

Instalação e configuração do ambiente:

Para quem está lendo isso e deseja fazer também, seguem abaixo as instruções para emular o ambiente, em Windows.

Antes de começarmos vale destacar que, durante o processo, eu perdi acesso as máquinas virtuais do virtual box, pois houve conflito com o WSL. Com isso, tive que optar pela opção de download do genymotion sem virtual box, pois nenhuma virtualização minha estava funcionando mais, para evitar futuros conflitos, optei pela opção do genymotion “without virtualbox”

1) Genymotion:

- Genymotion: basta iniciar o download no link abaixo. <without virtualbox>

<https://www.genymotion.com/>

2) Android Studio:

Optei pela opção do Android Studio devido as dificuldades citadas anteriormente, após o passo 4 eu mostro um problema com a utilização do JADX via Android Studio.

- ADB: É uma utilidade usada pra interagir com um dispositivo android, que eu usei na aula anterior pra abrir uma shell dentro do smartphone. Vocês podem instalar ela de duas formas:

- De forma avulsa, se você usa algum sistema Linux que tem isso disponibilizado (é o meu caso com Arch). Talvez seja possível fazer isso com o Windows usando o WSL, mas eu não saberia auxiliar vocês nisso

- Baixando o Android Studio, que com ele vem outras utilidades, incluindo o ADB

3) JADX:

Optei pelo jadx

3.1)baixei o arquivo em .zip <https://github.com/skylot/jadx>, extrai o arquivo

3.2) extrair os arquivos

apktool	9/12/2017 4:59 PM	File folder	
jadx	9/12/2017 5:24 PM	File folder	
setups	9/12/2017 5:14 PM	File folder	
targets	9/12/2017 5:20 PM	File folder	
tools	9/12/2017 5:24 PM	File folder	
jadx-0.6.1.zip	9/12/2017 5:18 PM	WinRAR ZIP archive	4,014 KB

3.3) jadx --> bin --> jadx-gui.bat

- JADX ou APKTOOL: Duas ferramentas de análise estática, qualquer uma delas vai servir. Na aula eu vou usar o JADX porque é o que eu estou acostumado, mas se quiserem usar algo diferente, sintam-se à vontade (<https://github.com/skylot/jadx> ou <https://ibotpeaches.github.io/Apktool/>)

4) Frida:

A instalação via pip não deveria ter grandes dificuldades, entretanto, foi necessário adicionar o path do python para executar o comando “pip”.

- Frida: Ferramenta de análise dinâmica feita em Python e distribuída via PIP. Um pip install frida-tools deve resolver tudo, mas segue o site pra referência (<https://frida.re/docs/installation/>)

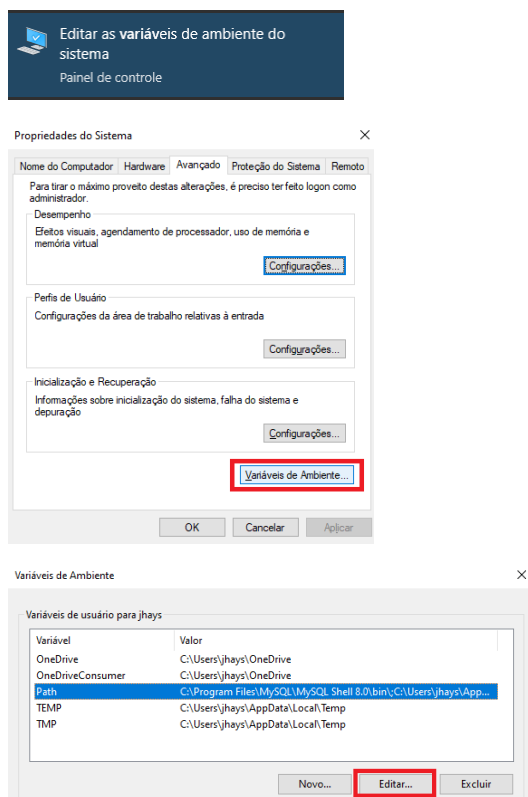
Dois problemas, uma solução:

O JADX e o python precisavam ser adicionados ao PATH do Windows, segue abaixo o passo a passo de como fazer.

Passo 1:

Adicionar o caminho do python nas variáveis de ambiente

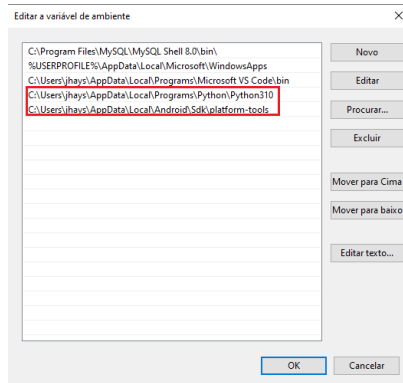
Pesquisar por variáveis de ambiente



Passo 2:

-Adicionar o path.

O path é relativo de acordo com o local de instalação, mas se o seu caso foi o default na instalação do python e Android Studio, provavelmente estará nos mesmos diretórios abaixo, mudando apenas o User.



Correção de erro:

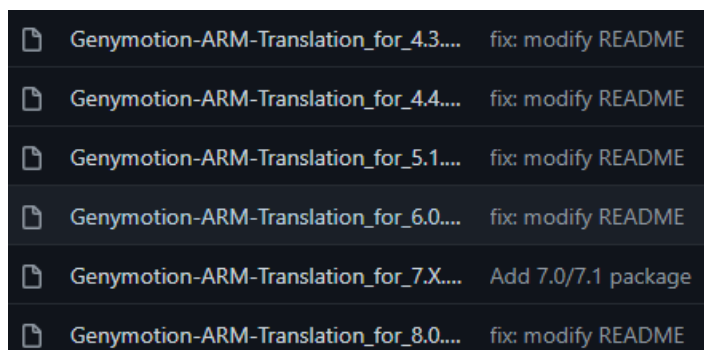
Problema --> eu estava tentando instalar um aplicativo que não tem suportes arquitetura do meu PC, pois, alguns apps que só possuem suporte para ARM

```
PS C:\Program Files\tag de engenharia reversa android> adb.exe install MSTG-Android-Java.apk
Performing Streamed Install
adb: failed to install MSTG-Android-Java.apk: Failure [INSTALL_FAILED_NO_MATCHING_ABIS: Failed to extract native libraries, res=-113]
PS C:\Program Files\tag de engenharia reversa android>
```

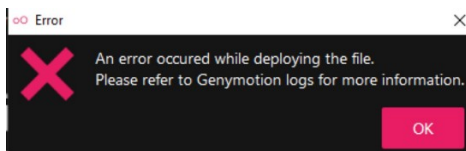
Solução --> baixar o “tradutor” .zip de ARM, de acordo com a versão do android instalado no genymotion

no link para download: https://github.com/m9rco/Genymotion_ARM_Translation dentro da pasta packages

dentre os arquivos .zip disponíveis, temos 6 opções:



Tentei a opção 4.4 mas não obtive sucesso:



Segui as mesmas instruções, mas utilizando um android 7.1, ao invés do 4.4:

	Custom Phone	4.4 - API 19	768 x 1280	320 - XHDPI	Genymotion	Off	
	Custom Phone_1	7.1 - API 25	768 x 1280	320 - XHDPI	Genymotion	On	

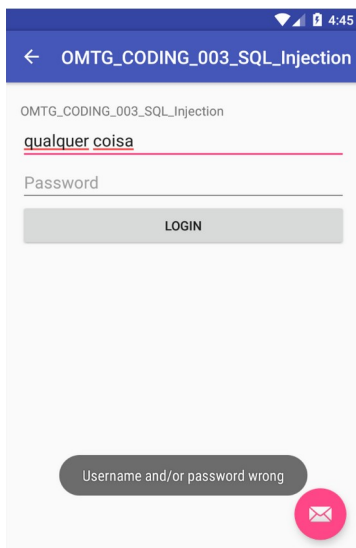
Obtive sucesso:

```
PS C:\Program Files\tag de engenharia reversa android> adb.exe install .\MSTG-Android-Java.apk
Performing Streamed Install
Success
```

Sql injection, Exercício 1:

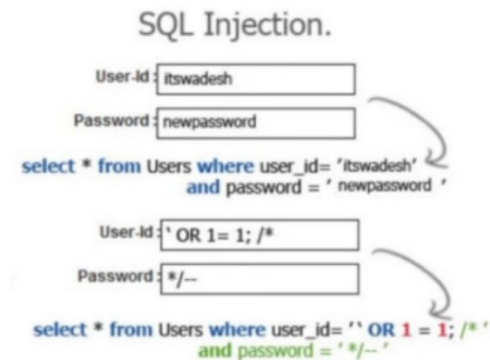
Análise dinâmica:

- testei a login e senha “admin” para ver como funcionava a execução

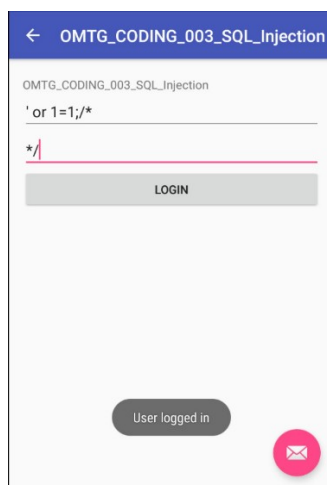


Seguindo as dicas dadas no get de web1/web2:

Passamos para o login um fecho aspas e a sentença or 1=1 (sempre verdadeira) e comentamos a requisição de senha



Conseguimos acesso com sucesso:



Vale destacar que a forma que eu acessei não foi a mais correta,

Forma que eu fiz (não ideal):

Note que ficou uma aspa solta no fim do comando

```
select user, password from users where user='' or 1=1; /* ' and password=' */-- '
```

Forma ideal:

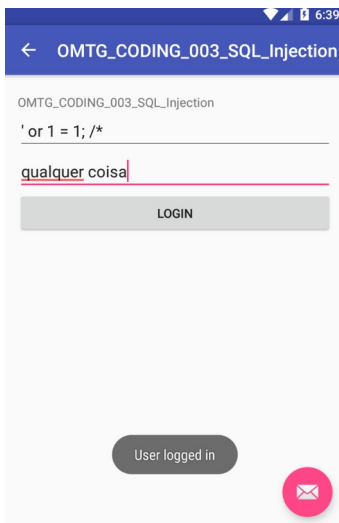
Note que a requisição da senha ficou comentada, de tal forma que não será necessário enviá-la

```
select user, password from users where user='' or 1=1 /* '' and password='$passwordinput'
```



The screenshot shows a mobile application interface with a blue header bar containing a back arrow and the text "OMTG_CODING_003_SQL_Injection". Below the header, the text "OMTG_CODING_003_SQL_Injection" is repeated. There are two input fields: the first contains the SQL payload "' or 1 = 1; /*" and the second is labeled "Password". Below these fields is a grey "LOGIN" button. At the bottom of the screen, there is a grey button that says "User logged in" and a red circular icon with a white envelope symbol.

Em alguns casos há verificação se o campo password foi preenchido ou não. Neste caso seria necessário preencher o campo password com qualquer palavra. Segue o exemplo abaixo:



This screenshot is similar to the previous one, showing the same mobile application interface. The first input field contains the same SQL payload. The second input field, labeled "Password", now contains the text "qualquer coisa" (any thing). The "LOGIN" button is greyed out. At the bottom, the "User logged in" button and the red envelope icon are visible. The status bar at the top right shows the time as 6:39.

bad encryption, exercicio 2:

usando o jadx, podemos ver o código java utilizado para encriptar a senha

```
/* JADX INFO: Access modifiers changed from: private */
public static boolean verify(String str) {
    byte[] encryptedDecoded = Base64.decode("vJqfip28ioydips=", 0);
    byte[] userPass = encrypt(str);
    if (userPass.length != encryptedDecoded.length) {
        return false;
    }
    for (int i = 0; i < userPass.length; i++) {
        if (userPass[i] != encryptedDecoded[i]) {
            return false;
        }
    }
    return true;
}

private static byte[] encrypt(String str) {
    byte[] bytes = str.getBytes();
    for (int i = 0; i < bytes.length; i++) {
        bytes[i] = (byte) (bytes[i] ^ Ascii.DLE);
        int curr = (~bytes[i]) & 255;
        bytes[i] = (byte) curr;
    }
    return bytes;
}
```

etapa 1:

No caminho de encriptação houve o encode em base 64, neste caso, precisamos fazer um decode de base 64.

```
byte[] encryptedDecoded = Base64.decode("vJqfip28ioydips=", 0);
```

decriptação da etapa 1:

```
te.py / ...
import base64

s='vJqfip28ioydips='
bs=base64.b64decode(s)
```

etapa 2:

houve uma segunda forma de encriptacao:

```
bytes[i] = (byte) (bytes[i] ^ 16);  
bytes[i] = (byte) ((~bytes[i]) & 255);
```

decriptacao da etapa 2

```
for i in base:  
    i = ~i & 255  
    i = i ^ 16  
    print(chr(i),end=' ')
```

codigo completo e resultado final:

```
1 import base64  
2  
3 s = 'vJqfip28ioydips='  
4 base = base64.b64decode(s)  
5  
6 for i in base:  
7     i = ~i & 255  
8     i = i ^ 16  
9     print(chr(i),end='')  
10 print()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
romio@romio-VirtualBox:~/Documents/Python$ /bin/python3 /home/romio/Documents/Python/teste.py  
SuperSecret
```

←

OMTG-DATAST-001-BadEnryption

Bad Encryption

Identify the password. Read the source and try to find out what kind of encryption is used.

SuperSecret

VERIFY PASSWORD

Congratulations, this is the correct password