hardsecure

CASE STUDY

ANDROID

MALWARE ANALYSIS

ANDROID REVERSE ENGINEERING

R. Acácio de Paiva nº 16, 1ºdir,
1700-006 Lisboa, Portugal

www.hardsecure.com
geral@hardsecure.com

ÍNDICE

ÍNDICE DE FIGURAS

OBJETIVO

O presente documento tem por objetivo apresentar, de forma detalhada e com evidências, o resultado da análise de uma aplicação Android utilizada durante uma campanha de *phishing* contra clientes de organizações bancárias portuguesas, entre elas um nosso cliente X. Estas campanhas de *phishing* têm como objetivo roubar as credenciais de acesso dos utilizadores das organizações bancárias de forma a serem utilizadas posteriormente para ações não autorizadas sobre as contas dos utilizadores nas plataformas mobile dos respetivos bancos.

Outro objetivo deste documento é expor a necessidade de uma análise de segurança e de *malware* dos vários componentes que constituem uma aplicação Android, sejam estes código desenvolvido à medida ou bibliotecas de terceiras partes, antes da exposição e publicação da mesma, de forma a garantir o funcionamento esperado da aplicação, uma real proteção dos dados pessoais visados na aplicação e diminuindo potenciais danos ou superfícies de ataque que tipos de *malware* ou explorações de vulnerabilidades de segurança possam causar a uma empresa.

ENQUADRAMENTO

O Banco X permite aos seus clientes autenticaram-se e realizarem ações e movimentos nas respetivas contas bancárias através de plataformas/portais *web* assim como através da sua aplicação mobile Android.

Após registo de alguns incidentes de segurança relacionados com ações, movimentos estranhos e não autorizados pelos clientes do Banco X, foi pedido à Hardsecure que inicia-se o processo de investigação do incidente. Através dos relatos de vários utilizadores da aplicação *mobile* Android do Banco X, foi possível identificar de imediato que se tratava de uma campanha de *phishing* contra os clientes do Banco X, que recorria desta vez à utilização de uma aplicação Android clone disponível na App Store, e que se fazia passar pela aplicação original do Banco X.

Com a ajuda da tool 0xSI_f33d, uma ferramenta de partilha de domínios maliciosos focada em campanhas maliciosas direcionadas a cidadãos portugueses, descobrimos que a presente aplicação maliciosa era de facto uma variante de uma aplicação utilizada noutra campanha de *phishing* a clientes de bancos portugueses e já analisada pelo autor da ferramenta, Pedro Tavares.

ESTUDO E ANÁLISE DO VETOR DE ATAQUE

Estes tipos de campanhas são observados desde 2020, onde campanhas de *phishing* e *smishing* são lançadas contra utilizadores das aplicações dos bancos. Estes utilizadores são identificáveis muito provavelmente devido à exposição dos seus números de telemóvel ou email em bases de dados que os associam aos respetivos bancos.

Devido aos vários requisitos de segurança impostos num mecanismo de autenticação de aplicações consideradas críticas e não só nos dias de hoje, as aplicações de carácter bancário começaram a utilizar um segundo fator de autenticação para validar a autenticidade e a legitimidade do utilizador ao fazer login na sua conta bancária ou realizar algum tipo de ação sobre a mesma. Normalmente, este segundo fator de autenticação corresponde a um SMS recebido no telemóvel do utilizador e que valida a autenticidade do mesmo.

O desenvolvimento deste tipo de aplicações Android que espelham o comportamento de uma aplicação mobile bancária é certamente mais trabalhoso do que desenvolver apenas uma *landing page* para um *web browser* que solicita e obtêm as credenciais de acesso. No entanto, a aplicação Android permite automatizar o processo de *phishing* e recolha do segundo fator de autenticação. Sem a aplicação Android, o atacante está dependente de um processo semiautomático para recolher o segundo fator de autenticação enviado para o telemóvel da vítima. A título de exemplo são apresentadas duas imagens que refletem o fluxo de uma típica campanha de *phishing* e o fluxo de uma campanha de *phishing* com recurso a uma aplicação mobile Android.

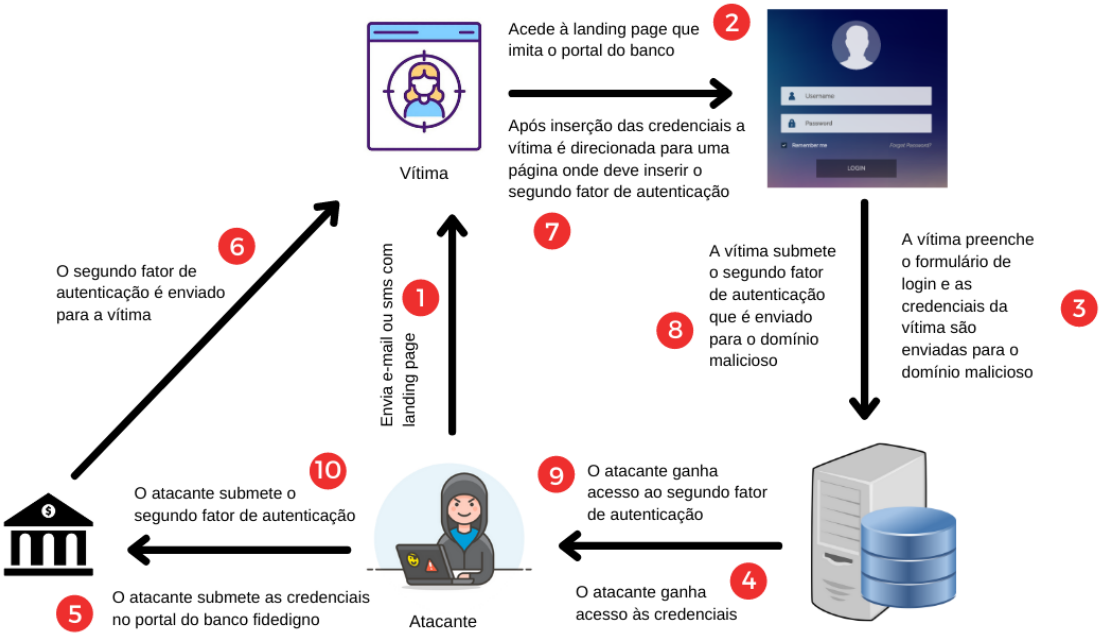

Figura 1 - Fluxo de uma típica campanha de *phishing* dependente da inserção do segundo fator de autenticação por parte da vítima.

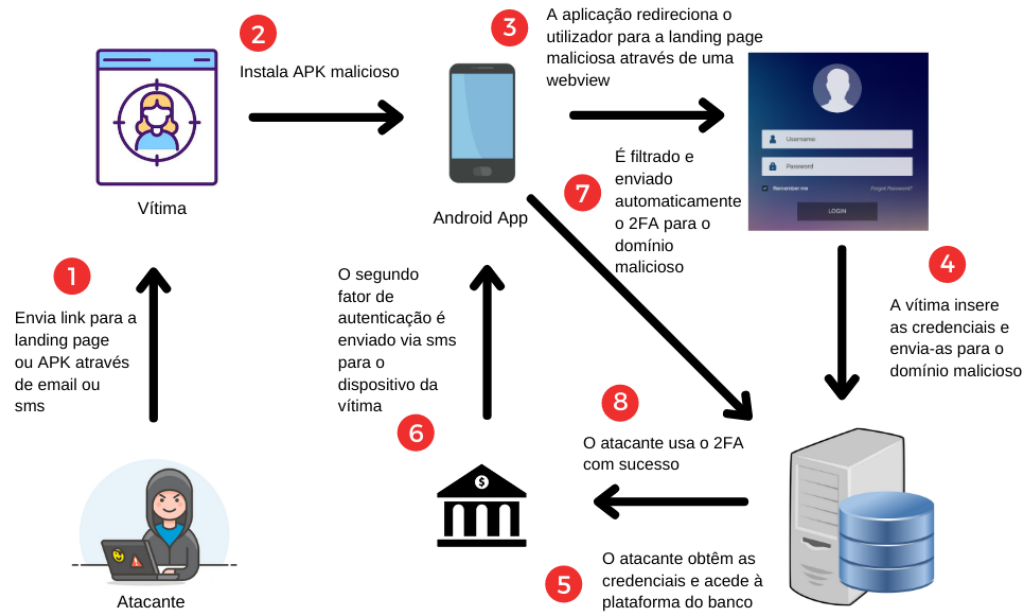

Figure 2 - Fluxo de uma campanha de *phishing* recorrendo a uma aplicação Android maliciosa capaz de automatizar a recolha e envio do segundo fator de autenticação (2FA)

É possível verificar que neste segundo caso, o atacante pode automatizar todo o processo após a inserção das credenciais do utilizador na *landing page* que simula o portal do banco, uma vez que tem a possibilidade de recolher automaticamente e enviar o conteúdo (2FA) recebido através sms no dispositivo do utilizador.

ARTEFATOS E EVIDÊNCIAS

O Android é um sistema operativo mobile baseado numa versão modificada do Linux e outros softwares *open-source*, projetado principalmente para dispositivos móveis com tela de toque, como smartphones e tablets. Este sistema destaca-se pelas suas características únicas e pela forma como as aplicações estão processadas, são desenhadas e construídas, o que se reflete no seu comportamento.

De forma a fazer uma análise rápida à aplicação Android, aos vários componentes que constituem o APK passamos a usar algumas tools como o apktool, o dex2jar, o jd-gui e o burp.

Todas as aplicações Android são compostas por um Android Manifest. O Android Manifest é um ficheiro XML que contém meta-dados importantes sobre a aplicação. Este ficheiro inclui o nome do pacote, nomes das várias atividades, atividade principal (o ponto de entrada para a aplicação), nomes dos vários *services* e *receivers*, suporte à versão Android, suporte a recursos de hardware, permissões e outras configurações. As permissões têm um papel fundamental uma vez que dão à aplicação a possibilidade de aceder a determinados recursos ou fazer determinadas ações fora da *sandbox* imposta pelo sistema.

Neste caso, para permitir este ataque de *phishing*, a aplicação deve usar as seguintes permissões no Android manifest:

1. INTERNET para aceder à página falsa do portal do banco onde está presente o formulário de login e onde são inseridas as credenciais de acesso;

2. RECEIVE_SMS para saber o momento em que o dispositivo móvel recebe um novo SMS;

3. READ_SMS para ler o conteúdo do SMS recebido e efetuar um filtro de modo a filtrar o Segundo fator de autenticação do SMS pretendido;

4. RECEIVE_BOOT_COMPLETED para lançar a aplicação assim que o dispositivo é ligado ou reiniciado;

5. FOREGROUND_SERVICE para dar início aos componentes da aplicação denominados como *services*, os quais são rotinas que podem correr em *background* e que raramente são desligadas;

6. BIND_NOTIFICATION_LISTENER para retirar o segundo fator de autenticação do conteúdo da notificação lançada sobre o ecrã do dispositivo assim que o dispositivo recebe um novo SMS e caso a aplicação não tenha as permissões 2 e 3.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="[redacted]" >  <!-- the application package -->

    <!-- The list of permissions -->
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>


    <!-- The application -->

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="My Application"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="My Application" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>


</manifest>
```

Figura 3 – Exemplo de um ficheiro AndroidManifest.xml

Esta aplicação em específico apresentava dois pontos de entrada que poderiam dar início ao código malicioso:

1. Sempre que a aplicação recebia um *broadcast* do dispositivo com a ação BOOT a indicar que o dispositivo tinha sido ligado ou reiniciado. Neste caso, a aplicação usa uma classe do tipo *receiver* que espera pela ação ou *intent* e que dá início à rotina indicada.

```java
public class MyCustomBroadcastReceiver extends BroadcastReceiver {

    @Override
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        if(action != null) {
            if (action.equals(Intent.ACTION_BOOT_COMPLETED) ) {
                // TO-DO: Code to handle BOOT COMPLETED EVENT
                // TO-DO: I can start an service.. display a notification... start an activity
            }
        }
    }
}
```

Figura 4 – Exemplo de uma classe com características de receiver que aguarda o intent ACTION_BOOT_COMPLETED para iniciar a rotina especificada.

2. Sempre que a aplicação é lançada pelo utilizador do dispositivo ao clicar no ícone da aplicação o que irá dar início à classe principal ou *main*.

```
package com.example.android.activity;

import android.os.Bundle;
import android.app.Activity;

public class MainActivity extends Activity {

@Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

   }
}
```

Figura 5 – Exemplo de uma *main activity* que é lançada por *default* assim que o utilizador clica no ícone da app

Ambos os pontos de entrada irão dar início ao rastreio de diversos métodos em diferentes classes e serviços que utilizam diferentes técnicas de encriptação e de ofuscação de forma a dificultar o trabalho de *Reverse Engineering*. Com a ajuda do manual do sistema Android disponível em *developer.android.com*, temos uma ideia sobre qual o papel de cada classe durante a execução de uma aplicação Android assim como que respetivos métodos são iniciados em que circunstâncias. Por exemplo, o método *onCreate()* é executado sempre que uma classe é iniciada. De seguida, é apresentada uma imagem retirada do anterior web site e que explica as circunstâncias em que específicos métodos são despoletados numa atividade ou classe.

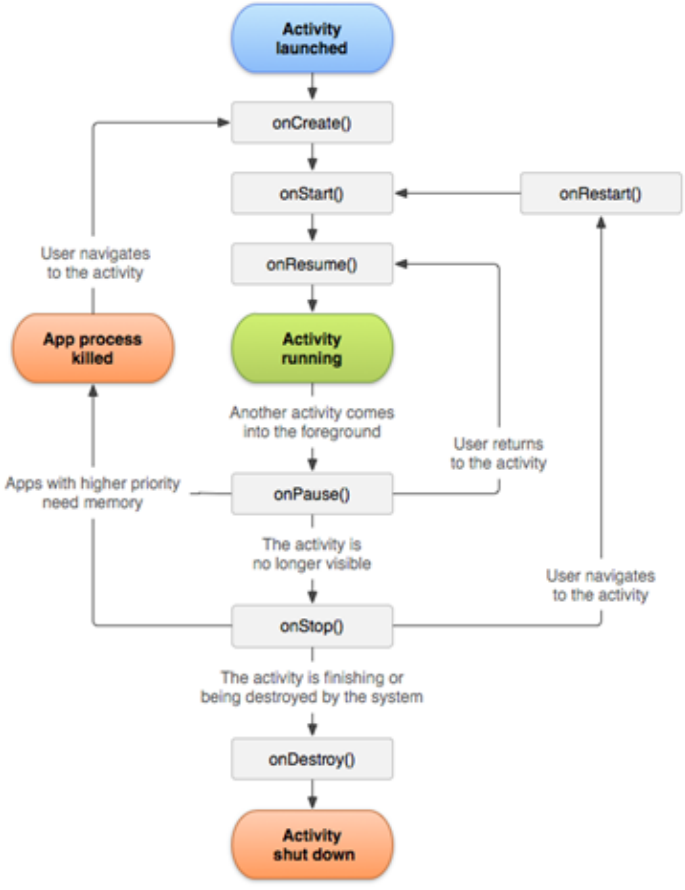

Figura 6 – O diagrama mostra os vários estados e métodos despoletados de uma atividade.

Depois de recebido o *broadcast* do *boot* ou de iniciada a aplicação, a criação de um serviço é iniciada através de um *intent* que especifica a classe java ou kotlin do serviço a ser executado.

```
Intent newIntent = new Intent(this, newService.class);
startService(newIntent);
```

Figura 7 – Criação de um serviço assincronamente.

Após algumas chamadas de métodos, deparámo-nos com um método "initializeProcessGlobals()" que iria dar inicio à inicialização de várias variáveis usadas no comportamento malicioso tais como as credenciais do utilizador e do URL com o domínio malicioso. Adicionalmente neste método seriam chamados outros métodos que deram início à instanciação dos objetos que gerem as notificações, os pedidos HTTP, a atividade do ecrã e os SMS recebidos.

```
10  public static void initializeProcessGlobals() {
11      if (!processGlobalsRun) {
12          processGlobalsRun = true;
13          try {
14              //initialize webView
15              //initialize sms listener
16              //initialize variables (user, pass) to be populated
17              //initialize http client to send variables
18          } catch (Exception e){
19              throw new RuntimeException(e);
20          }
21      }
22  }
```

Figura 8 – Exemplo do método que por sua vez chama os métodos que irão iniciar a webview, o sms listener, as variáveis a serem populadas e o cliente http.

Ao fazer o *drill-down* dos métodos, chegamos a um URL utilizado pelo cliente HTTP para popular a variável a utilizar na *webview*. Ao navegarmos com o browser para o URL indicado recebemos o URL da página falsa de login do banco onde os utilizadores irão inserir as credenciais para a operação de *phishing*.

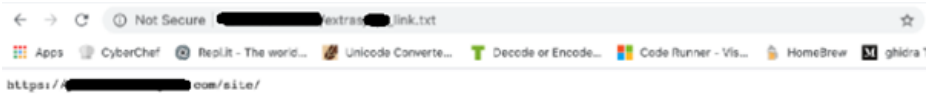

Figura 9 – Extração do url da fake login page.

Neste momento, o URL seria utilizado na *webview* e apresentada a página falsa ao utilizador para que este inserisse as suas credenciais.

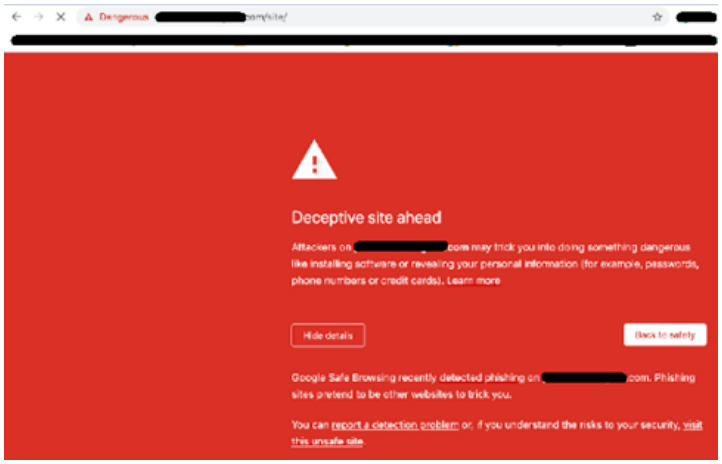

Figura 10 – Suposta fake login page já advertida pelo browser

É então essencial perceber o funcionamento do comportamento malicioso através da utilização da *webview* que funciona como um browser embutido na aplicação e que permite navegar até ao domínio malicioso, imitando assim o portal do banco, de forma a enganar o utilizador a preencher as suas credenciais. Tal ação depois despoleta a execução de um pedido REST POST que envia as credenciais inseridas pelo utilizador (*username* e *password*) para o URL remoto, também designado de domínio malicioso.

```
4  WebView myWebView = (WebView) findViewById(R.id.webview);
5  WebSettings webSettings = myWebView.getSettings();
6  webSettings.setJavaScriptEnabled(true);
7  webView.addJavascriptInterface(new WebAppInterface(this), "Android");
8  myWebView.loadUrl("http://www.fakeloginpage.com");
```

Figura 11 – Criação de *webview*, configuração da *interface* javascript e carregamento da página de login falsa do banco.

Uma das particularidades de se utilizar uma *webview* numa aplicação android passa pela possibilidade de adicionar uma *interface* javascript (linha 7 da figura 11), o que permite ao atacante manipular código do lado da aplicação (Figura 12) através de chamadas javascript no código hospedado no domínio malicioso (Figura 13) e que posteriormente é enviado para a *webview* sob a forma de página HTML como ocorre normalmente num web browser.

```
public class WebAppInterface {
    Context mContext;

    /** Instantiate the interface and set the context */
    WebAppInterface(Context c) {
        mContext = c;
    }

    /** Show a toast from the web page */
    @JavascriptInterface
    public void showToast(String toast) {
        Toast.makeText(mContext, toast, Toast.LENGTH_SHORT).show();
    }
}
```

Figura 12 – Exemplo de uma interface javascript na app Android.

```
<input type="button" value="Say hello" onClick="showAndroidToast('Hello Android!')" />

<script type="text/javascript">
    function showAndroidToast(toast) {
        Android.showToast(toast);
    }
</script>
```

Figura 13 – Exemplo de código javascript numa página HTML a executar um método da interface javascript "Android" na aplicação Android.

Neste caso em específico, o atacante escolheu o URL a que a *webview* deveria aceder e, portanto, a página HTML que seria descarregada para a mesma. Nas figuras 12 e 13, a título de exemplo, o atacante iria criar um *toast* (notificação) no *display* do smartphone do utilizador a dizer "Hello Android!".

Por fim, analisámos o método que regista o SMS *listener* e que executa um pedido REST POST para o domínio malicioso assim que o utilizador recebe o segundo fator de autenticação em forma de mensagem ou eventualmente em forma de notificação.

```
25  private class SMSReceiver extends BroadcastReceiver {
26
27      @Override
28      public void onReceive(Context context, Intent intent) {
29          if (intent.getAction().equals(Telephony.Sms.Intents.SMS_RECEIVED_ACTION)) {
30              SmsMessage[] smsMessages = Telephony.Sms.Intents.getMessagesFromIntent(intent);
31              for (SmsMessage message : smsMessages) {
32                  // Exfiltrate the 2FA code
33                  // Populate the variable
34                  // Send it over hhtp client previously created
35              }
36          }
37      }
38  }
```

Figura 14 – Método que agarra o sms quando este cai no dispositivo e que exfiltra o mesmo e popula as variáveis que eventualmente são enviadas através do *http client* para o domínio malicioso.

RECOMENDAÇÕES

Para os utilizadores em geral, recomendamos, no caso de usar regularmente o homebanking:

- Alterar com frequência a palavra-chave;
- Não aceder ao site do banco através de *links* enviados em e-mails, sms's, ou qualquer outra aplicação de mensagens.

- Apenas instalar aplicações Android (.APK) em APP Stores legitimas como a Google Play Store;
- Não abrir anexos de emails não solicitados, mesmo que enviados por amigos ou conhecidos pois podem ser emails já comprometidos;
- Nunca enviar o seu nome de utilizador, código de acesso ou cartão-matriz por e-mail ou SMS ou qualquer aplicação de mensagens;
- Desconfiar sempre de mensagens com endereços estranhos ou escritos em português incorreto;
- Nunca inserir dados pessoais em páginas que não garantam uma ligação segura, isto é, que não comecem por "https://";
- Terminar sempre a sessão após concluídas as operações no *homebanking*;
- Finalmente, consultar periodicamente a sua conta bancária;

CONCLUSÃO

Através do uso de ferramentas especializadas e de técnicas específicas, é potencializada a segurança das várias aplicações Android que fazem parte dos instrumentos de operação de uma instituição ou da interação entre a instituição e o utilizador final. Através deste artigo conseguimos, através de um caso real, expor a necessidade de uma análise de segurança e de malware dos vários componentes que constituem uma aplicação Android, sejam estes código desenvolvido à medida ou bibliotecas de terceiras partes, antes da exposição e publicação da mesma, de forma a garantir o funcionamento esperado da aplicação, uma real proteção dos dados pessoais visados na aplicação e diminuindo potenciais danos ou superfícies de ataque que tipos de malware ou explorações de vulnerabilidades de segurança possam causar a uma instituição. Neste caso, a análise efetuada serviu para identificar comportamentos maliciosos, os vários *workflows* do mesmo comportamento, identificar os vários URLs e domínios utilizados neste comportamento para posterior *report* às entidades competentes. Noutros casos, uma análise poderá ser necessária para validar que componentes terceiros, como bibliotecas que são utilizadas nas aplicações desenvolvidas “in-house”, se comportam da forma esperada e que não contêm nenhum tipo de *malware* como *spyware, toll-fraud, ransomware, phishing, call-fraud, Trojan*, etc.