ITMC411
Security in mobile computing

LECTURE 6

Analyzing Android Applications

The Security Model

- No app should be **able to access** another app's data without **authorization**
  - **also** not be **able to affect** the **operation** of the other application adversely or without the appropriate consent.
- Open and extensible environment
- Android must know who created an app.
  - At least to know whether Google made it or not.

Code Signing

Digital Certificates

- Public-key cryptography
- Private key held only by app developer
- Generate key with **keytool**
- Sign app with **jarsigner**
- Signature in **META-INF** directory

Certificate Validation

- Android does not verify the **certificate** in any way
- Certificates don't need to come from a trusted **Certificate Authority**
- Most are **self-signed**
- Certificate **checked** only when app is **installed** **"security profiles"**

Signing Vulnerabilities

- **Master Key**
- "**Extra**" Field Length
- "**Name**" Field Length

Master Key

- Found in 2013 by **BlueBox** Security
- If **two files** are in the **APK archive** with the same **filenames** occurred in the **zip archive**
  - Only the **first file's hash is checked**
  - But the **second file** is actually deployed to the device
- **Arbitrary** code execution possible

For more information:

https://github.com/Fuzion24/AndroidZipArbitrage/

"Extra" Field Length

- Length field is a **16-bit value**
    - Java treats it as **signed**
- Can **overflow** and become **negative**
- Allows **injection** of altered files that **pass signature verification**

For more information:

http://www.saurik.com/id/18

"Name" Field Length

- **Length** **not checked** by the Java verification code
- Allows **code injection** into the **filename**
- While passing **signature validation**

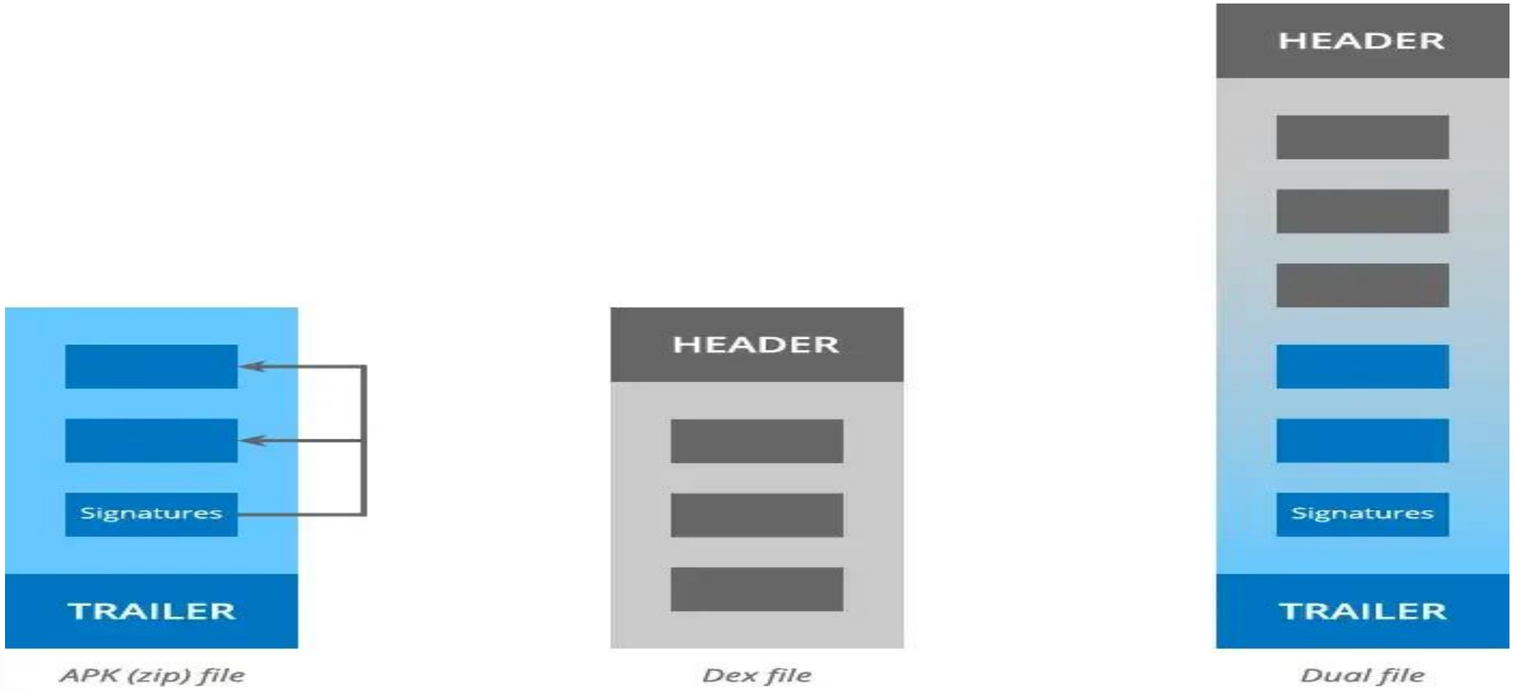
For more information:

http://www.saurik.com/id/19

Janus vulnerability

Janus vulnerability

- Janus vulnerability comes from the possibility to add **extra bytes to APK files and to DEX files.**
- discovered by **GuardSquare** in 2017 in **Android 8.0**

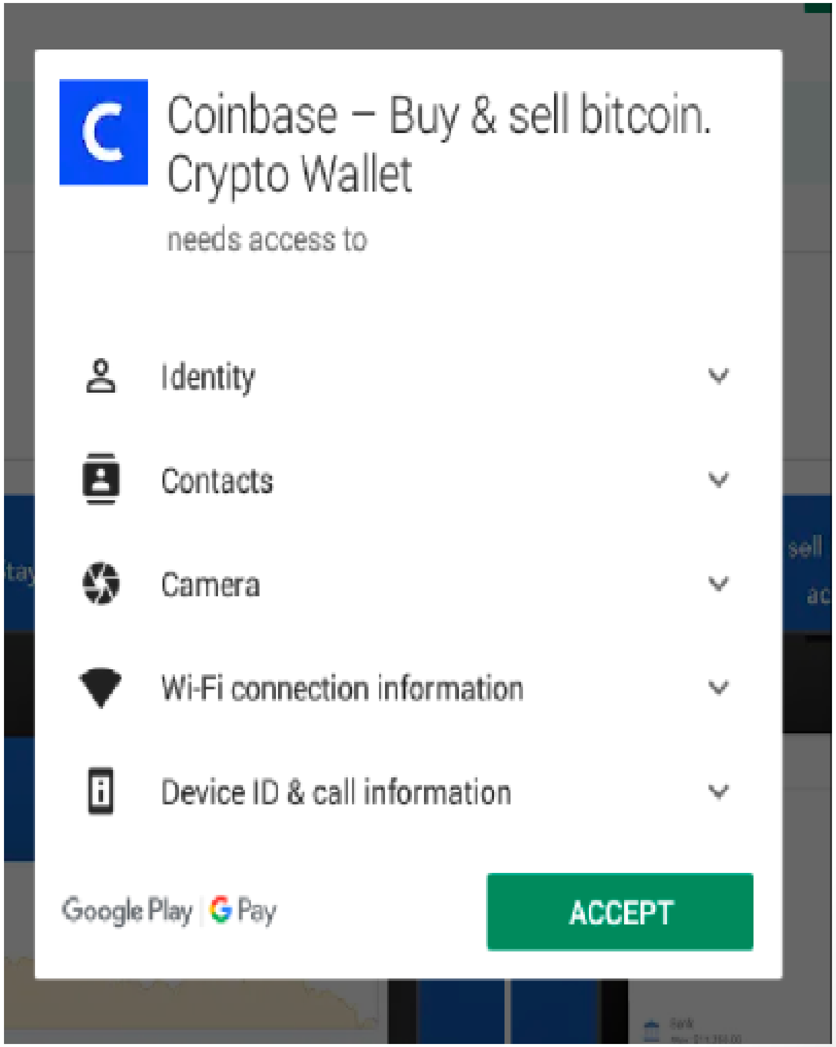


Exploiting Apps vulnerable to Janus (CVE-2017–13156)
https://medium.com/mobis3c/exploiting-apps-vulnerable-to-janus-cve-2017-13156-8d52c983b4e0

Understanding Permissions

The Android Permission Model

- Permissions shown at **install time**



Coinbase – Buy & sell bitcoin. Crypto Wallet

needs access to

Identity

Contacts

Camera

Wi-Fi connection information

Device ID & call information

Permission Protection Levels

- An app can define a **new permission**
- When it does, **a protection level** is assigned to it
- **Skype** defines this permission

```
<permissionandroid:name=
"com.skype.raider.permission.C2D_MESSAGE“
android:protectionLevel="signature"/>
```

Permission Protection Levels

PROTECTION LEVEL	DESCRIPTION
normal	The **default value for a permission**. Any application may request a permission with this protection level.
dangerous	Indicates that this permission has the **ability to access** some potentially **sensitive information** or **perform actions on the device**. Any application may request a permission with this protection level.
signature	Indicates that this permission **can only be granted to another application that was signed with the same certificate** as the application that defined the permission.
signatureOrSystem	This is the same as the **signature** protection level, except that the permission **can also be granted to an application that came with the Android system image or any other application that is installed on the /system partition.**
system	This permission **can only be granted to** an application that came with the **Android system image or any other application** that is installed in particular folders on the **/system** partition.
development	This permission **can be granted from a privileged context to an application at runtime**

"Signature" Protection

- **Recommended** for apps that don't intend to **share** data or functionality with apps from other developers
- No other apps can access your **app's components**

This field was deprecated in API level 28.
*use **signingInfo** instead*

Malicious Apps

- Can just ask for **permissions** and hope the user allows it (**social engineering**)
- Or include **a kernel exploit** to gain **root**, such as **Gingerbreak**

Application Sandbox

Data Folder Permissions

- Each app runs as its **own user**
- Unless it requests to run as **sharedUserId** and has the same **signature as another app**
- Some apps allow **world-execute**
  - means that any other **files** or **subfolders** inside this directory with **lax permissions** set on them will result in the exposure of these files to any user (and hence application) on the system.

Sandbox Limitations

- Not a separate virtual machine for each app.
- Only Linux user and group permissions.

Filesystem Encryption

"Full Disk Encryption"

- **Prevents** data theft from a **stolen device**
- **Available** since Android **v.3.0**
- Not enabled **by default** in versions prior to **5.0**
- **Encrypts** with **AES-CBC**, a strong algorithm
- **FDE** is going away, replaced by **file-based encryption**.

File-based Encryption

Android Version	FBE Support
Android 7.0 (Nougat)	Yes
Android 8.0 (Oreo)	Yes
Android 9.0 (Pie)	Yes
Android 10 (Q)	Yes
Android 11 (R)	Yes
Android 12 (S)	Yes
Android 13 (T)	Yes

Encryption Limitations

- **SD** card **not encrypted**.
- Only **protects** data **at rest**.
- If attacker can execute code on the device, **encryption does nothing**.

Generic Exploit Mitigation Protections

Exploit Mitigations

- Make the **underlying OS** more secure.
- So even unpatched legacy code is safer.
- Many of these **mitigations** are inherited from **Linux kernel**.

Exploit Mitigations

EXPLOIT MITIGATION	VERSION INTRODUCED	EXPLANATION
Stack cookies	1.5	Protects **against basic stack-based overflows** by including a "**canary**" value after the stack that is checked.
safe_iop	1.5	Provides a library that helps **reduce integer overflows**.
dlmalloc extensions	1.5	Helps prevent double free() vulnerabilities and other common ways to **exploit heap corruptions**.
calloc extensions	1.5	Helps **prevent integer overflows** during memory allocations.
Format string protections	2.3	Helps **prevent** the exploitation of **format string vulnerabilities**.
NX (No eXecute)	2.3	**Prevents code** from **running on the stack or heap**.
Partial ASLR (Address Space Layout Randomization)	4.0	Randomizes the location of libraries and other memory segments in an attempt to defeat a common exploitation technique called ROP (Return- Oriented Programming).

Exploit Mitigations

PIE (Position Independent Executable) support	**4.1**	Supports **ASLR** to **ensure all memory components are fully randomized**. Effectively ensures that **app_process** and linker are randomized in memory so that these cannot be used as a source of **ROP gadgets**.
RELRO (RELocation Read-Only) and **BIND_NOW**	**4.1**	Hardens data sections inside a process by making them **read-only**. This prevents common exploitation techniques such as **GOT** (Global Offset Table) overwrites.
FORTIFY_SOURCE (Level 1)	**4.2**	Replaces common **C functions** that are known to cause security problems with "**fortified**" versions that **stop memory corruption** from taking place.

Exploit Mitigations

SELinux (Permissive mode)	**4.3**	in which permission denials are **logged** but **not enforced**.
SELinux (Enforcing mode)	**4.4**	in which permissions denials are both **logged** and **enforced**
FORTIFY_SOURCE (Level 2)	**4.4**	Replaces additional functions with their “**fortified**” versions.

Kernel Protections

EXPLOIT MIGITATION	VERSION INTRODUCED	EXPLANATION
Removed **setuid/setguid** programs	4.3	Removed all **setuid/setgid** programs and added support for **filesystem capabilities** instead.
Restrict **setuid** from installed apps	4.3	The **/system** partition is mounted as **nosuid** for all processes that were spawned by **zygote**. This means that installed applications cannot abuse vulnerabilities in any **SUID** binaries to gain root access.

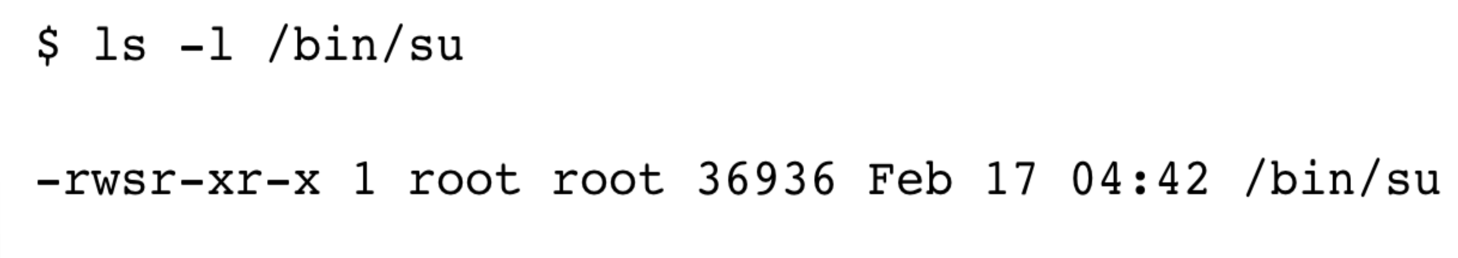
Rooting Explained

Root Access

- By default Android doesn't allow users to use **root**
- **Rooting** typically adds a **su** binary
  - Allows elevation to **root**
  - So **su** itself must run as **root**

SUID Permissions

- Runs with **owner's permissions**
- Even when launched by someone else

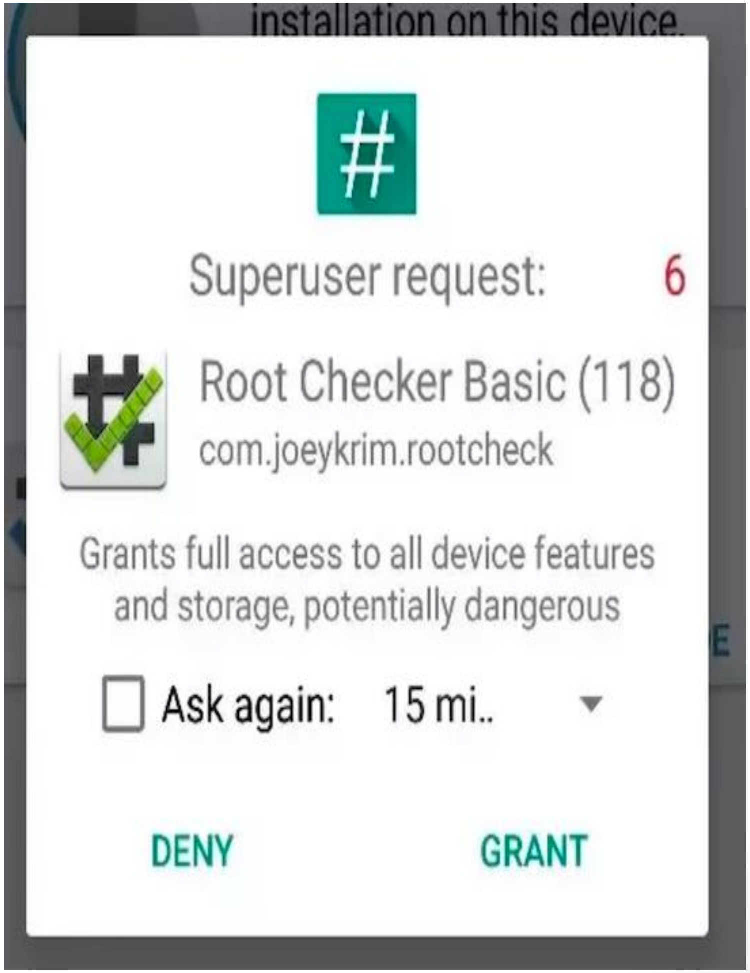


```
$ ls -l /bin/su

-rwsr-xr-x 1 root root 36936 Feb 17 04:42 /bin/su
```

Security of su

- On Linux, it asks for a **password** to allow elevation
- On Android, it pops up a box like this



installation on this device.

Superuser request: 6

Root Checker Basic (118)

com.joeykrim.rootcheck

Grants full access to all device features and storage, potentially dangerous

□ Ask again: 15 mi..

Rooting Methods

- **Two** main ways of gaining **root access** on an Android device
  - Using an **exploit**
  - Using an **unlocked bootloader**

Exploits

- **Gingerbreak** (**EXPLOITING AOSP KERNEL CODE**)
  - Exploited **vold** to write to the **Global Offset Table (GOT)** in Android 2.2 and 3.0
  - Bug in Google's original Android Open Source Project (AOSP) code from Google.
- **Exynos abuse** **(EXPLOITING CUSTOM DRIVERS)**
  - Bug in **driver** for **exynos processors**, used by Samsung
  - Only **affected** some devices

Exploits

- **Samsung Admire (ABUSING FILE PERMISSIONS WITH SYMLINKS)**
  - Exploited **dump files** and **logs** to change pemissions on **adb**
  - Worked only on specific device
- **Acer Iconia (EXPLOITING SUID BINARIES)**
  - Pre-installed **SUID** binary with **code injection** vulnerability

Exploits

- **Master Key** **(EXPLOITING ANDROID AOSP SYSTEM CODE)**
  - Make a modified system app, when two files have the same name
  - Re-install it with the same signature
  - Works on most Android versions prior to 4.2
- **Towelroot** **(EXPLOITING LINUX KERNEL VULNERABILITIES ON ANDROID)**
  - Exploits locks used when threading
  - Rooted many devices

Unlocked Bootloader

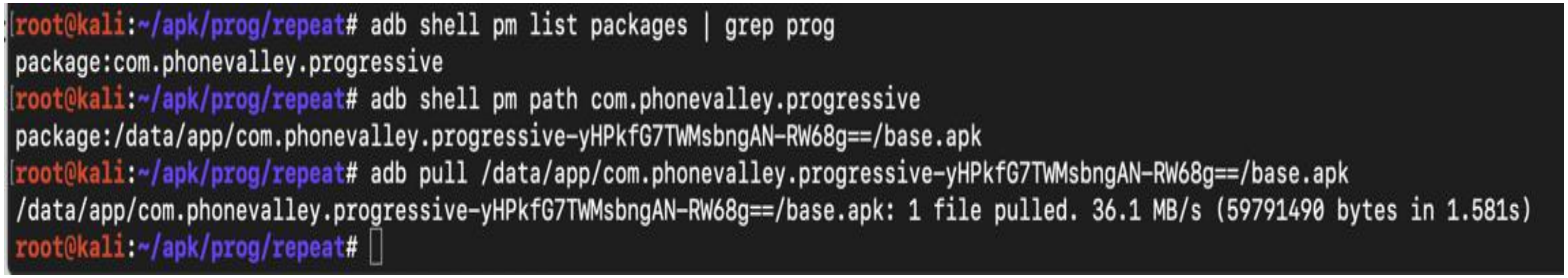
- Flash new **firmware** onto device
  - A new recovery image, **or**
  - A rooted **kernel image** containing **su**
- May **void warranty** or **brick** your phone

- **popular recovery image :**
  - **ClockWorkMod , CF-Autoroot**

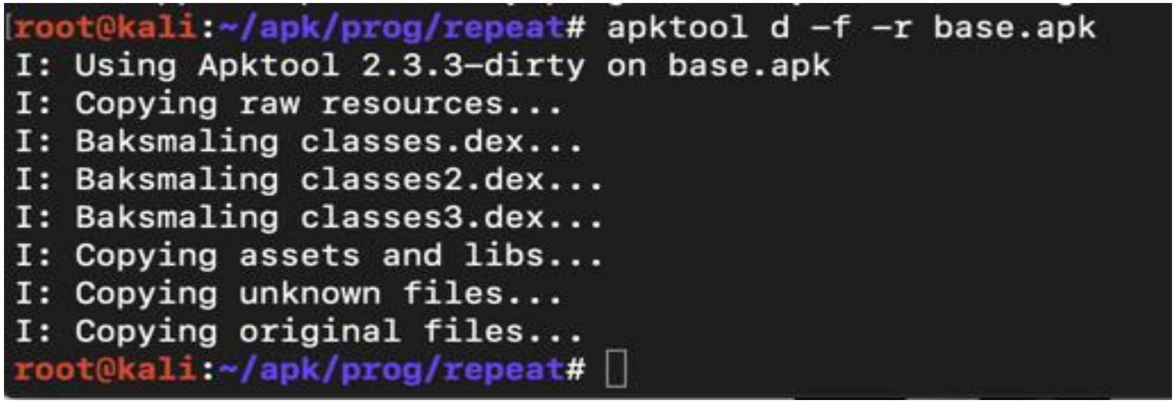
Reverse-Engineering Applications

In the Projects

- Pulling an **APK** from the phone with **adb**
- Disassemble with **apktool**



```
root@kali:~/apk/prog/repeat# adb shell pm list packages | grep prog
package:com.phonevalley.progressive
root@kali:~/apk/prog/repeat# adb shell pm path com.phonevalley.progressive
package:/data/app/com.phonevalley.progressive-yHPkfG7TWMsbngAN-RW68g==/base.apk
root@kali:~/apk/prog/repeat# adb pull /data/app/com.phonevalley.progressive-yHPkfG7TWMsbngAN-RW68g==/base.apk
/data/app/com.phonevalley.progressive-yHPkfG7TWMsbngAN-RW68g==/base.apk: 1 file pulled. 36.1 MB/s (59791490 bytes in 1.581s)
root@kali:~/apk/prog/repeat# 
```



```
root@kali:~/apk/prog/repeat# apktool d -f -r base.apk
I: Using Apktool 2.3.3-dirty on base.apk
I: Copying raw resources...
I: Baksmaling classes.dex...
I: Baksmaling classes2.dex...
I: Baksmaling classes3.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
root@kali:~/apk/prog/repeat# 
```