to this data and sensors must be protected against compromised or malicious applications**

Device Threats



Device Threats

- **Applications may be malicious or compromised**
  - For instance, by exploiting browser vulnerabilities
- **It is essential to prevent attackers from accessing:**
  - Data
  - Sensors
- **Attackers might bypass restrictions by exploiting other system vulnerabilities**
  - Perform a LPE (Local Privileged Escalation)

→ Reduce the risks and make LPE as difficult as possible

Security Model

Security Model

- **Android considers applications as untrusted**
- **Least privilege principle**
  - Only permit each component to perform necessary actions
  - Implement isolation and sandboxing of processes and applications
  - Restrict interactions between components
- **Hardening and exploit mitigations**
  - Make vulnerabilities difficult to exploit
  - Ideally, make vulnerabilities unexploitable

Isolation and sandboxing

- **Android uses Linux features to isolate applications and daemons**
  - Linux users, groups (DAC security)
  - SELinux (MAC security)
  - SECCOMP to filter syscalls

Isolation and sandboxing - Linux users

- **Some user IDs are reserved for system use**
  - system is 1000, shell is 2000, bluetooth is 1002, etc.
  - Applications UID range is 10000 → 19999
- **Applications**
  - Applications get a UID at installation time
  - Get a dedicated folder for data storage
    - Not able to read other applications folders (Unix file permissions)
    - */data/data/<PKG_NAME>/*

Isolation and sandboxing - SELinux

SYNACKTIV

- **SELinux: Security Enhanced Linux**
  - Enforced starting with Android 4.4 (2013)

- **Implemented as a Linux Security Module (LSM)**
  - Implements security filtering hooks which are called inside the kernel

```c
// Extract of fs/ioctl.c
SYSCALL_DEFINE3(ioctl, unsigned int, fd, unsigned int, cmd, unsigned long, arg)
{
	struct fd f = fdget(fd);
	int error;
	if (!f.file)
		return -EBADF;

	error = security_file_ioctl(f.file, cmd, arg);

	if (error)
		goto out;
	error = do_vfs_ioctl(f.file, fd, cmd, arg);
	// [...]
```

Isolation and sandboxing - SELinux

SYNACKTIV

- **The SELinux policy defines rules between subject, objects and actions**
- **Subjects and objects are identified with security context called SELinux labels**
- **The firmware contains a set of SELinux rules (the policy) loaded during the boot**
  - Actions not included in the rules are forbidden
- **Rule example**

```
allow appdomain app_data_file:file rw_file_perms;
```

subjects

objects

actions

{getattr open read ioctl lock w_file_perms}

Isolation and sandboxing - SECCOMP

SYNACKTIV

- **SECCOMP is a Linux feature that filters syscalls**
  - Enforced system-wide since Android 8.0
  - Reduces the Kernel attack surface
- **Filtering profiles are directly defined in the Android libc (Bionic)**
  - Profiles: System, Application, Application Zygote
  - Filtering profile is enabled when an application starts
    - Configured by the JVM during application launch

Isolation and sandboxing - SECCOMP

SYNACKTIV

- **The system profile is relatively permissive**
  - 17/271 ARM64 syscalls blocked
  - 70/368 ARM syscalls blocked

- **Applications can register additional filters to strengthen sandboxing**
  - Chrome
  - Media Extractor - media decoding daemon (stagefrights)

Kinds of Applications

- **Four different kinds of applications with associated SELinux contexts**
  - Isolated
  - Untrusted
  - Privileged
  - System
- **Android Note: An Application = Java Package**

Application Contexts

SYNACKTIV

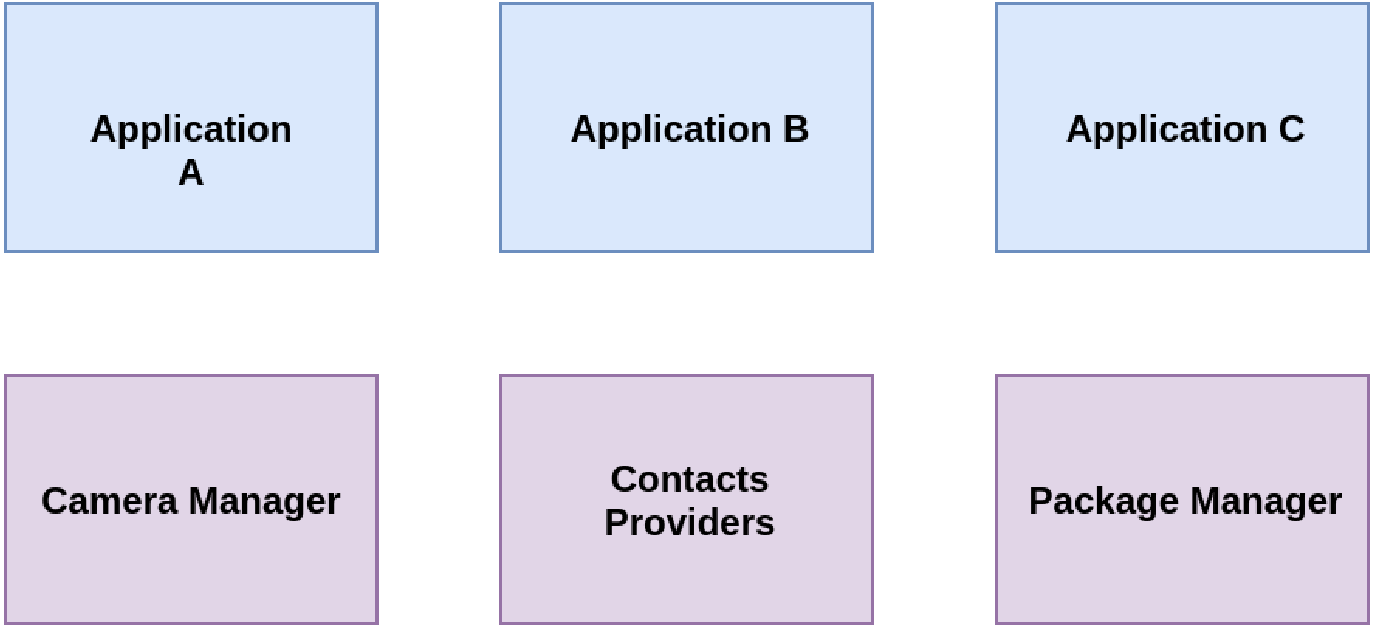
- **Isolated Applications**
  - Mainly used for Chrome renderer processes
  - The most restricted isolation
  - Isolation: context=***isolated_app*** and u0_i<uid> (90000 → 99999)
    - Different uid per isolated processus
- **Untrusted Applications**
  - All third-party applications installed by the user
  - Isolation: context=***untrusted_app*** and u0_a<uid> (10000 → 19999)

Application Contexts

SYNACKTIV

- **Privileged Applications**
  - Applications in the firmware or signed by the vendor
  - Bypass most Android services permission checks
  - Isolation: context=*priv_app*/*platform_app* and uid=u0_a<uid>
- **System Applications**
  - Highest privileged applications running as system
  - Signed by the vendor
  - Isolation: context=*system_app* and uid=system (1000)

Android isolates processes …



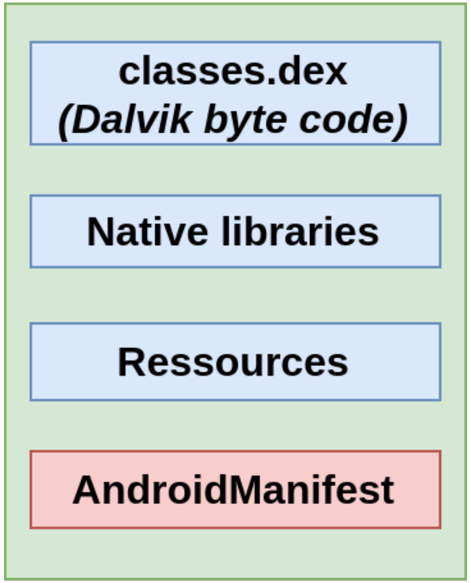
But the system needs to do things… It needs interactions !

Android Permissions Security Model

Android Application

SYNACKTIV

- **Applications are packaged in an APK archive**
- **Their behavior is described in the AndroidManifest.xml**
  - General information (name, version, icon)
  - Components exposed to the system
  - <u>Permissions requested</u>



Android APK

Permissions in the AndroidManifest.xml

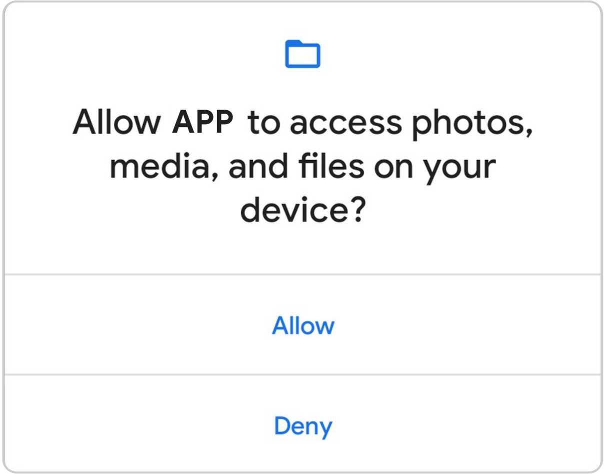
SYNACKTIV

- **Permissions example :**

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">

    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.WRITE_CONTACTS" />
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

    <application
        ...
    </application>

</manifest>
```

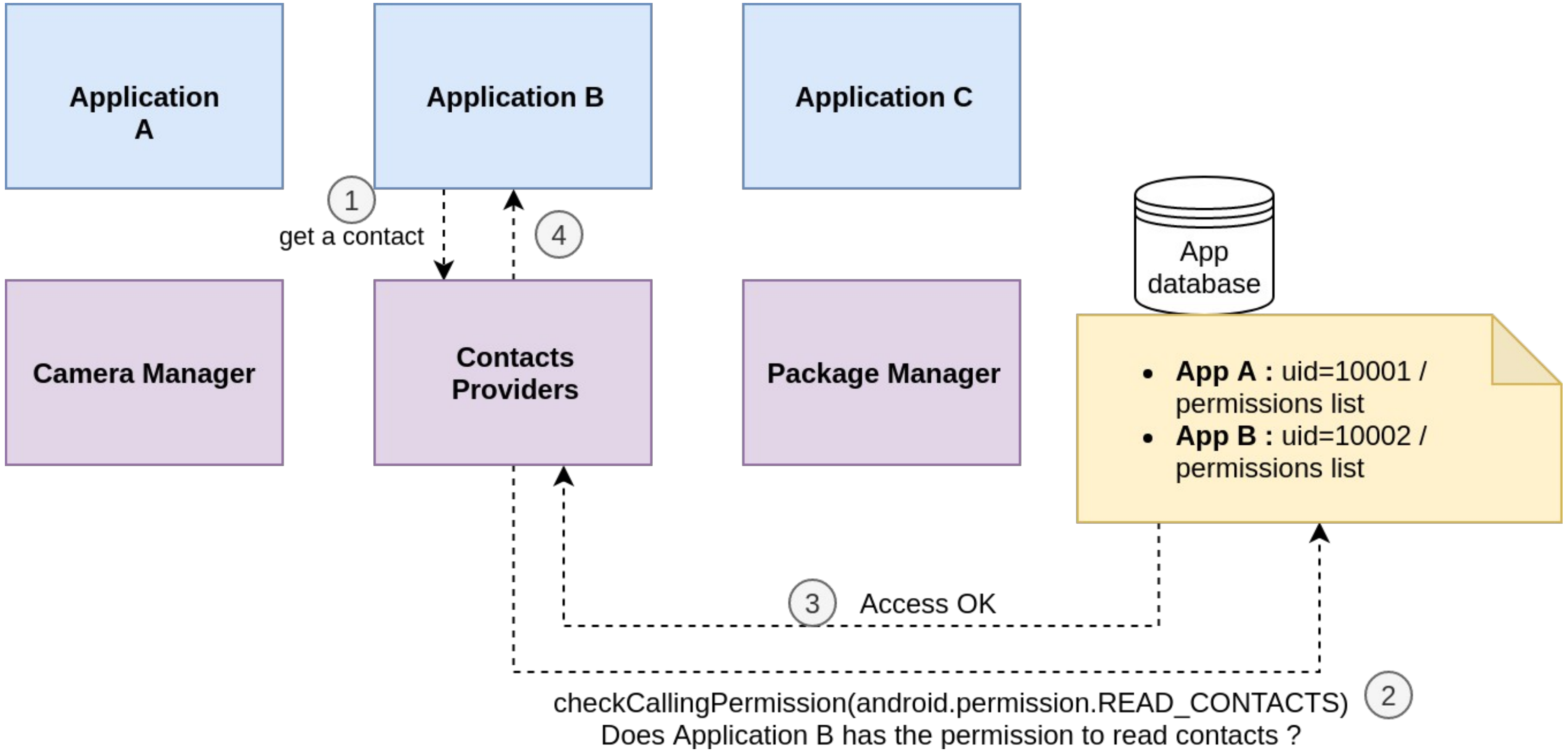
ACL with Android Permissions

- **Different types of permissions**
  - Install-time permissions
  - Runtime permissions
- **Some permissions are directly mapped to Unix Groups**
- **Others are checked at runtime during interactions with other components**
- **Provide access control to system resources and interactions with other apps**



Runtime permission

ACL in Interactions



Hardening and Mitigations

Hardening and Mitigations

SYNACKTIV

- Even with robust isolation, there is still some attack surface
- This surface must be hardened to limit and make LPE more difficult

Hardened components

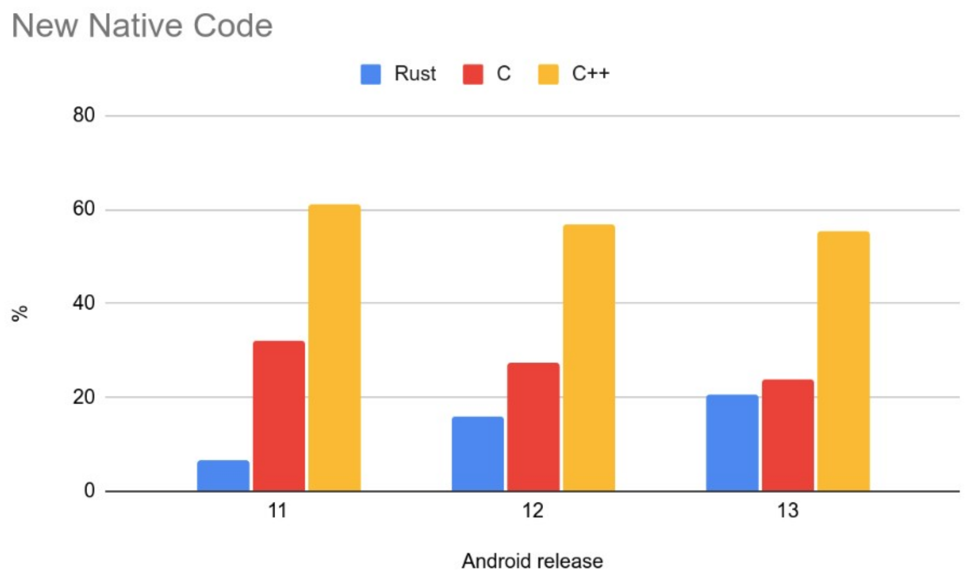
- **Some components have strong restrictions**

  → Reduce the attack surface of exposed component
- **Media Extractor (ex mediaserver)**
  - Specific SECCOMP rules
    - Allow ~ 34/271 syscalls ARM64 and ~42/364 syscalls ARM
- **Sandbox Chrome/Webview**
  - Very limited view of FS + Only 3 services accessible
  - Strong sandbox with SECCOMP

Hardened components

SYNACKTIV

- **More and more Rust in Android**
  - Bluetooth stack
  - Keystore2
  - Ultra-wideband stack
  - DNS-over-HTTP/3



https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html

Mitigations

SYNACKTIV

- **Against remote exploitation**
  - ASLR - Address Space Layout Randomization
  - PIE - Position Independent Executable
- **Scudo Heap allocator (Android 11)**
  - Designed for security
  - Detects allocation corruptions
  - Detects double-free

Mitigations

SYNACKTIV

- **CFI - Control Flow Integrity**
  - Prevents an attacker from altering the execution flow
  - Added at built time for specific binaries
  - Enabled in all media parsers since Android 8.1
  - Enabled in the Kernel since Android 9

Mitigations

SYNACKTIV

- **Compiler added checks:**
  - UndefinedBehaviorSanitizer: integer overflow, misaligned addresses
  - BoundsSanitizer: check array access
  - ShadowCallStack: protect the return address
- **Process aborts if a sanitizer check is triggered**
  - Prevent attackers from exploiting vulnerabilities

Conclusion

SYNACKTIV

- **Each Android release improves the OS security**
  - Enhanced isolation
  - Improved mitigation
- **Even if there are vulnerabilities**
  - Difficult to exploit them
  - Some bugs are now non-exploitable
  - Highly privileged components remain constrained

SYNACKTIV

https://www.linkedin.com/company/synacktiv

https://twitter.com/synacktiv

https://synacktiv.com