Cibersegurança

Segunda série de exercícios

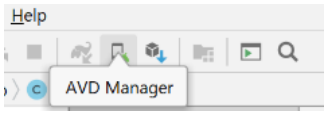
Objetivos:

- Compreender a estrutura de um APK
- Saber fazer o *repackaging* de uma aplicação Android
- Saber obter um certificado digital a partir de um *keystore* Java
- Saber utilizar algoritmos simétricos de uma biblioteca criptográfica
- Usar a aplicação www.virustotal.com para detetar *malware*
- Usar a ferramenta FlowDroid sobre um APK classificado como *malware* e encontrar o código problemático

❖ Preparação

Pré-requisitos: Ambiente integrado de desenvolvimento Android Studio ou apenas as ferramentas de linha de comando - https://developer.android.com/studio

Para evitar o uso de dispositivos reais, as questões seguintes assumem a existência de um Android Virtual Device (AVD). Existindo o AVD "cslab", o mesmo pode ser lançado com o emulador Android usando o comando:




```
emulator @cslab
```

Mais informações em:

https://developer.android.com/studio/run/emulator-commandline

https://developer.android.com/studio/command-line/sdkmanager

https://developer.android.com/studio/command-line/avdmanager

Note que caso queira usar um dispositivo real terá de ativar o modo de programador para poder instalar APKs via linha de comandos ou Android Studio: https://developer.android.com/studio/debug/dev-options

Parte I - *Refactoring*

1. Considere o APK `RepackagingLab.apk` em anexo:
    a. Instale o APK no AVD e veja a imagem que aparece no ecrã inicial;
    b. Realize o repackaging do APK, mudando a imagem do ecrã inicial. Note que a imagem é um recurso presente em `[apk]\res\drawable`;
    c. Instale o novo APK.

    Entrega**:**

- Evidências dos passos executados;
- novo APK;
- certificado associado à chave privada que assina a nova versão do APK.

2. O `Lab2_2.apk` em anexo é uma aplicação que pede uma frase ao utilizador e verifica se é ou não a frase correta. No entanto, a frase correta foi cifrada e escrita, juntamente com a respetiva chave, no código fonte da aplicação.
    a. Instale o APK e teste a aplicação;
    b. Descompile o APK e analise o código. Na classe `MainActivity`, o método `void verify(View v)` é chamado para verificar se a *string* introduzida na caixa de texto é a correta. A partir deste método determine onde está:
        i. a frase correta cifrada
        ii. a chave usada para cifrar
        iii. o algoritmo de decifra e comparação;
    c. Realize um programa em Java que decifra a frase correta e teste a frase na aplicação Android.

    Entrega**:**

- Breve descrição dos passos realizados;
- Código fonte do programa que decifra a frase.

Notas sobre a biblioteca criptográfica da plataforma Java (JCA):

- O ficheiro `SymCipher.java` tem um exemplo de uma aplicação que usa a JCA para cifrar e decifrar uma mensagem com AES em modo CBC e *padding* PKCS#5;
- Na JCA a classe `Cipher` cifra e decifra usando o algoritmo (simétrico ou assimétrico), o modo de operação e o tipo de *padding*, indicados no método `getInstance`. Se a combinação não for suportada é lançada exceção em tempo de execução;
- As instâncias da classe `SecretKeySpec` são representações de chaves simétricas para determinado algoritmo simétrico (e.g. DES, AES), sendo possível afetar e ler diretamente o valor da chave. Estas representações são designadas de transparentes, em oposição à representação opaca (https://docs.oracle.com/en/java/javase/11/docs/api/java.base/javax/crypto/spec/SecretKeySpec.html).

Parte II – Análise estática de código

1. Considere o APK `com.mediawoz.gotq.apk` presente no repositório https://github.com/tootsy42/Ludroid-dataset. Faça *download* para o sistema de ficheiros. Analise-o com a aplicação www.virustotal.com e apresente um resumo dos resultados.

   ** Não instale o APK num dispositivo real. **

2. Use a aplicação FlowDroid https://github.com/secure-software-engineering/FlowDroid para analisar o APK referido na alínea anterior.
   a. Use a opção -o <file.xml> para guardar o resultado da análise num ficheiro;
   b. Procure no ficheiro de *output* os seguintes elementos. Porque motivo foram identificadas as classes `HttpURLConnection` e `FileOutupStream` como *source* e *sink*, respetivamente?

```
<Sink Statement="virtualinvoke $r11.&lt;java.io.FileOutputStream: void write(byte
[],int,int)&gt;($r1, 0, $i5)" Method="&lt;com.exchange.Public.DownloadingService$
a: void a()&gt;">
  ...
</Sink>
<Sources>
<Source Statement="$r12 = virtualinvoke $r5.&lt;java.net.HttpURLConnection: java.
io.InputStream getInputStream()&gt;()" Method="&lt;com.exchange.Public.Downloadin
gService$a: void a()&gt;">
  ...
</Source>
</Sources>
```

   c. O APK é *malware* (um Trojan – cavalo de tróia), o qual faz *download* de outros APK para o dispositivo. Identifique no código o local onde este *download* acontece.

3. [*extra*] Considere o artigo "FlowDroid: Precise Context, Flow, Field, Object-sensitive and Lifecycle-aware Taint Analysis for Android Apps" (https://www.bodden.de/pubs/far+14flowdroid.pdf), onde são descritas as técnicas usadas pela ferramenta `FlowDroid`. Quais os pressupostos do modelo de ameaças? Qual a diferença entre fluxo explícito e implícito de dados?