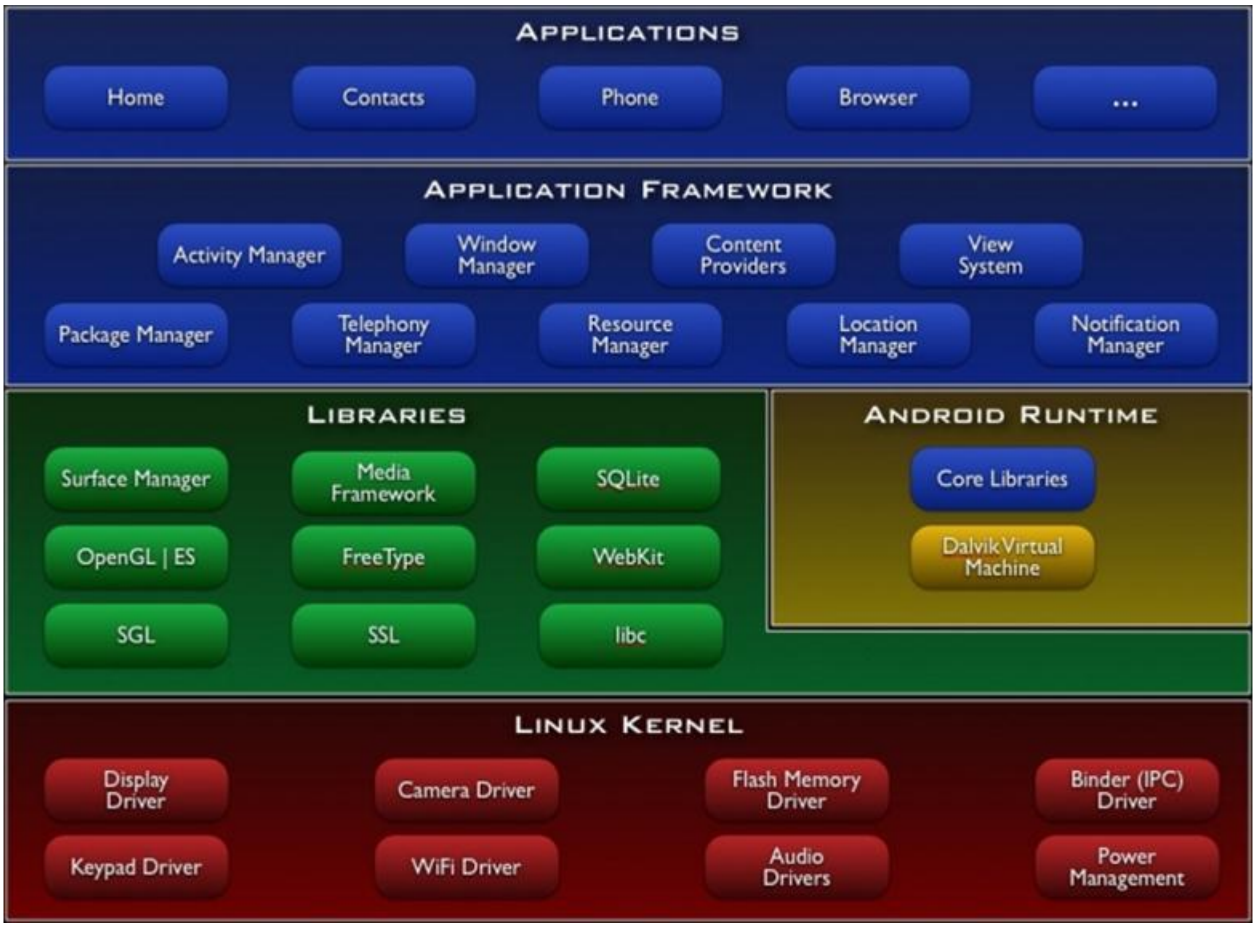
Android OS:

Android is a mobile OS designed primarily for touchscreens. It is free and open-source software, its source code known as Android Open-Source Project (AOSP) which is licensed under Apache License. the AOSP is used to develop many electronic devices.
Android is based on a modified version of Linux Kernel (UNIX), which means that its file structure is similar to Linux file structure.

Android File Structure:

As explained before, since Android is UNIX based, the file structure is similar to Linux, System partitions and files are protected and cannot be accessed unless the device is rooted by the user.
The Android architecture is basically consisting of a stack of different layers, each layer consisting of various programming components and each layer provides services to the layers above it.



Basically, Android has 5 different layers: (Bottom to Top)
- Linux Kernel
- Libraries (Native C/C++)
- Android Runtime
- Application Framework (Java API Framework)
- Applications

Linux Kernel:

This is the heart of the Android architecture. It manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc.
The Linux kernel also acts as abstraction layer between the device hardware and the rest of the software stack. It is responsible for management of memory, power, process, etc.
Since Linux Kernel is the heart of Android operating system, it provides Android with several key security features such as:
- User based permissions model.
- Process isolation.
- Extensible mechanism for secure IPC.
- The ability to remove unnecessary and insecure parts of the kernel.

What Linux security models are applied on Android?

1- Linux Security: Linux security is based on the concept of users and groups. It assigns a unique user-id (UID) whenever a new user is created and these users can be added to a group which has a unique group-id (GID), which is used to distinguish between other groups. Each file in Linux has the UID of the particular user assigned to it, and only this user has the responsibility for the file and can alter the permissions on it. same thing on Android it assigns UID to each Android application and runs it in its own process.

2- Application Sandbox: Android uses the UID to set up a kernel-level Application Sandbox. The kernel enforces security between apps and the system at the process level through standard Linux facilities such as user and group IDs that are assigned to apps. By default, apps can't interact with each other and have limited access to the OS. If app X tries to do something malicious, such as read application Y's data or dial the phone without permission, it's prevented from doing so because it doesn't have the appropriate default user privileges. All the layers above the kernel run within the Application Sandbox.

3- System Partition and Safe Mode: system partitions (all layers mentioned above) is set to read-only. when booting the device in safe mode, third-party application may be launched manually by the device owner but not launched by default.

4- Security Enhanced Linux (SELinux): Android uses SELinux to enforce mandatory access control (MAC) over all processes even those who are running under root / superuser privileges, which means limiting access to resources based on the sensitivity of the information that the resource contains and the authorization of the user to access information with that level of sensitivity. With SELinux, Android can better protect and confine system services, control access to application data and system logs, reduce the effects of malicious software, and protect users from potential flaws in code on mobile devices.

5- Verified boot (Android 6.0 and later): verified boot guarantees the integrity of the device software by cryptographically verifying all executable code and data that is part of the Android version being booted before it is used. It establishes a full chain of trust starting from a hardware-protected root of trust to the bootloader, to the boot partition and other verified partitions, during the device boot, each stage verifies the integrity and authenticity of the next stage before handing over execution.
In addition, it uses rollback protection to ensure that the device is running a safe version of Android, and ensures that the device only update to a newer version of Android.

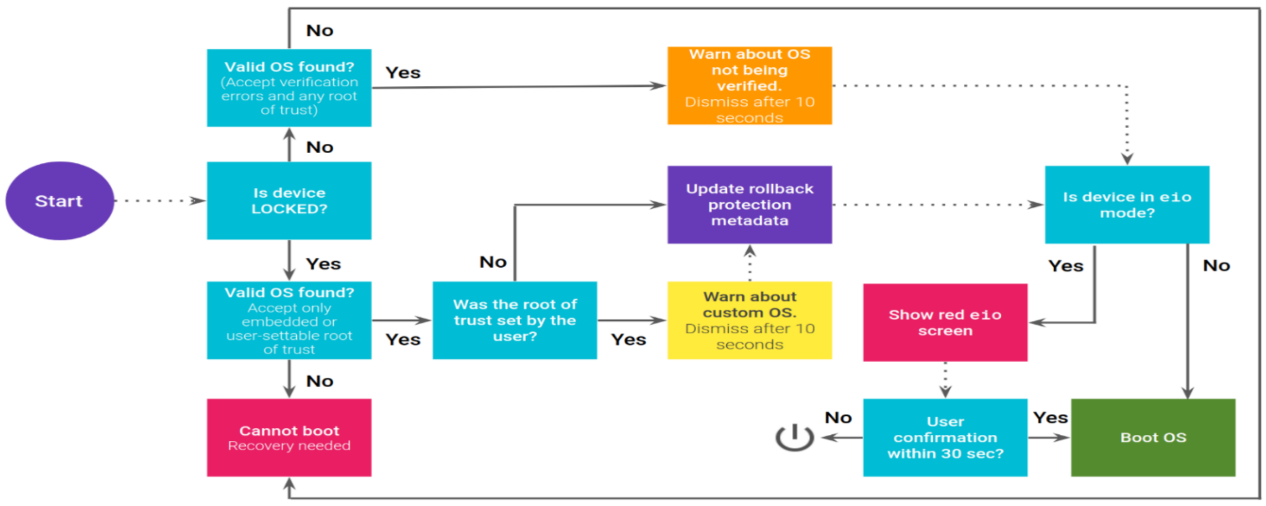
6- Cryptography: Android is provided with a set of cryptographic APIs for use by applications These include implementations of standard and commonly used cryptographic such as AES, RSA, DSA, and SHA. Additionally, APIs are provided for higher level protocols such as SSL and HTTPS.

7- Device Rooting: by default, only kernel and a small subset of core applications run with root permissions, any user or application with root permission can modify the operating system, kernel and all other applications, which means that root have access to all application and data.

8- Encryption: Android enable users to encode all user data on the device using symmetric keys. Once device is encrypted, all user-created data is automatically encrypted before committing to disk, then the system automatically decrypt data before returning it to the calling process.
Android have to encryption methods:
-File-based encrypting (FBE), (Android 7.0 and later): File-based encryption allows different files to be encrypted with different keys that can be unlocked independently. Devices that support file-based encryption can also support Direct Boot, which allows encrypted devices to boot straight to the lock screen, thus enabling quick access to important device features like accessibility services and alarms.
-Metadata encryption, (Android 9.0 and later): When FBE is used, other information, such as directory layouts, file sizes, permissions, etc. is not encrypted, this other information is known as filesystem metadata. with this encryption a single key present at boot time encrypts whatever content is not encrypted by FBE. This key is protected by Keymaster, which in turn is protected by verified boot.
-Full-disk encryption (FDE), (Android 5.0 – 9.0): encrypts all device data using a single key that is protected by the user's device password. Which means that there is no access to any part of the disk and core functionality of the device Because it is protected behind their single user credential, features like alarms could not operate, accessibility services will be unavailable, and phones could not receive calls.

9- Device Administration: Android device Administration supports enterprise apps, which provides device administration features at the system level. These APIs allow the creation of security-aware apps that are useful in enterprise settings, in which IT professionals require rich control over employee devices.

Android Boot Flow:

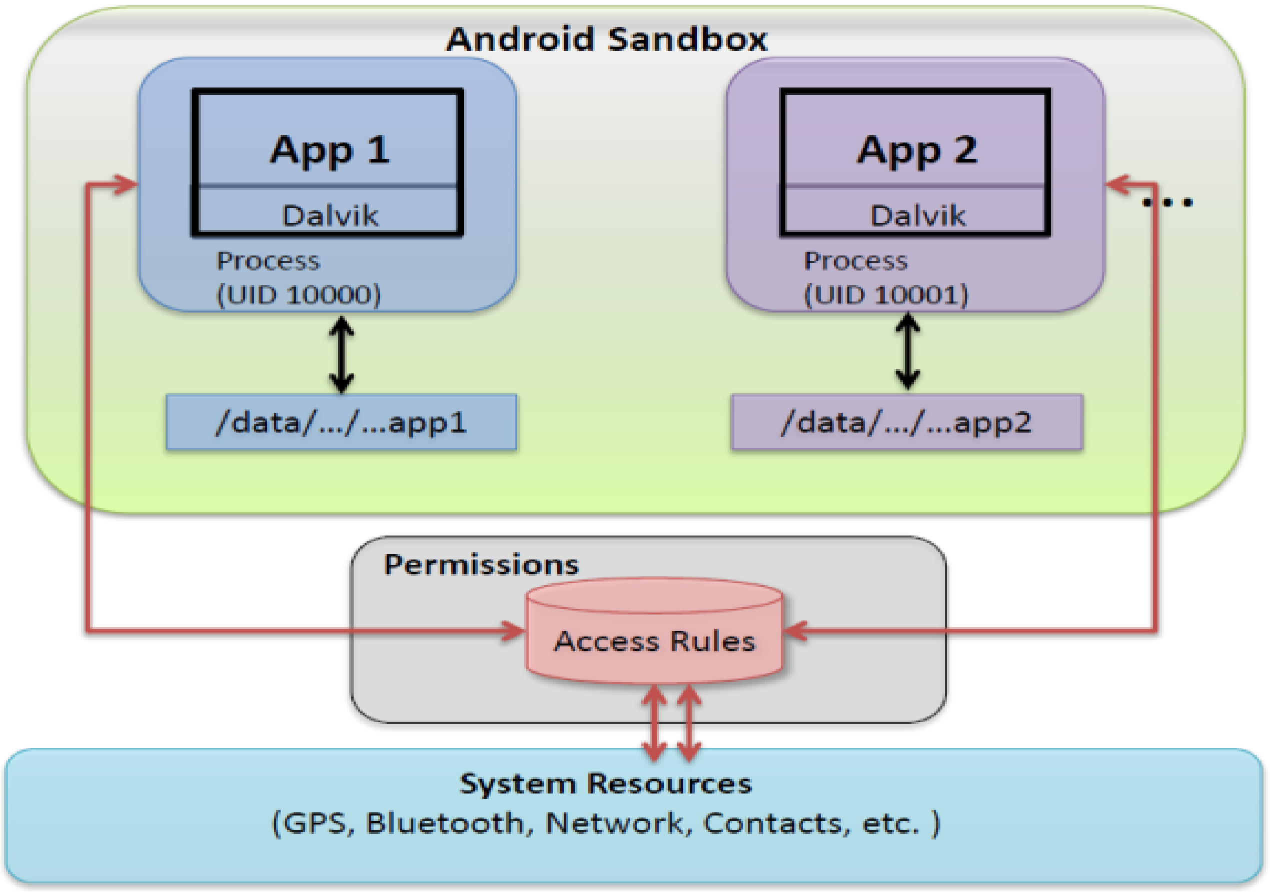

Start
Is device LOCKED?
No
Valid OS found? (Accept verification errors and any root of trust)
No
Yes
Warn about OS not being verified. Dismiss after 10 seconds
Yes
Valid OS found? Accept only embedded or user-settable root of trust
No
Cannot boot Recovery needed
Yes
Was the root of trust set by the user?
No
Update rollback protection metadata
Yes
Warn about custom OS. Dismiss after 10 seconds
Is device in eio mode?
Yes
No
Show red eio screen
User confirmation within 30 sec?
No
Yes
Boot OS

Libraries:

Libraries are layer above Linux kernel, they are called 'native libraries' and written in C/C++ languages and java. The libraires are used by various components of the Android system, they also run as processes within the underlying Linux kernel.
Those libraries are nothing but a set of instructions that tell the device how to handle different kinds of data.
Important key libraries:
- SQLite: a lightweight but a powerful relational database engine available for all applications to store data (useful to check for any sensitive data storage in the database).
- WebKit: this is the browser engine providing tools for browsing web pages (useful to check if any sensitive pages are getting cached).
- Surface: Manager: responsible for the graphics on the device screen.
- OpenGL and SGL: both cross-language, cross-platform application program interface (API) used for 2D and 3D computer graphics.
- Media: provides support to play and record an audio and video formats.
- SSL: provides encrypted link between web-server and web-browser.

Android RunTime (ART) & Dalvik Virtual Machine (DVK):

this is located on the same layer as the libraries layer. It consists of the core Java libraries and the Dalvik virtual machine (DVM). The core java libraries are used for developing Android based applications.
The DVM has been designed to run multiple VMs efficiently, and are used to run each application as its own process. DVM helps for better memory management, applications need permissions in order to interact one with another, threading support



What are the differences between Dalvik and ART?

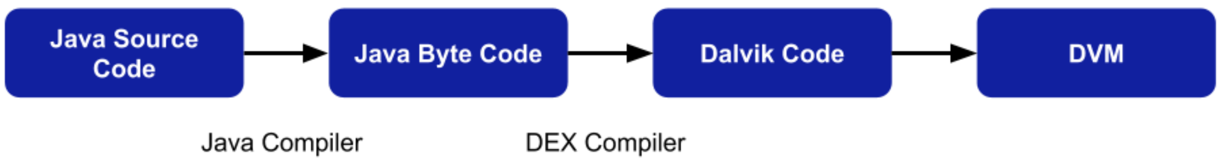
in Android, the Java classes will be converted into DEX bytecode and the DEX bytecode format is translated to Machine code with the help of ART or Dalvik runtimes. Then the Android will run the machine code.

What is a runtime?

It is possible to think of the runtime as a software that is used to convert High-Level-Language in to Machine level code.

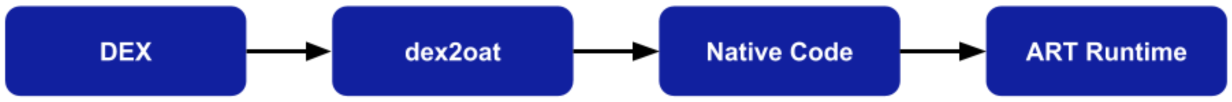
What is Dalvik?

Dalvik is Just In Time (JIT) compiler, which means whenever running an app the code that is needed for execution of the app will only be compiled at that moment and rest of the code will be compiled in the future when needed. Android itself is a Linux system with Dalvik sitting on top of it. DVM takes android app, turns them from java code into bytecode that the Linux system can run.



What is ART?

Android RunTime uses Ahead Of Time (AOT), by using AOT it converts or compiles the whole High-Level-Language code into machine code during installation of the app using dex2oat tool and not dynamically as the application runs (like in case of Dalvik).



Application Framework:

the next layer above the libraries, it includes the programs that manage the basic functions of the phone like resource allocation, voice call management, etc.
developers use these framework APIs to develop further complex applications.
Some of the important blocks of this framework are:
- resource manager: handles the resource management, providing access to non-code resources such as localized strings, graphics, and layout files.
- location manager: handles the location-based services like maps and GPS.
- activity manager: manages the activity of the application life cycle.
- telephone manager: manages voice calls.
- content provider: manages data sharing between apps.

Applications:

Applications is the top layer of android architecture. The pre-installed applications like home, contacts, camera, gallery etc. and third-party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.
this layer runs within the Android run time with the help of the classes and services provided by the application framework.

Android Partitions:

Android devices include several partitions that serve different functions in the boot process. the partitions are listed in /proc/partitions and their names run from ‘mmcblk0’ – ‘mmcblk18’

/boot/: this partition stored all the files related to the booting process of the device, it includes the kernel and the ramdisk. Without this partition the device will not be able to boot.

/system/: this partition basically contains the entire OS, other than the kernel and the ramdisk. This includes the Android user interface and the default system applications.

/recovery/: this is an alternative boot partition that enables booting the device into a recovery console for performing advanced recovery and maintenance operations on it. Typically used when errors occur during system bootup or when flushing a new system image. Another usage of this partition is to wipe the data partition to restore the device to factory settings and cleaning cache partition.

/data/: also called userdata, this partition contains the user’s data – this is where the contacts, messages, settings and apps installed go. Wiping this partition essentially performs a factory reset on the device.

/cache/: this is the partition where Android stores frequently accessed data and app components.

/misc/: this partition contains different system settings in form of on/off switches, these settings may include CID (Carrier or Region ID), USB configuration and certain hardware settings. This is an important partition and if it is corrupt or missing, several device functions and features wont function normally.

/sdcard/: this is the main storage area for user files and data, it also contains the application’s data and settings. this partition exists even if no external usage or SD card is installed. and this is the default partition to open as a directory when the device is plugged into the computer using USB cable. In some devices this partition can be accessed through the symlink ‘storage/emulated/0/’.

/sd-ext/: this is not a standard Android partition, it is basically an additional partition on your SD card that acts as the /data partition when used with certain ROMs that have special features called APP2SD+ or data2ext enabled. It is especially useful on devices with little internal memory allotted to the /data partition. Thus, users who want to install more programs than the internal memory allows can make this partition and use it with a custom ROM that supports this feature.

Android Package Kit (APK):

APKs are ZIP files at their core, but they must contain additional information to properly function as an APK. It can be written either in Java or Kotlin.
An APK file contains app code in the DEX file format, native libraries, resources, assets, etc. It must be digitally signed with a certificate to allow installation on an Android device.

APK Structure: as mentioned above, the APK is a compressed package. And it contains several files and directories.

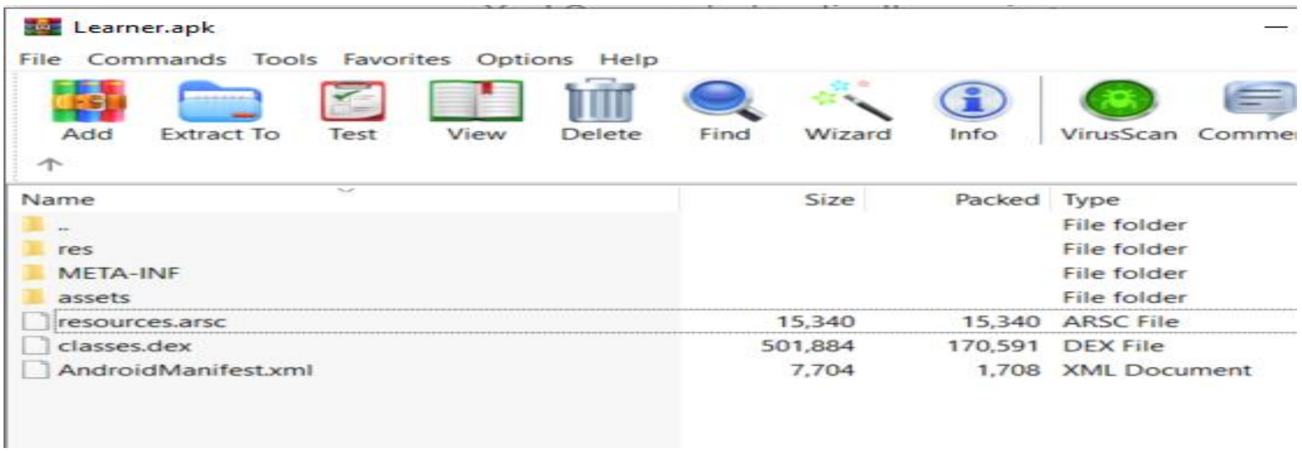
AndroidManifest.xml: this is the binary XML file format, it is a required file that describes the application and it contains the application metadata. Here's a partial list of them:
- application's package name, version, license ...etc.
- all the application components, such as activities, resources.
- what permissions this application requires to run, and the permissions required to access this application information by other apps. (Great link on how to use during pentest).

META-INF: this is a folder that contains verification information, this is what is generated when signing the application. This file is basically fingerprint information for every file contained within the APK. This means that any modification to the APK requires resigning the APK.
The contents of this folder are:
- CERT.SF: list of all files with SHA-1 hashes.
- CERT.RSA: contains signed content of CERT.RF and is used to verify app integrity with the public key.
- MAINFEST.MF: contains the SHA-256 digest of all the files and the APK. Used to verify the validity of each file in the zip.

Res: this folder contains all the resources that are not compiled to resources.arsc. it contains images, audio files, design layouts, and every non-logical code or files that the application requires.

Classes.dex: contains the code of the application. This file is understandable by DVM and ART.

Resources.arsc: a file that contains all the information (compiled in a binary format) that links the code(classes.dex) to the resources(res).

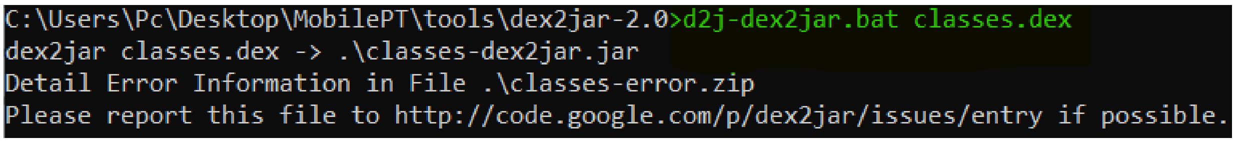

Learner.apk
File Commands Tools Favorites Options Help
Add Extract To Test View Delete Find Wizard Info VirusScan Comme

Name	Size	Packed	Type
..			File folder
res			File folder
META-INF			File folder
assets			File folder
resources.arsc	15,340	15,340	ARSC File
classes.dex	501,884	170,591	DEX File
AndroidManifest.xml	7,704	1,708	XML Document

Android Tools:

Dex2jar: (github)

The .dex files can be decompiled to readable code using dex2jar tool.
Dex2jar supports a variety of conversion operations such as JAR to DEX, DEX to SMALI.
The converting process can be done using the code in the image below



```
C:\Users\Pc\Desktop\MobilePT\tools\dex2jar-2.0>d2j-dex2jar.bat classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar
Detail Error Information in File .\classes-error.zip
Please report this file to http://code.google.com/p/dex2jar/issues/entry if possible.
```

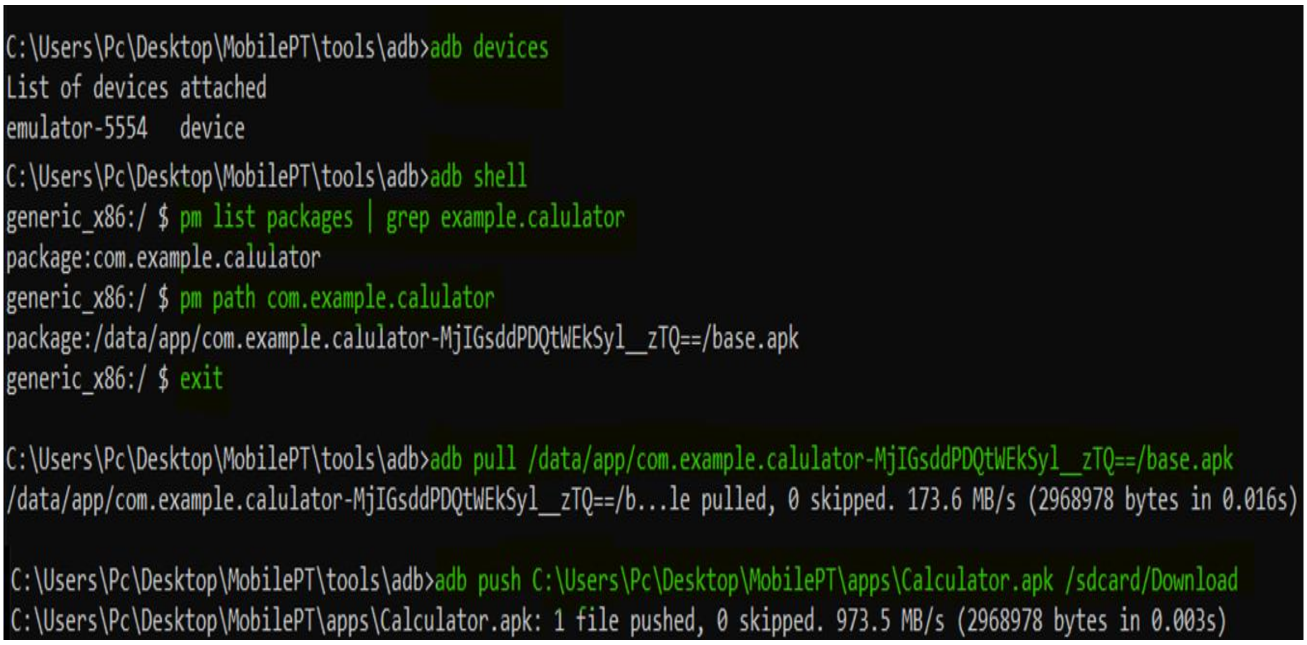
more info about dex2jar here.

JD-GUI: (download)

After converting .dex file to .jar, a compiler tool needed in order to read the source code understand it. JD-GUI enables user to read JAVA files clearly and easily.

ADB (Android Debug Bridge):

This is a command line tools that enables users to communicate with Android devices.
It also allows users to control the device from the computer and transfer files between them.
various commands can be used via ADB:
adb devices check the listed devices.
adb shell connect the user to the device
adb root connect the user to the device with root privileges
adb push [file path] [android path] file transfer, importing a file on the device.
adb pull [android path] [file path] file transfer, exporting a file from the device.
After connecting to the device using 'adb shell' it is possible to list all the packages on the device.
pm list packages | grep [package name] lists all packages available on device.
pm path [package name] getting the package path in the device.
After locating the package path, it is possible to pull the package using 'adb pull' command.



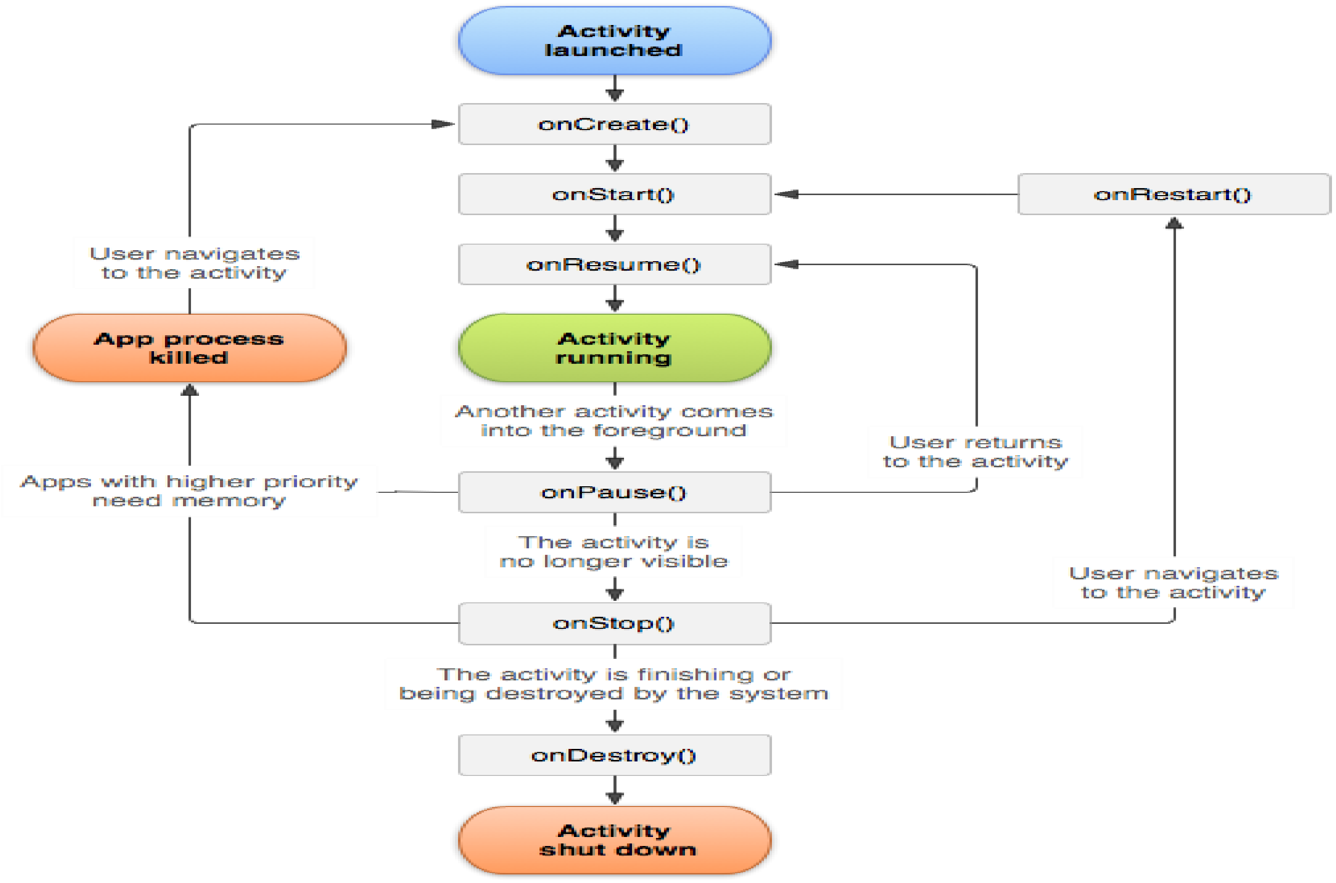
```
C:\Users\Pc\Desktop\MobilePT\tools\adb>adb devices
List of devices attached
emulator-5554   device
C:\Users\Pc\Desktop\MobilePT\tools\adb>adb shell
generic_x86:/ $ pm list packages | grep example.calulator
package:com.example.calulator
generic_x86:/ $ pm path com.example.calulator
package:/data/app/com.example.calulator-MjIGsddPDQtWEkSyl__zTQ==/base.apk
generic_x86:/ $ exit

C:\Users\Pc\Desktop\MobilePT\tools\adb>adb pull /data/app/com.example.calulator-MjIGsddPDQtWEkSyl__zTQ==/base.apk
/data/app/com.example.calulator-MjIGsddPDQtWEkSyl__zTQ==/b...le pulled, 0 skipped. 173.6 MB/s (2968978 bytes in 0.016s)

C:\Users\Pc\Desktop\MobilePT\tools\adb>adb push C:\Users\Pc\Desktop\MobilePT\apps\Calculator.apk /sdcard/Download
C:\Users\Pc\Desktop\MobilePT\apps\Calculator.apk: 1 file pushed, 0 skipped. 973.5 MB/s (2968978 bytes in 0.003s)
```

Android Activity Lifecycle:

Android Activity:
Android activity is basically every screen that a user can interact with and perform actions on it, most of times the applications contain multiple screens (activities) that users can use to navigate in the application.

Activity Lifecycle:
Android Activity Lifecycle is controlled by 7 methods and they describe how activity will behave at different states.

onCreate: this method is first called when an activity is first created. It defines what happens when an activity is created for the first time.
onStart: this is called immediately after onCreate. It will make the app visible to the user. (Defines what happens when an activity starts after an onStop event).
onResume: this is the stage where the user can interact with the application. (Defines what happens when an activity starts after an onPause).
onPause: this is called when another activity comes into the foreground (the previous activity may still be visible in the background, however the user won't be able to interact with it.
onStop: when this method is executed, user won't be able to see the activity or interact with it. (Defines what happens when an activity is no longer visible).
onDestroy: this is called when reaching the final stage of an activity. At this point the system is free to garbage collect the activity data to up space. (Defines what happens when the application is closed, or the process is killed).

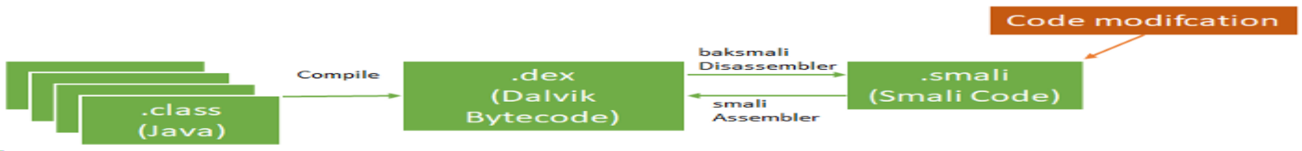

Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Android Application Reverse Engineering:

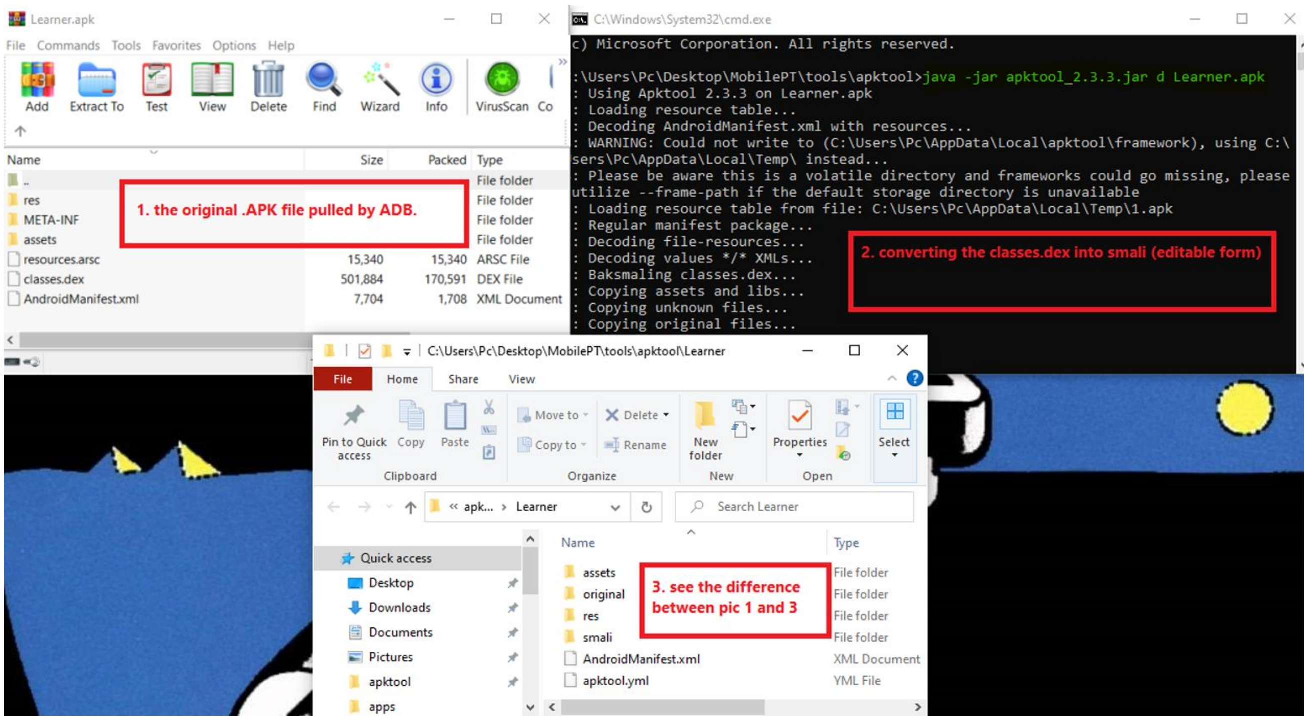
Android Applications are written in Java and compiled by the Android SDK tools into an Android package (APK), an archive file with an .apk suffix used to install the application on an android device, all the code is considered to be one application.
The Android Application is built from four different types of components:
-Activity.
-Services.
-Broadcast Receivers.
-Content Providers.
The application must declare its components in a Manifest file which must be at the root if the application project directory. Before the Android system can start an application component, the system must know that the component exists by reading this file (AndroidManifest.xml). the Manifest file also states the user permissions that the application requires. (refer to page 7 for more information about this file and the APK structure).

Application Reversing Steps:

- Decompilation: this is the first step in reversing. the Android application is extracted using ADB tool (refer to page 8 for more info) so the .apk file is decompressed into a separate folder containing the classes.dex file and other resources subfolders. Afterwards, the classes.dex (dalvik code) file is converted into readable format using apktool (website), The .dex file is then disassembled into multiple files with .smali extensions. which can be inspected and edited in order to extract/change relevant information.



NOTE: **[baksmali disassembler is used by the apktool in order to disassemble the java classes into editable forms (smali). To assemble all classes to a readable form for android (classes.dex) smali assembler is used – by apktool]


Learner.apk

Name	Size	Packed	Type
..			File folder
res			File folder
META-INF			File folder
assets			File folder
resources.arsc	15,340	15,340	ARSC File
classes.dex	501,884	170,591	DEX File
AndroidManifest.xml	7,704	1,708	XML Document

1. the original .APK file pulled by ADB.

C:\Windows\System32\cmd.exe

```
c) Microsoft Corporation. All rights reserved.

:\Users\Pc\Desktop\MobilePT\tools\apktool>java -jar apktool_2.3.3.jar d Learner.apk
: Using Apktool 2.3.3 on Learner.apk
: Loading resource table...
: Decoding AndroidManifest.xml with resources...
: WARNING: Could not write to (C:\Users\Pc\AppData\Local\apktool\framework), using C:\
sers\Pc\AppData\Local\Temp\ instead...
: Please be aware this is a volatile directory and frameworks could go missing, please
utilize --frame-path if the default storage directory is unavailable
: Loading resource table from file: C:\Users\Pc\AppData\Local\Temp\1.apk
: Regular manifest package...
: Decoding file-resources...
: Decoding values */* XMLs...
: Baksmaling classes.dex...
: Copying assets and libs...
: Copying unknown files...
: Copying original files...
```

2. converting the classes.dex into smali (editable form)

C:\Users\Pc\Desktop\MobilePT\tools\apktool\Learner

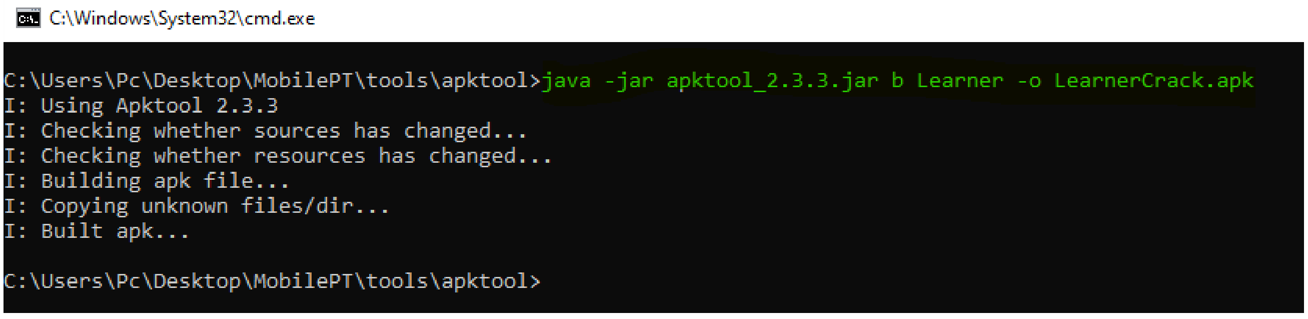
Name	Type
assets	File folder
original	File folder
res	File folder
smali	File folder
AndroidManifest.xml	XML Document
apktool.yml	YML File

3. see the difference between pic 1 and 3

- Java Code Review: the compiled java code from the APK can be decompiled back to the original Java code to reveal the original functions and variables used in the app.

- Smali Editing: since editing the Java code can be a problem (this can cause the application to crash/information loss) the modification of the code should be done at machine-code level (smali).

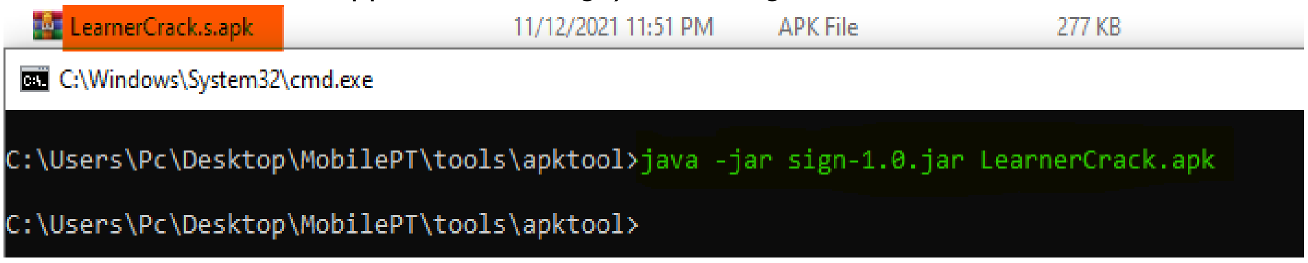
- Recompiling: after modifying the smali code, the APK must be recompiled again.



C:\Windows\System32\cmd.exe

```
C:\Users\Pc\Desktop\MobilePT\tools\apktool>java -jar apktool_2.3.3.jar b Learner -o LearnerCrack.apk
I: Using Apktool 2.3.3
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...

C:\Users\Pc\Desktop\MobilePT\tools\apktool>
```

- APK Signing: the android application won't run if it is not signed, every app must have a signature, since the app is modified the signature in no longer valid (change in the code causes the app hash to change). A new signature needs to be created.



```
LearnerCrack.s.apk    11/12/2021 11:51 PM    APK File    277 KB
```

C:\Windows\System32\cmd.exe

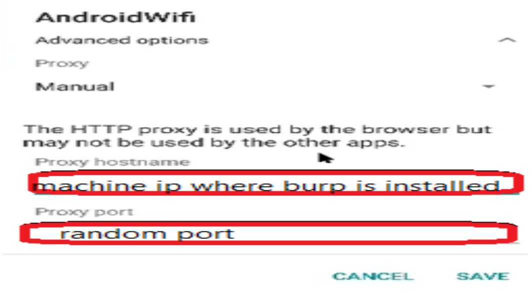
```
C:\Users\Pc\Desktop\MobilePT\tools\apktool>java -jar sign-1.0.jar LearnerCrack.apk

C:\Users\Pc\Desktop\MobilePT\tools\apktool>
```

NOTE: Another method of reverse engineering used in penetration testing is to manipulate the .xml files. (XML – Extensible Markup Language, a format used to structure data for storage and transport. In short, XML is a popular and commonly used format for sharing data).
By manipulating the .xml files of an application it is possible to bypass restrictions or affecting application activity without manipulating smali files.

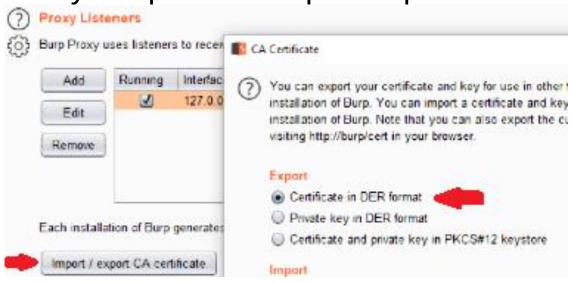
Android Traffic Analysis:

Analyzing the traffic of an application allows to understand how the app exchanges data, it is also possible to manipulate requests or change the SSL certificate to a malicious one.
A great tool for traffic analysis is Burp-Suite (download, certificate).
In order to analyze traffic on Android device several steps must be taken:

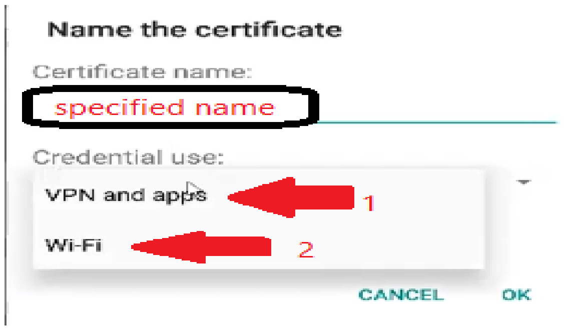
- configuring proxy on the device's Wi-Fi:
  Settings > network & internet > Modify network > setting proxy hostname and port


AndroidWifi
Advanced options
Proxy
Manual
The HTTP proxy is used by the browser but may not be used by the other apps.
Proxy hostname
machine ip where burp is installed
Proxy port
random port
CANCEL SAVE

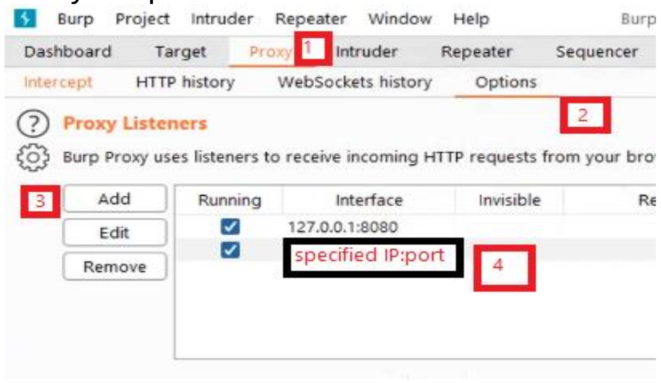
-exporting burp's certificate from the interface:
  Proxy > Options > Import/export CA certificate (Certificate in DER format)


Proxy Listeners
Burp Proxy uses listeners to recei
Add | Running | Interfac
Edit | ✓ | 127.0.0
Remove
Each installation of Burp generates
Import / export CA certificate
CA Certificate
You can export your certificate and key for use in other
installation of Burp. You can import a certificate and key
installation of Burp. Note that you can also export the c
visiting http://burp/cert in your browser.
Export
● Certificate in DER format
○ Private key in DER format
○ Certificate and private key in PKCS#12 keystore
Import

- using ADB to pull burp's certificate and installing it on device.
  The certificate must be installed both on 'VPN and apps' and 'Wi-Fi'


Name the certificate
Certificate name:
specified name
Credential use:
VPN and apps 1
Wi-Fi 2
CANCEL OK

- setting proxy listener on burp:
  Proxy > Options > add


Burp Project Intruder Repeater Window Help Burp
Dashboard Target Proxy 1 Intruder Repeater Sequencer
Intercept HTTP history WebSockets history Options
2
3
4
Proxy Listeners
Burp Proxy uses listeners to receive incoming HTTP requests from your bro

	Running	Interface	Invisible	Re
Add				
Edit	✓	127.0.0.1:8080		
Remove	✓	specified IP:port		

Cryptography

Cryptography is a description of secure communication techniques that allow only the sender and the receiver of a message to view its contents.
In this topic, I'll explain and cover all topics of Cryptography needed for the mobile security module.

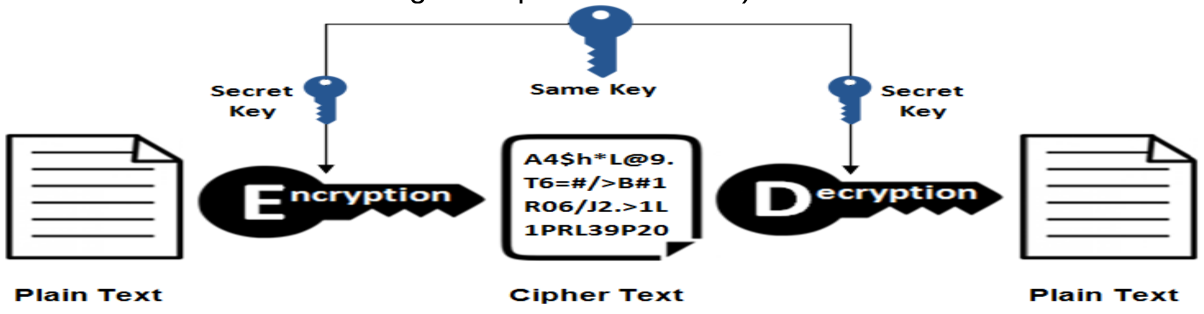
Cipher algorithm and keys:

A mathematical formula designed to obscure the value and content of data. Most cipher algorithms use a key or more in order to encrypt/decrypt the data. in order to increase the protection of the data, increasing the size of the keys is required. However, the larger the key, the more computing time is needed in order to encrypt/decrypt the data.

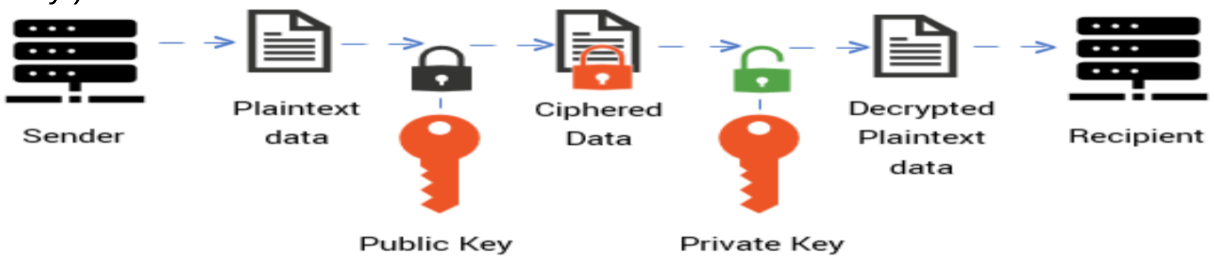
Encryption:

Is a method of transforming readable data 'plain-text' into a form that is unreadable. (data remains confidential and private).
There are two main components of encryption:

Symmetric cryptography: With symmetric cryptography the same key is used for both encryption and decryption of the data. The algorithm and key combination determine how the plain-text will be jumbled up, which is a process of substitution and transposition of those characters (if the algorithm or key are weak, then the encryption will also be weak) 256-bit AES is the bit length (the strength of the algorithm), higher number in algorithm means the stronger it is, but the slower to encrypt and decrypt. (IE: a door with many locks is more secure but it will take longer to open and close it).



Asymmetric cryptography: Asymmetric cryptography a pair of keys are involved (public & private key). The public key is published and private key is secret. The data that is encrypted with the public key can be decrypted only with the private key.
Both keys are mathematically related and the two keys are generated at the same time. Any site uses HTTPS method has a public and private key, they use to exchange asymmetric session key in order to encrypt the communication.
(if encrypt with public key, private key is required to decrypt. and if encrypted with private key, public key is required to decrypt – it's not possible to encrypt and decrypt using the same key.)

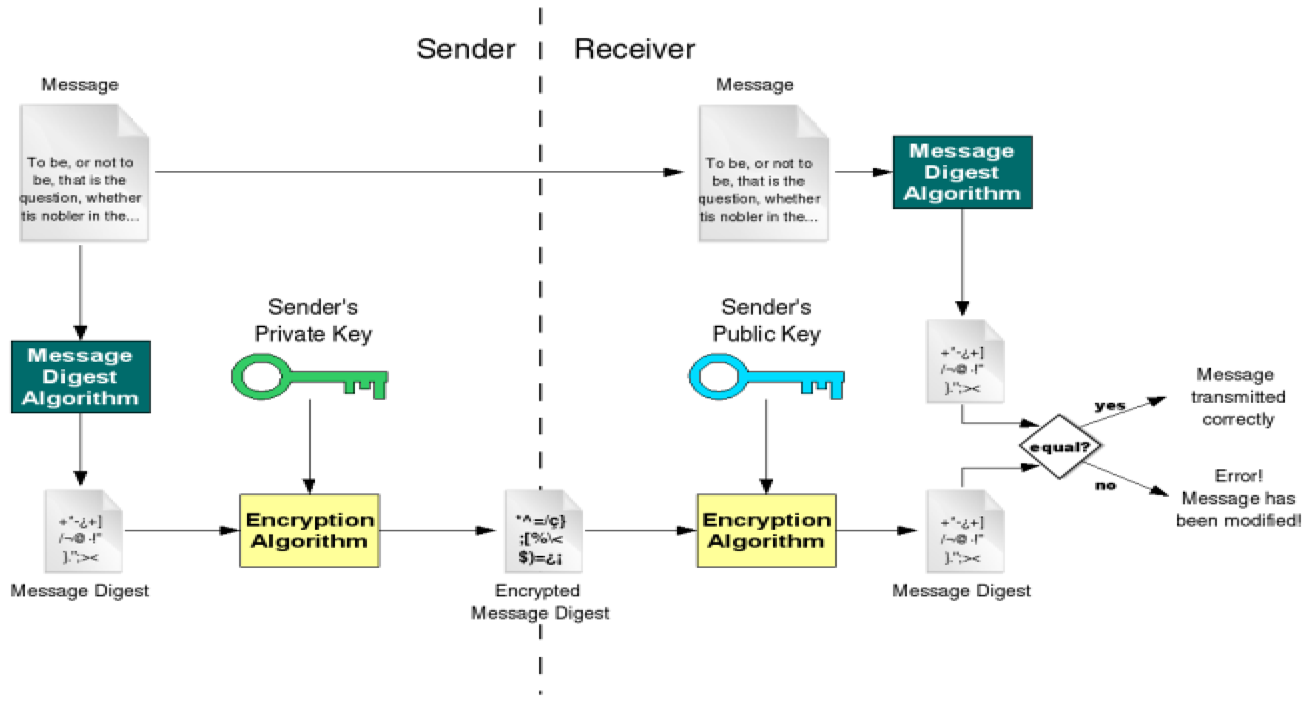


Data integrity and message digests:

In the case of data transfer, data flowing between applications in a public network environment can flows across any number of nodes or networks, encrypted data prevent these nodes/networks from understanding the data, but because loss of control on those nodes/networks, data can be altered before it reaches the destination.
Even if the data is encrypted, this may cause a harm for the applications and disturb it.
In order to exchange data probably and to ensure that it is not altered, Message Digest or Hash functions is used for this purpose.
Those functions present a fingerprint of the data, if the data changes the message digest/hash changes in ways that cannot predict.
The message digest is calculated and appended to the encrypted data, when message is received the message digest is recalculated from the encrypted data and compared to the message digest that was appended to the original message, if the values do not match, this means that the data is corrupted and will not be processed.
the hash function takes data in any size and convert it into a fixed size string of characters.
The main feature of hash function is that there is no way to convert back to the original input (one way hash function – no keys required), any change happens to the data will changes the hash value.

Digital signature:

this is used to validate the data's integrity.
The digital signature is basically a one-way hash or message digest of the original data that is encrypted with the signer's private key.
The recipient first uses the signer's public key to decrypt the digital signature, then the recipient can see information about the hash algorithm so this enables the recipient to generate a one-way hash of the same data, and then comparing the hashes (received hash and generated hash), if they match this means the data has not changed since it was signed and the recipient can assure that the public key used to decrypt the digital signature corresponds to the private key used to create the digital signature.



Authentication:

This is the process of establishing that the sender/receiver is who claims to be.
Authentication is established through public-key certificates. And must be digitally signed by a third party who vouches for the authenticity.

Digital certificate:

This is an electronic document that is used to identify an entity and associate it with a public-key. This addresses the problem of impersonation. Those certificates are issued by (CAs – certificate authority), CAs are entities that validate identities and issues certifications.
Clients and servers use certificates issued by CA to determine the other certificates that can be trusted.
The certificate issued by the CA binds a particular public key to the name of entity that the certificate identifies. And in addition to a public-key, the certificate includes the name of the entity, expiration date, CA's name that issued the certification, serial number and other information. But the important part is the certification includes the digital signature of the issuing CA.
In short, a certificate is a document that encapsulate information about the entity that owns it. (it is similar to an ID card or a driving license). It is also important to know that the structure of the certificate uses X.509 standards and it is defined by the ITU (more information here).

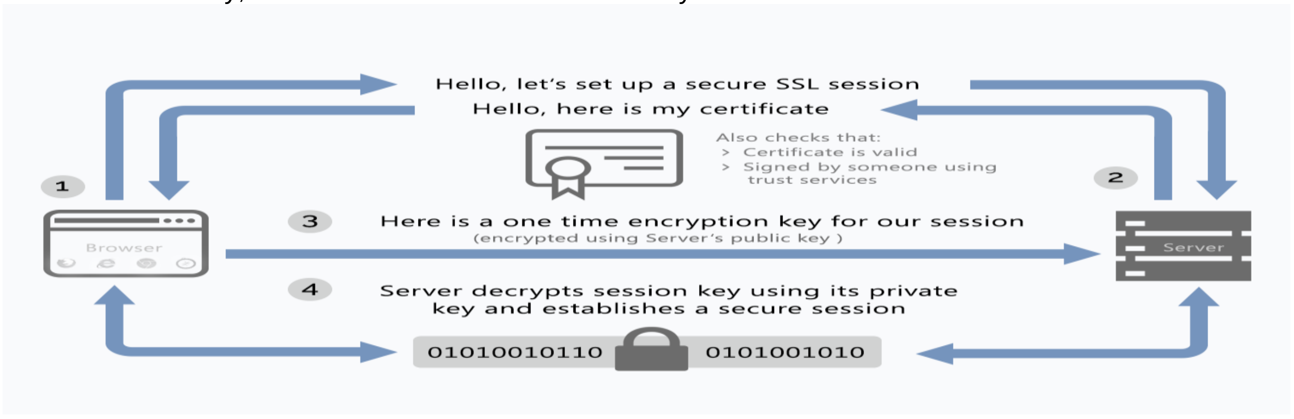
SSL (Secure Socket Layer) & TLS (Transport Layer Security):

SSL and TLS are cryptographic technologies include the symmetric and asymmetrical algorithms, hashes, digital signatures, message authentication codes...etc. to make a working security protocol. They are designed to provide communication security over the network.
SSL is positioned as a protocol layer between the (TCP) layer and the Application Layer to form a secure connection between clients and servers so that they can communicate in a secure manner over a network by providing: Privacy: where data messages are encrypted so that only the two application endpoints understand the data. Integrity: where message digests detect if any data was altered in flight. Authentication: which verifies the identity of the remote node, application, or user by using digital certificates.
SSL is the older encryption protocol and TLS is the new one (more secured).
the connection is private because a symmetric algorithm protected such as AES with asymmetric algorithm such as RSA is used to encrypt/decrypt the data transmission, the keys for the symmetric encryption are generated uniquely for each connection and based on secret negotiation at the start of the session, the server and client negotiate the details of which encryption algorithm and cryptographic keys to use before the first bite of data transmitted.

TLS handshake:

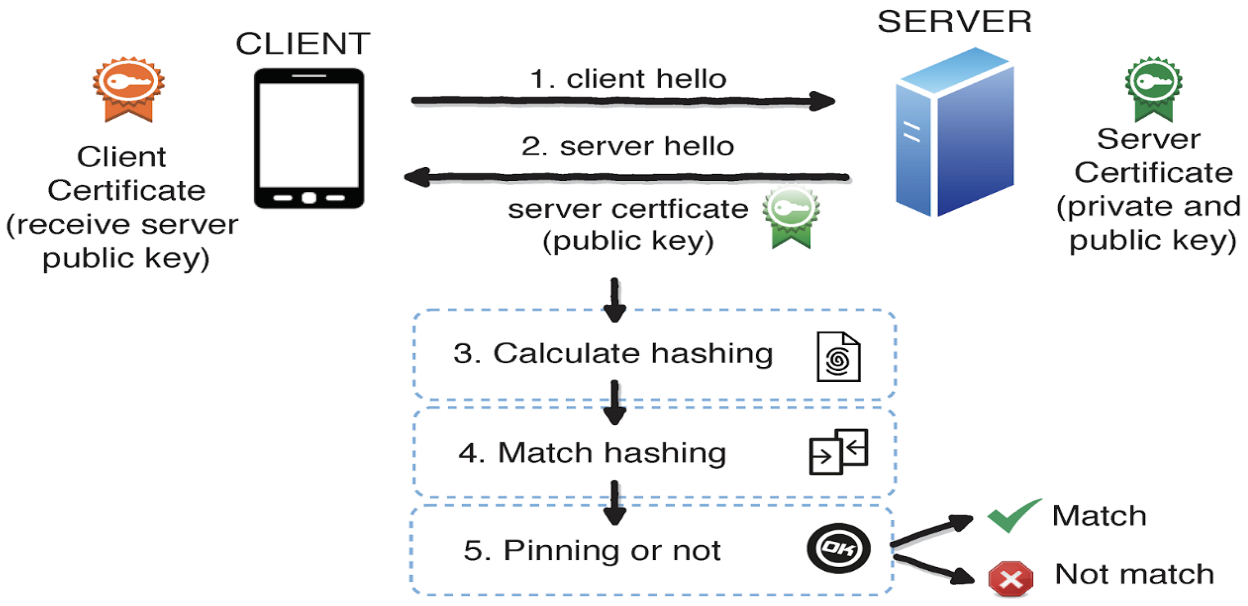
When two systems use TLS attempt to connect, each system need to verify that the other supports TLS, this process is called TLS-handshake.
In TLS-handshake both parties decide upon the TLS version, encryption algorithm, cipher suite... etc. that will be used in the procedure.
The handshake process:

- the client requests the server to open a secure line, server response with TLS versions and cipher suits it works with. Once they agree upon common ones the handshake starts.
- the server sends copy of its public key, attached to its digital certificate to the client, the client checks if the certificate legitimate and moves into the next step.
- the client uses the server's public key and its private key to encrypt a 'session key' which will be used in order to encrypt/decrypt data in the particular session (this key will become invalid when connection is terminated). Using asymmetric encryption to encrypt symmetric encryption.
- both parties test the connection by sending encrypted messages, if they can decrypt them using the session key, then the handshake was successfully and connection is secured.


Hello, let's set up a secure SSL session
Hello, here is my certificate
Also checks that:
> Certificate is valid
> Signed by someone using trust services
1
2
Browser
3
Here is a one time encryption key for our session
(encrypted using Server's public key)
Server
4
Server decrypts session key using its private key and establishes a secure session
01010010110
0101001010

SSL Pinning:

SSL Pinning is the process of associating a host with its certificate or public key. Once a client (application in our case) knows a host's certificate or public key, the application pins it to that host.
This process allows the application to only trust valid or pre-defined certificate or public key. The applications developers use SSL pinning techniques as an additional security layer for application traffic. As normally, the application trusts custom certificate and allows the application to intercept the traffic.
Restricting the set of certificates through pinning prevents attackers from analyzing the functionality of the application and the way it communicates with the server.
The SSL pinning ignores the verification process with the intermediate CA and root CA, and directs the client to ignore any other certificate (even if it most trusted) if it doesn't match the pinned certificate.
Types of SSL Pinning:
Certificate pinning: this is a common method used by applications, when the application receives the certificate sent by the server, it compares it with the certificate that it has inside. The use of certificate pinning is, most of the time, observed in the case of a mobile app, as it is easy for the developer to include the expected certificate.
When the certificate arrives to expiration, the app has to be updated to insert the new valid certificate. This is the disadvantage of this method, if a service regularly changes its certificate, the app has to be updated regularly.
Public key pinning: This method consists in pinning a public key instead of a certificate. Then the public key pinned is compared to the public key extracted from the certificate.
Hashing: Another method is to compare information hashes instead of the certificate, or of the public keys.


CLIENT
SERVER
Client Certificate (receive server public key)
1. client hello
2. server hello
server certficate (public key)
Server Certificate (private and public key)
3. Calculate hashing
4. Match hashing
5. Pinning or not
OK
Match
Not match

NOTE: as explained in the picture above, if the pin fails to verify, the connection will not be generated and no traffic will be sent.

Ways to implement SSL pinning in Android:

TrustManager: TrustManager is responsible for deciding whether the app should accept credentials submitted by the host or not. It verifies the certificate against pre-coded values stored in a keystore. This method is created either using TrustManagerFactory (more info), or by implementing one of the TrustManager subclasses.

OkHttp: this is one of the simplest methods of implementing SSL pinning by using CertificatePinner class (more info),
NOTE: OkHttp 2.1 and below are vulnerable and can be exploited. (cve-details).

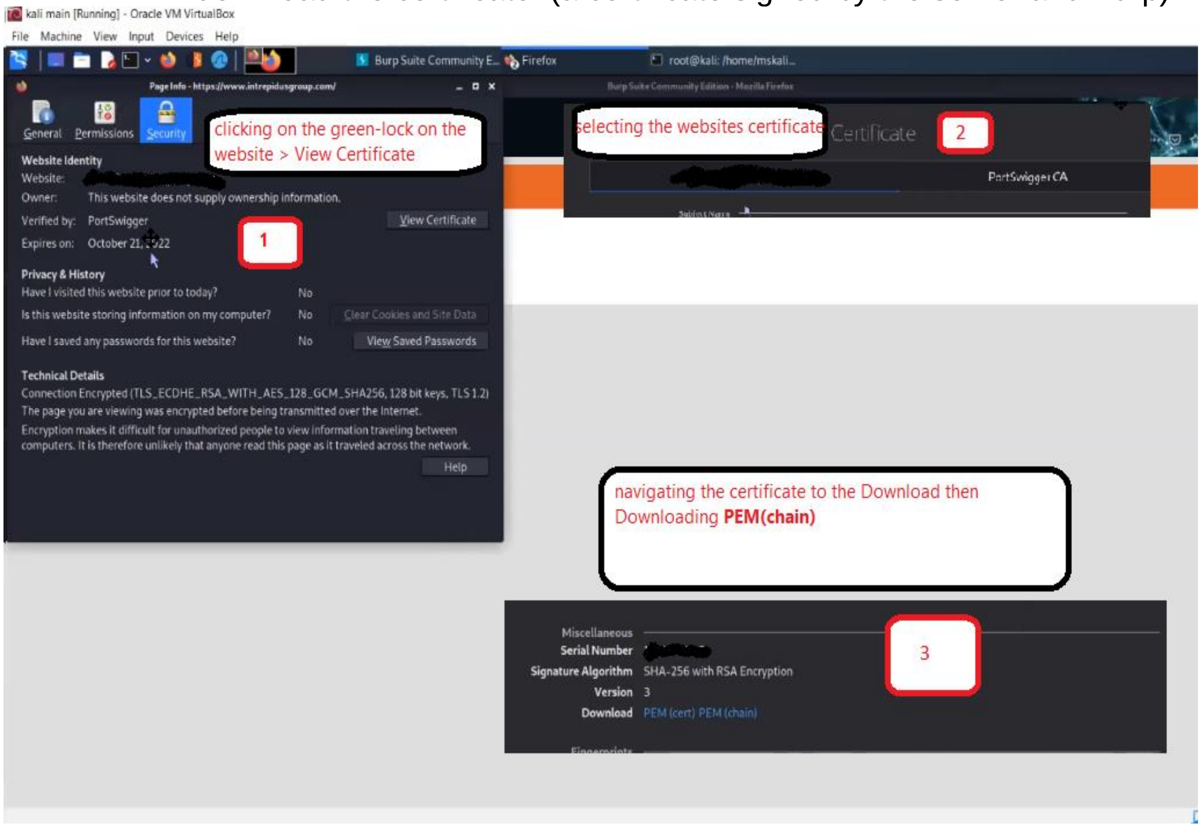
HttpUrlConnection: this can be implemented by personally writing a code that verify the parameters of the certificate and checking them via HttpUrlConnection.
NOTE: when using this method, it is better to bypass the code instead on manipulating it.

Pinning in Smali: if the SSL pinning is performed via the network configurations, then it has a high chance to be saved and found in the Smali code.
Navigating the relevant Smali files and finding hash then changing it will cause the application to verify and trust a specified certificate.

Bypassing the SSL pinning:

In order to bypass the SSL pinning several steps must be done:

1. setting proxy in the Android device in order to capture packets using Burp. (refer to page.12)
2. Downloading the certificate of the application webserver:
   - Using Burp, navigating to the website of the application webserver and download the certificate. (a certificate signed by the server and Burp).

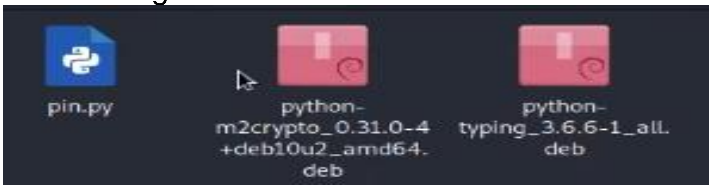

kali main [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Burp Suite Community E...
Firefox
root@kali: /home/mskali...
Page Info - https://www.intrepidusgroup.com/
General
Permissions
Security
clicking on the green-lock on the website > View Certificate
1
selecting the websites certificate
Certificate
2
PortSwigger CA
navigating the certificate to the Download then Downloading PEM(chain)
3
View Certificate
Clear Cookies and Site Data
View Saved Passwords
Help

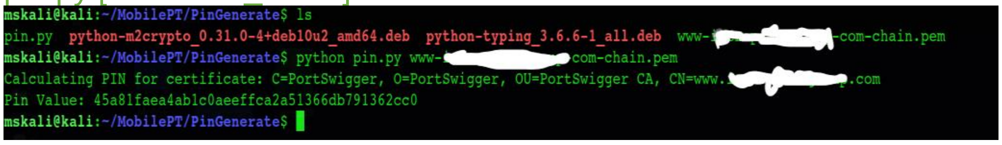
Website Identity
Website: [redacted]
Owner: This website does not supply ownership information.
Verified by: PortSwigger
Expires on: October 21, 2022

Privacy & History
Have I visited this website prior to today? No
Is this website storing information on my computer? No
Have I saved any passwords for this website? No

Technical Details
Connection Encrypted (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256, 128 bit keys, TLS 1.2)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.

Miscellaneous
Serial Number [redacted]
Signature Algorithm SHA-256 with RSA Encryption
Version 3
Download PEM (cert) PEM (chain)

- After having the certificate, it needs to be generated to a hash value, so this value can replace the hash value of the original pinned certificate:
  This process is done via several steps:
  1. Installing python-typing on kali (install tutorial)
  2. Installing python-m2crypto on kali (install tutorial)
  3. Getting pin.py on kali (Raw code- github)
  4. After getting all files listed above, create a new directory and place them all together.
     

  5. Installing the packages:
     a. dpkg -i python-typing….deb
     b. dpkg -i python-m2crypto…deb
  6. moving the certificate to the file containing the packages.
  7. generating hash (pin) for the new certificate using the command python pin.py [certificate_name]
     

```
mskali@kali:~/MobilePT/PinGenerate$ ls
pin.py  python-m2crypto_0.31.0-4+deb10u2_amd64.deb  python-typing_3.6.6-1_all.deb  www-[redacted]-com-chain.pem
mskali@kali:~/MobilePT/PinGenerate$ python pin.py www-[redacted]com-chain.pem
Calculating PIN for certificate: C=PortSwigger, O=PortSwigger, OU=PortSwigger CA, CN=www.[redacted].com
Pin Value: 45a81faea4ab1c0aeeffca2a51366db791362cc0
mskali@kali:~/MobilePT/PinGenerate$ 
```

  8. replacing the new pin with the pin in the app smali file.
  9. Recompiling and signing the app (refer to page.11) then installing it on the device.

A great link explains how to bypass SSL Pinning on Android.

Application Static & Dynamic Analysis:

Before starting with Analysis, Android components need to be clearly explained, in order to under perform better analysis and understand the results.

Android Components:

Activities: represent a single screen with a user interface and is the entry point to user interaction. Activities can be set as exported, exported activities can be accessed from another applications, this enables bypassing authorization mechanisms.
Services: are general-purpose entry points for keeping applications running in the background. It might for example be playing music in the background or fetching data over the internet without interrupting the current user activity.
Exposed services are a service that an application may provide to other applications, these services can relate to the application's internal operations and may provide an additional attack surface.
Broadcast receivers: broadcast receivers are components that listen for system calls. they enable the system to deliver events to applications outside of
the regular user flow. they can even deliver broadcasts to applications that are not running. a lot of broadcasts come from the system itself such as events of screen turning off or low battery.
Content providers: manage a shared set of data that can be stored within the device's file system as well as on the web using for instance SQLite database. upon permission other applications can query and modify data through the content provider. If exposed attacker can interact with the system without authentication to the target system.

Static Analysis: In static analysis application is tested from the inside out. It analyzes the source code or binary without executing the application and it does not rely on runtime environment.
A great and recommended tool for static analysis is MobSF.
MobSF (Mobile Security Framework): this tool is an automated, open-source, all-in-one mobile application pen-testing framework that is capable of performing Static, Dynamic and Malware analysis. it mostly used for static analysis, can be used for effective and fast analysis for Android, IOS and Windows mobile application and supports both binary and zipped source code.
It has a graphic UI in the form of web service. Web service consist of a dashboard that presents the results of the analysis, its own documentation site, an integrated emulator & an API that allows users to trigger the analysis automatically. It is hosted on a local environment, so sensitive data never interacts with the cloud.
For each vulnerability found, MobSF will attach CWE (Common Weakness Enumeration) data and point to the related files. MobSF also analyze the manifest.xml file for possible misconfigurations.
Requirements (kali):

- MobSF (github)
- Python 3.6 + (download).
- Oracle JDK 8 + (download).

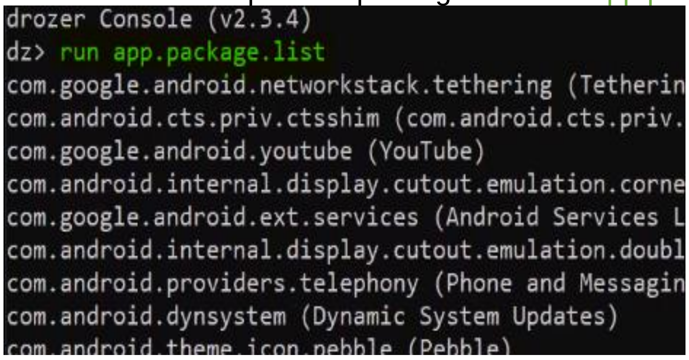
Deploying MobSF (kali):

- git clone https://github.com/MobSF/Mobile-Security-Framework-MobSF.git
- navigating to the MobSF directory and locating requirements.txt
- pip install -r requirements.txt (this installs all requirements of the application).
- ./setup.sh (setup MobSF).
- ./run.sh 127.0.0.1:8000 (running MobSF User Interface on localhost).

MobSF documents.

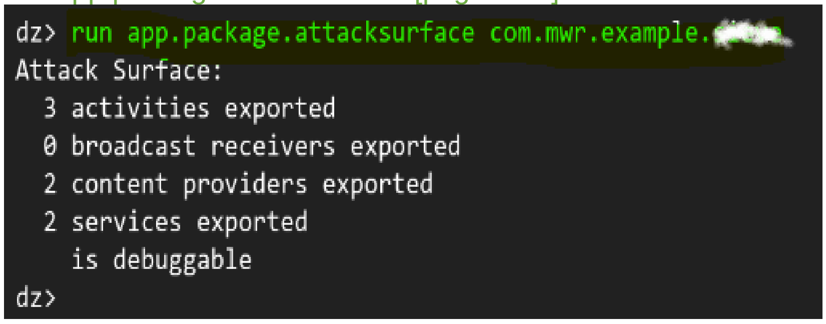
Dynamic Analysis: involves examining the app during runtime. This type of analysis can be manual or automatic. It usually doesn't provide the information that static analysis provides, but it is a good way to detect interesting elements (assets, features, entry points, etc.) from a user's point of view. This type of analysis focuses on testing and evaluation of apps via their real-time execution. The main objective of dynamic analysis is finding security vulnerabilities or weak spots in a program while it is running. Dynamic analysis is conducted both at the mobile platform layer and against the backend services and APIs, where the mobile app's request and response patterns can be analyzed.
A recommended way to dynamic test an application is using Drozer tool (download).
This tool is used to interact with the application during the runtime and trigger exposed components.
Deploying Drozer:

- Installing the agent.apk from Drozer directory on the device. Using ADB. (Refer to page.8)
- Opening CMD from the Drozer directory. And deploying the following commands:
  1. adb forward tcp:31415 tcp:31415 (port forwarding)
  2. drozer.bat console connect (connecting to the agent)
  3. locating the application via run app.package.list
     NOTE: to locate specified package use: run.app.package.list -f [pkg.name]



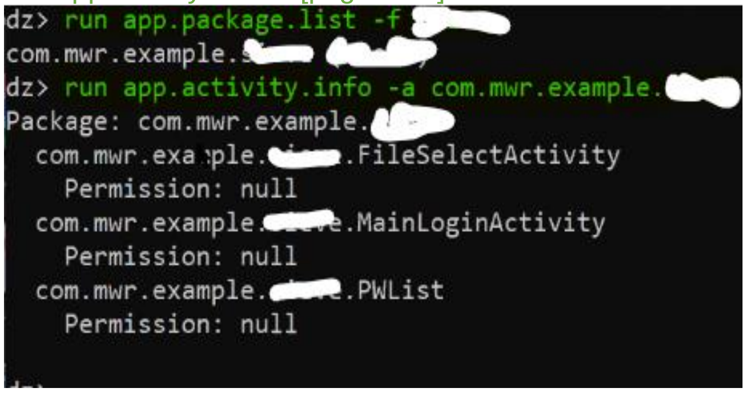
```
drozer Console (v2.3.4)
dz> run app.package.list
com.google.android.networkstack.tethering (Tetherin
com.android.cts.priv.ctsshim (com.android.cts.priv.
com.google.android.youtube (YouTube)
com.android.internal.display.cutout.emulation.corne
com.google.android.ext.services (Android Services L
com.android.internal.display.cutout.emulation.doubl
com.android.providers.telephony (Phone and Messagin
com.android.dynsystem (Dynamic System Updates)
com.android.theme.icon.pebble (Pebble)
```

  4. checking possible vulnerabilities:
     run app.package.attacksurface [pkg.name]



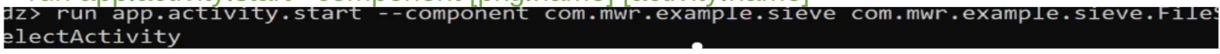
```
dz> run app.package.attacksurface com.mwr.example.[redacted]
Attack Surface:
  3 activities exported
  0 broadcast receivers exported
  2 content providers exported
  2 services exported
    is debuggable
dz>
```

  5. checking exported activities of the application and permissions:
     run app.activity.info -a [pkg.name]



```
dz> run app.package.list -f [redacted]
com.mwr.example.[redacted] [redacted]
dz> run app.activity.info -a com.mwr.example.[redacted]
Package: com.mwr.example.[redacted]
  com.mwr.example.[redacted].FileSelectActivity
    Permission: null
  com.mwr.example.[redacted].MainLoginActivity
    Permission: null
  com.mwr.example.[redacted].PWList
    Permission: null
```

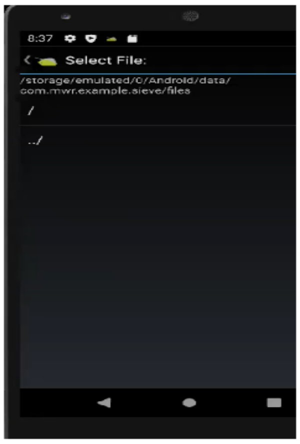
6. accessing a specified activity:
   run app.activity.start –component [pkg.name] [activity.name]



```
dz> run app.activity.start --component com.mwr.example.sieve com.mwr.example.sieve.FileS
electActivity
```

1

running this command in order to access the specified activity on the device.

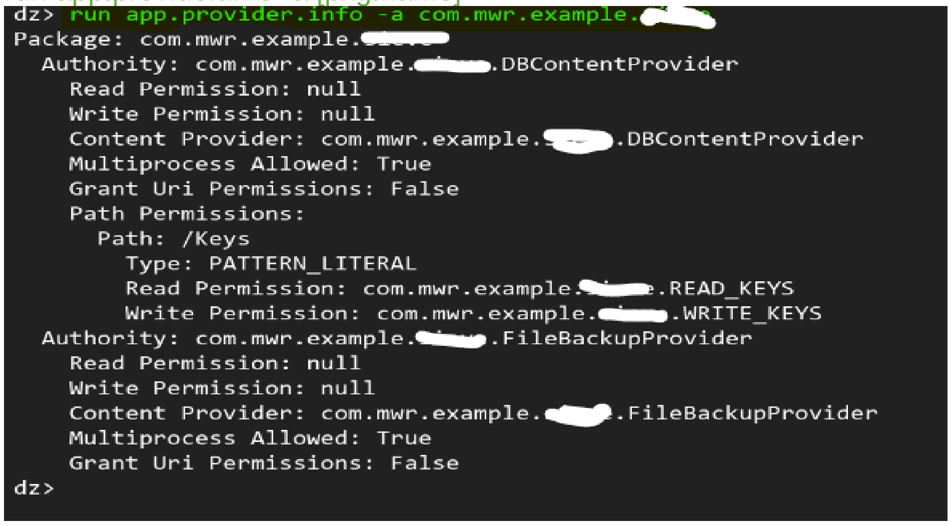


```
8:37
Select File:
/storage/emulated/0/Android/data/
com.mwr.example.sieve/files
/
../
```

2

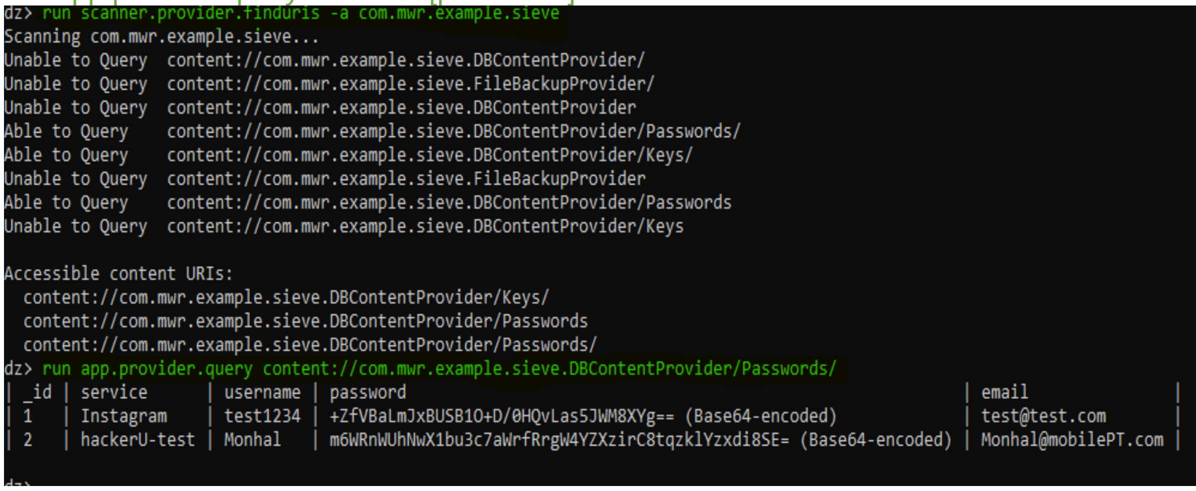
the activity is opened on the device . (remotely opened) and passed the authentication step.

7. checking exported content providers:
   run app.provider.info -a [pkg.name]



```
dz> run app.provider.info -a com.mwr.example.[redacted]
Package: com.mwr.example.[redacted]
  Authority: com.mwr.example.[redacted].DBContentProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.[redacted].DBContentProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
    Path Permissions:
      Path: /Keys
        Type: PATTERN_LITERAL
        Read Permission: com.mwr.example.[redacted].READ_KEYS
        Write Permission: com.mwr.example.[redacted].WRITE_KEYS
  Authority: com.mwr.example.[redacted].FileBackupProvider
    Read Permission: null
    Write Permission: null
    Content Provider: com.mwr.example.[redacted].FileBackupProvider
    Multiprocess Allowed: True
    Grant Uri Permissions: False
dz>
```

8. checking available URIs and quiring it:
   run scanner.provider.finduris -a [pkg.name]
   run app.provider.query content://[path.to.uri]



```
dz> run scanner.provider.finduris -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider/
Unable to Query  content://com.mwr.example.sieve.DBContentProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwords/
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Keys/
Unable to Query  content://com.mwr.example.sieve.FileBackupProvider
Able to Query    content://com.mwr.example.sieve.DBContentProvider/Passwords
Unable to Query  content://com.mwr.example.sieve.DBContentProvider/Keys

Accessible content URIs:
  content://com.mwr.example.sieve.DBContentProvider/Keys/
  content://com.mwr.example.sieve.DBContentProvider/Passwords
  content://com.mwr.example.sieve.DBContentProvider/Passwords/
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/
| _id | service     | username | password                                                | email               |
| 1   | Instagram   | test1234 | +ZfVBaLmJxBUSB1O+D/0HQvLas5JWM8XYg== (Base64-encoded)          | test@test.com       |
| 2   | hackerU-test | Monhal  | m6WRnWUhNwX1bu3c7aWrfRrgW4YZXzirC8tqzklYzxdi8SE= (Base64-encoded) | Monhal@mobilePT.com |
```

USEFUL LINKS:

Random posts about mobile security and testing techniques.

Sample of a reverse engineering of real application.

Link rich of information about mobile app pentesting.

Real-time pentest report of Spotify.

Application workflow (how Android application work).

Frida Documents.

Basic rich information about Android Encryption.

for any questions, please contact me on monhalsarbouch@gmail.com.

Thank you, hope you enjoyed reading.