10-Dynamic Instrumentation

Frida

هي عبارة عن Dynamic tool بنستخدمها علشان نحلل التطبيق من غير ما نعمل decompile للتطبيق يعني مثلا عاوزين نغير في الكود والتطبيق شغال ب Frida بتسمح بكده يبقي كده بتستخدم في

- **reverse engineering**
- **Mobile penetration testing**

Frida contain 2 component

1- server ---> on phone (andorid)

2- client ----> on computer (ubuntu)

برتبط بقي بين server و client باستخدام Python or JavaScript وبكده عرفنا ايه هي الاداة دلوقتي هنشوف ازاي هنزل الاداة

Install frida tool

1- Frida Client (install on ubuntu)

```
➜  ~ pip install frida-tools
```

2- for server install from github and the link of installation https://github.com/frida/frida/releases/download/16.7.14/frida-server-16.7.14-android-x86_64.xz

after install extract it and upload to emulator on /data/local/bin --> because this path use to run binary file

```
➜  Downloads adb push frida-server-16.7\ \(2\).14-android-x86_64
/data/local/tmp
➜  cd /data/local/tmp
➜  chmod +x frida-server-16.7\ \(2\).14-android-x86_64
➜  su
➜  ./frida-server-16.7\ \(2\).14-android-x86_64
```

on ubuntu connect client with server

-U --> to connect to adb

```
➜  Downloads frida-ps -U
```

Start With Frida and Objection

دلوقتي لو عاوزين نغير apk بحيث نغيره علشان نعرف نعدل عليه لما نستخدم frida هنعمل كده باستخدام Objection

```
PS C:\Users\Dell\Downloads> objection patchapk -s .\FridaTarget.apk --
architecture x86
Checking for a newer version of objection...
Usage: objection [OPTIONS] COMMAND [ARGS]...
```

هينشئ تطبيق جديد بس بيسمح اني اعدل عليه باستخدام frida احنا بقي هننزل التطبيق الجديد ده

لو بقي Objection عادي ممكن نستخدم frida من غير ما نفعل **Objection** هو انا احنا هنشغل server وبعد كده باستخدام frida-ps هشنوف ايه هي التطبيقات اللي شغالة بعد كده همحدد التطبيق بتاعنا باستخدام frida

frida server

```
PS C:\Users\Dell> adb shell
generic_x86_arm:/ $ cd data/local/tmp
generic_x86_arm:/data/local/tmp $ ls
android-webview-command-line  chrome-command-line  content-shell-command-
line  frida-server  webview-command-line
generic_x86_arm:/data/local/tmp $ su
generic_x86_arm:/data/local/tmp # ./frida-server
```

frida-ps

```
PS C:\Users\Dell> frida-ps -U
 PID  Name
----  ------------------------------------------------------------
2197  .MultiDisplayServiceReceiver
2201  .MultiDisplayServiceReceiver
```

frida with apk

```
PS C:\Users\Dell> frida -U FridaTarget

     ____
    / _  |   Frida 17.0.7 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
   . . . .
```

```
   . . . .   Connected to Android Emulator 5554 (id=emulator-5554)

[Android Emulator 5554::FridaTarget ]->
```

دلوقتي بقي علشان نتعامل مع التطبيق بنستخدم اما javascript or python بس احنا هنستخدم javascript

javascript code

```
[Android Emulator 5554::FridaTarget ]-> for(var i=0; i < 5; i++) {
console.log(i); }
0
1
2
3
4
```

```
[Android Emulator 5554::FridaTarget ]-> Java.androidVersion
"11"
[Android Emulator 5554::FridaTarget ]->
```

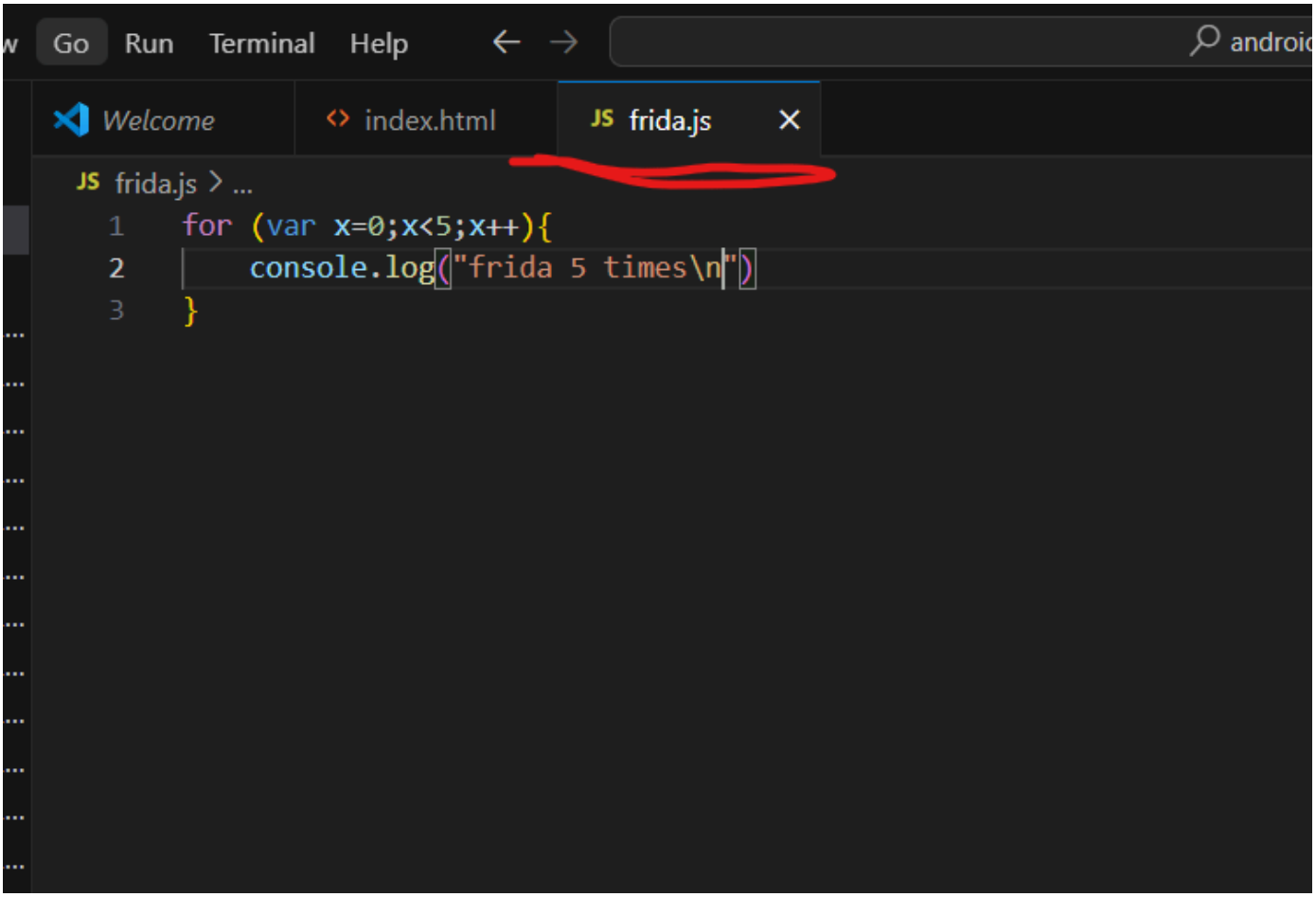
Command	Description
`Interceptor.attach()`	Hook sensitive functions like `send()`, `recv()`, or authentication methods.
`Java.perform()`	Interact with and modify Java methods in Android apps for runtime analysis.
`Java.use()`	Hook into specific classes in Android apps.
`Java.enumerateLoadedClasses()`	List all Java classes loaded into the memory of the target app.
`Module.findExportByName()`	Locate and hook exported functions like `libc` or crypto functions.
`Interceptor.replace()`	Replace the functionality of critical methods like password checks or encryption calls.
`ObjC.choose()`	Interact with live Objective-C objects (iOS apps).
`Stalker.follow()`	Follow the instructions executed by a specific thread for deeper debugging.
`Memory.readUtf8String()`	Read sensitive strings like passwords or tokens from memory.
`Memory.writeUtf8String()`	Overwrite sensitive data in memory, such as tokens or keys.

Command	Description
`Process.enumerateModules()`	List loaded libraries and binaries to identify sensitive modules.
`Process.enumerateThreads()`	View running threads in the app to target specific functionality.

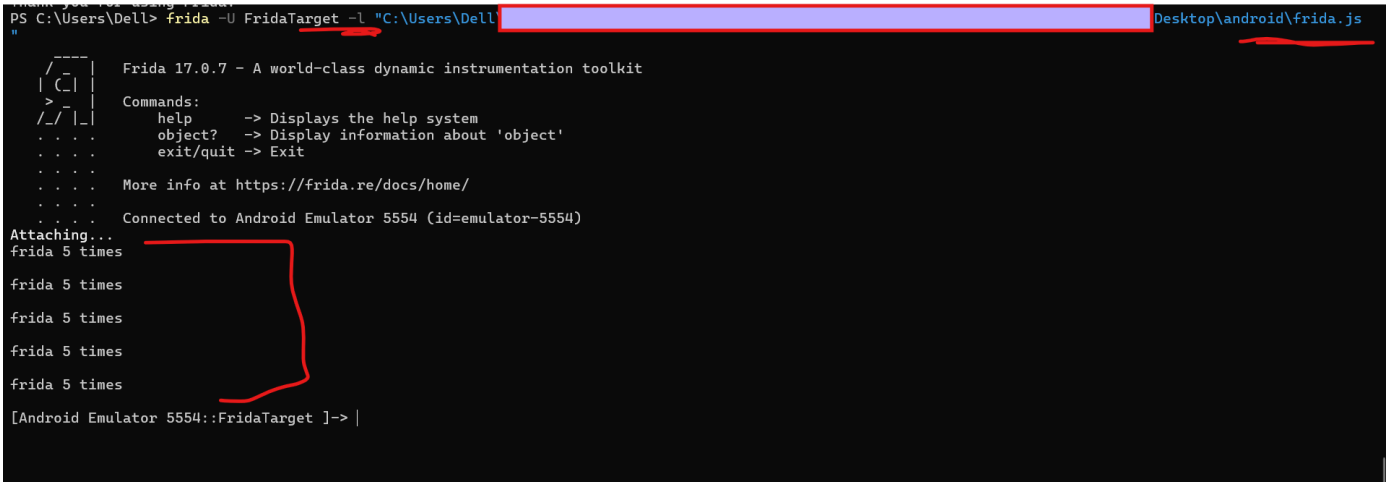
دلوقتي بقي لو عاوزين نستخدم مثلا javascript اما نجيب الكود ومحطه في frida زي ما عملنا Java.androidVersion او نحطه في file ونستخدم -l file.js علشان نستخدم script

ex:

javascript code




```
frida -U FridaTarget -l frida.js
```



لو عاوزين بقي نخليه reload اول ما نغير في الملف : نخليها on

```
%autoreload on/off
```

We can get JavaScript wrappers for Java classes by using `Java.use`:

```
Java.use("java.lang.String")
```

We can then instantiate those classes by calling $new:

```
var string_class = Java.use("java.lang.String");
var string_instance = string_class.$new("Teststring");
string_instance.charAt(0);
```

We can dispose of instances (for example to free up memory) using `$dispose()`, however this is almost never required, as the Garbage Collector should collect unused instances.

We can also replace the implementation of a method by overwriting it on the class:

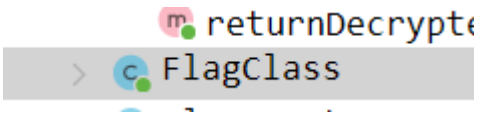
```
string_class.charAt.implementation = (c) => {
    console.log("charAt overridden!");
    return "X";
}
```

Get flag

عندنا activity واحد هو MainActivity

```
<activity android:theme="@style/Theme.FridaTarget.NoActionBar" android:label="@string/app_name" android:name="io.hextree.fridatarget.MainActivity" android:exported=
    <intent-filter>
        <action android:name="android.intent.action.MAIN"/>
        <category android:name="android.intent.category.LAUNCHER"/>
    </intent-filter>
</activity>
```

بعد كده هنروح ل FlagClass



فيه function 3 اتنين من غير parameter واخد ب parameter فدلوقتي هستدعي التلاتة باستخدام javascript

بس هنستخدم .$new() علشنان تبقي constructor يتنفذ اول مما نستدعي class

code

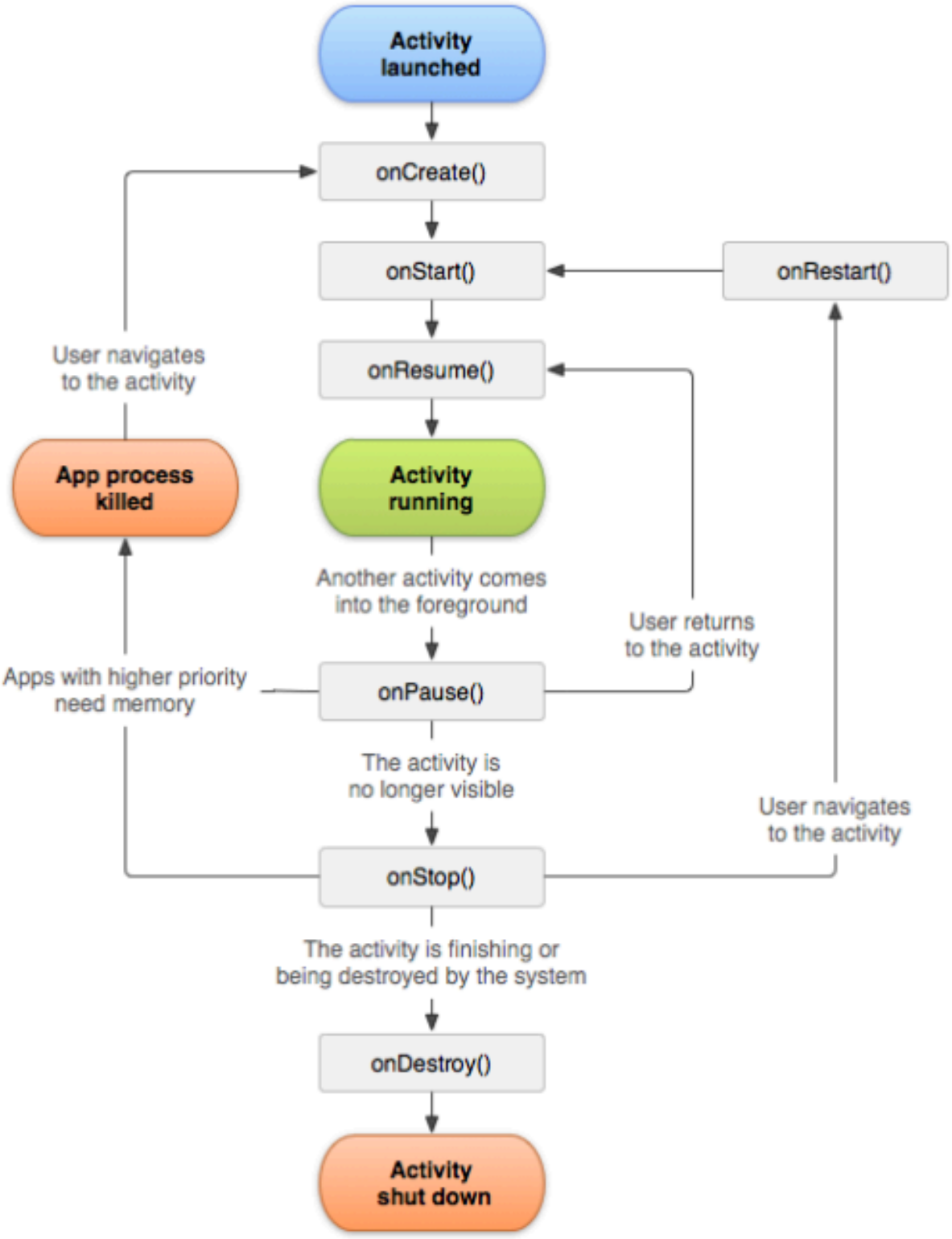
```
Java.perform(()=>{
 let flag_class=Java.use("io.hextree.fridatarget.FlagClass");
 let flag=flag_class.$new();
 console.log(flag.flagFromStaticMethod());
 console.log(flag.flagFromInstanceMethod());
 let password="sesame";
 console.log(flag.flagIfYouCallMeWithSesame(password));
})
```

هناخد الكود ده ونحطه في frida

flag 1 --> HXT{a-static-calling-with-frida}

flag 2 --> HXT{dynamic-droid}

flag 3 --> HXT{the-droid-youre-looking-for}



الـ `onResume()` داخل الـ Activity Lifecycle:

إلى الحالة "الظاهرة" أو "الواجهة"، أي عندما يعود النشاط إلى (Activity) هي واحدة من الدوال التي تُنفذ عند انتقال النشاط `onResume()` التي تُنفذ عندما يتم إيقاف النشاط مؤقتًا، وتُنفذ قبل دالة `onPause()` الواجهة بعد أن كان في الخلفية أو تم إيقافه مؤقتًا. هي تأتي بعد دالة `onStop()` في حالة توقف النشاط بالكامل.

شرح دورة الحياة بالتفصيل:

1. **`onCreate()`**: تُنفذ عند إنشاء النشاط لأول مرة. تستخدم هذه الدالة غالبًا لإعداد واجهة المستخدم وربط العناصر داخل النشاط.
2. **`onStart()`**: تُنفذ عندما يتم جعل النشاط مرئيًا للمستخدم، ولكنه قد لا يكون في الواجهة بالكامل (ممكن أن يكون في الخلفية مع وجود أنشطة أخرى أمامه).
3. **`onResume()`**: تُنفذ عندما يصبح النشاط في الواجهة ويتم التفاعل معه من قبل المستخدم. يتم استئناف أي عمليات متوقفة أو تنفيذ عمليات تحتاج إلى أن تتم عندما يصبح النشاط في الواجهة. مثلًا:
   - تحديث واجهة المستخدم.
   - استئناف عمليات الشبكة أو الوسائط (مثل تشغيل الفيديو).
   - تفعيل مستشعرات معينة مثل الـ GPS أو الكاميرا.

   في هذه المرحلة، يكون النشاط جاهزًا للتفاعل مع المستخدم.
4. **`onPause()`**: تُنفذ عندما يتم إيقاف النشاط مؤقتًا، ولكن لا يزال مرئيًا. قد تكون هذه الدالة مفيدة لحفظ بيانات مؤقتة، أو إيقاف بعض العمليات التي لا تحتاج إلى الاستمرار أثناء انشغال المستخدم مع تطبيق آخر.
5. **`onStop()`**: تُنفذ عندما يصبح النشاط غير مرئي تمامًا، سواء تم نقل المستخدم إلى تطبيق آخر أو تم إغلاق النشاط. في هذه المرحلة، يجب إيقاف أي عمليات تحتاج إلى موارد مستمرة.
6. **`onRestart()`**: تُنفذ إذا عاد النشاط إلى الواجهة بعد أن كان في الخلفية، وعادة ما يتم استدعاؤها بعد `onStop()` وقبل `onStart()`.
7. **`onDestroy()`**: تُنفذ عندما يتم تدمير النشاط، سواء كان بسبب إغلاق التطبيق أو بسبب تغيير في الإعدادات.

مثال توضيحي لاستخدام `onResume()`:

افترض أنك تقوم بتطوير تطبيق يعرض الأخبار، وعند التبديل بين الأنشطة (مثل الانتقال من شاشة لعرض مقال إلى شاشة رئيسية)، قد ترغب في تحديث محتوى المقال عندما يعود المستخدم إلى شاشة المقال (Activity). يمكن فعل ذلك داخل `onResume()`:

```
@Override
protected void onResume() {
    super.onResume();
    // هنا يمكنك إعادة تحميل المقالات أو تحديث المحتوى عند العودة للنشاط
    updateContent();
    Log.d("Activity Lifecycle", "Activity Resumed");
}
```

لماذا نستخدم `onResume()`؟

- **استئناف العمليات**: مثل استئناف تشغيل الفيديو أو الصوت.
- **تحديث الواجهة**: إعادة تحميل البيانات أو تحديث UI إذا كانت قد تغيرت في الخلفية.
- **التفاعل مع المستخدم**: تفعيل أي مكونات تفاعلية لمتابعة تقديم تجربة المستخدم بشكل سلس.
- **التعامل مع الموارد**: مثل استئناف المستشعرات أو الاتصال بالشبكة إذا تم إيقافها في `onPause()`.

باختصار، `onResume()` هي دالة تُستخدم لإدارة الحالة عند عودة النشاط إلى الواجهة، وتتيح لك التفاعل مع المستخدم أو استئناف العمليات التي تحتاج إلى أن تكون نشطة فقط عندما يكون النشاط في الواجهة.

Frida-trace

بسنتخدمها لو عاوزين ننتبع function او حاجة معينة مثلا عاوزين نعرف لما ندوس علي button ايه هو ده او لما نستخدم function نعرف ايه هي وكده

علشنا نتسخدمه بيبقي syntax بتاعه Class-Name ! Function-Name ونستخدم معاها option --> -j

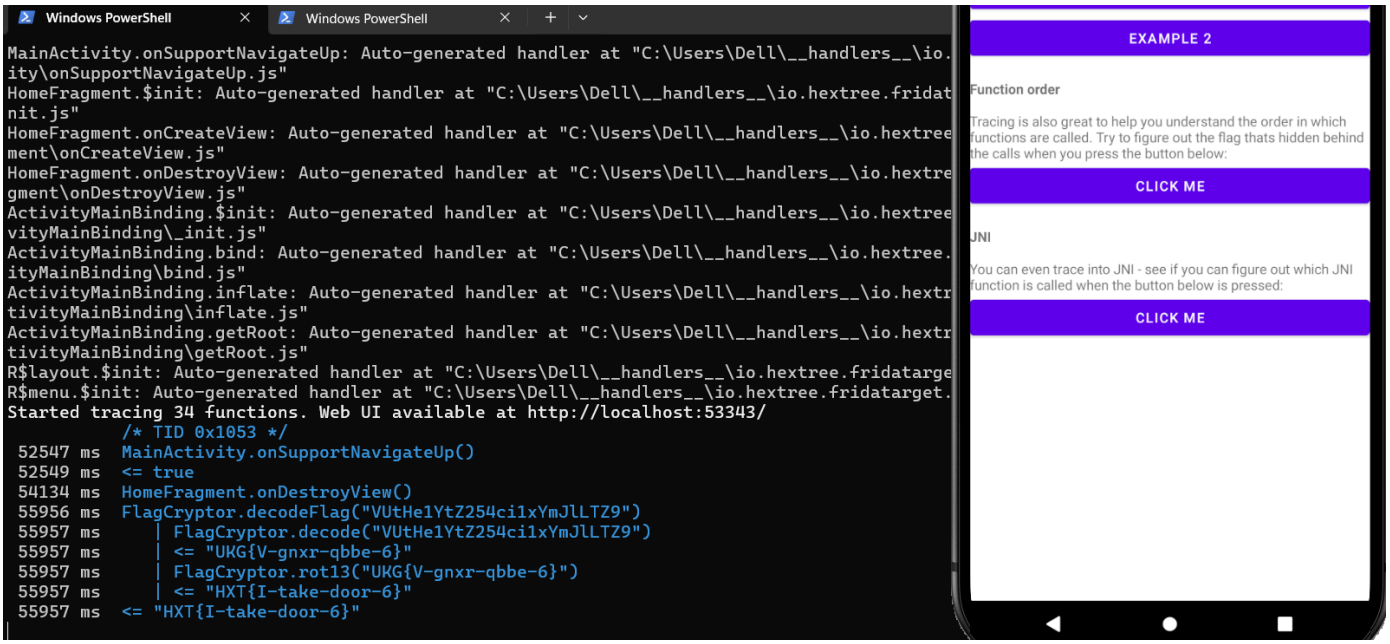
if we want to specific all function within class (io.hextree) --> -j "io.hextree.*!*"

if we want to mathc all function contain password --> -j "io.hextree.*!*password*"

هنا اهو جاب كل functions

```
PS C:\Users\Dell> frida-trace -U -j 'io.hextree.*!*' FridaTarget
Instrumenting...
AppBarMainBinding.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.AppBarMainBinding\_init.js"
AppBarMainBinding.bind: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.AppBarMainBinding\bind.js"
AppBarMainBinding.inflate: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.AppBarMainBinding\inflate.js"
AppBarMainBinding.getRoot: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.AppBarMainBinding\getRoot.js"
FragmentHomeBinding.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.FragmentHomeBinding\_init.js"
FragmentHomeBinding.bind: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.FragmentHomeBinding\bind.js"
FragmentHomeBinding.inflate: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.FragmentHomeBinding\inflate.js"
FragmentHomeBinding.getRoot: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.FragmentHomeBinding\getRoot.js"
R$id.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.R_id\_init.js"
FlagClass.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagClass\_init.js"
FlagClass.flagFromStaticMethod: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagClass\flagFromStaticMethod.js"
FlagClass.flagFromInstanceMethod: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagClass\flagFromInstanceMethod.js"
FlagClass.flagIfYouCallMeWithSesame: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagClass\flagIfYouCallMeWithSesame.js"
FlagCryptor.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagCryptor\_init.js"
FlagCryptor.decode: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagCryptor\decode.js"
FlagCryptor.decodeFlag: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagCryptor\decodeFlag.js"
FlagCryptor.rot13: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.FlagCryptor\rot13.js"
HomeViewModel.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ui.home.HomeViewModel\_init.js"
ExampleClass.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ExampleClass\_init.js"
ExampleClass.returnDecryptedString: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ExampleClass\returnDecryptedString.js"
ExampleClass.returnDecryptedStringIfPasswordCorrect: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ExampleClass\returnDecrypt
edStringIfPasswordCorrect.js"
MainActivity.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.MainActivity\_init.js"
MainActivity.onCreate: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.MainActivity\onCreate.js"
MainActivity.onCreateOptionsMenu: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.MainActivity\onCreateOptionsMenu.js"
MainActivity.onSupportNavigateUp: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.MainActivity\onSupportNavigateUp.js"
HomeFragment.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ui.home.HomeFragment\_init.js"
HomeFragment.onCreateView: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ui.home.HomeFragment\onCreateView.js"
HomeFragment.onDestroyView: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.ui.home.HomeFragment\onDestroyView.js"
ActivityMainBinding.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.ActivityMainBinding\_init.js"
ActivityMainBinding.bind: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.ActivityMainBinding\bind.js"
ActivityMainBinding.inflate: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.ActivityMainBinding\inflate.js"
ActivityMainBinding.getRoot: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.databinding.ActivityMainBinding\getRoot.js"
R$layout.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.R_layout\_init.js"
R$menu.$init: Auto-generated handler at "C:\Users\Dell\__handlers__\io.hextree.fridatarget.R_menu\_init.js"
Started tracing 34 functions. Web UI available at http://localhost:53343/
```

لما احدد حاجة بقي معينة



We can also trace into native objects, by specifing the `-I` option:

```
frida-trace -U -I 'libhextree.so' -j 'io.hextree.*!*' FridaTarget
```

We can use Frida to intercept function calls and return values. For example, to replace the return value of InterceptionFragment.function_to_intercept, we can just write a simple script:

```
Java.perform(() => {
    var InterceptionFragment =
Java.use("io.hextree.fridatarget.ui.InterceptionFragment");
    InterceptionFragment.function_to_intercept.implementation =
function(argument) {
        this.function_to_intercept(argument);
        return "SOMETHING DIFFERENT";
    }
})
```

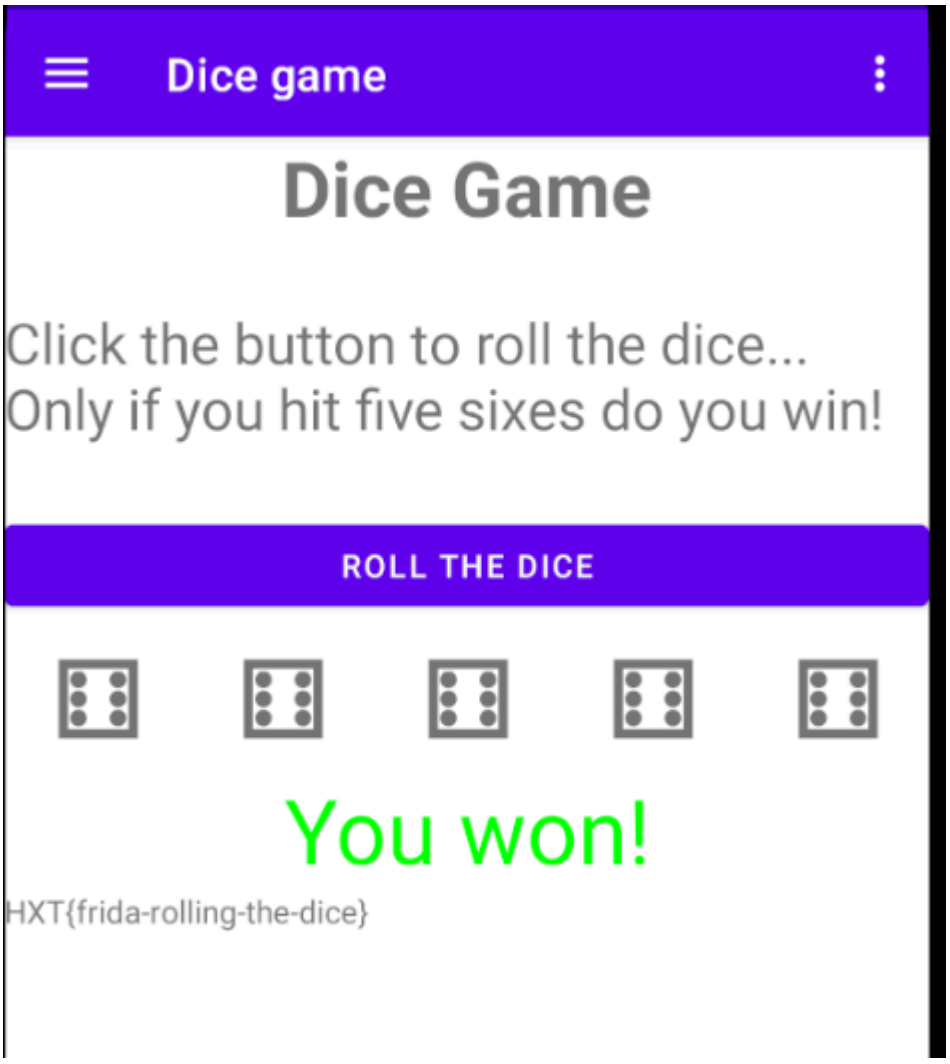
here he check the number return from randomDice() function if he=5 i win else i lose

```java
    public int randomDice() {
        Random random = new Random();
        int randomNumber = random.nextInt(6);
        return randomNumber;
    }

    public void rollDice() {
        boolean won = true;
        for (int i = 0; i < 5; i++) {
            TextView v = (TextView) getView().findViewById(this.diceViewMapping[i]);
            int dice = randomDice();
            if (dice != 5) {
                won = false;
            }
            v.setText(this.diceMapping[dice]);
        }
```

code

```
Java.perform(() => {
    var DiceGameFragment =
Java.use("io.hextree.fridatarget.ui.DiceGameFragment");
    DiceGameFragment.randomDice.implementation = function() {
        this.randomDice();
        return 5;
    }
})
```



With Frida we can often easily disable SSL validation or disable SSL pinning. For example, to bypass Network Security Config and SSLContext based pinning, we can use this simple script:

```
Java.perform(() => {
    var PlatformClass = Java.use("com.android.org.conscrypt.Platform");

PlatformClass.checkServerTrusted.overload('javax.net.ssl.X509TrustManager',
'[Ljava.security.cert.X509Certificate;', 'java.lang.String',
'com.android.org.conscrypt.AbstractConscryptSocket').implementation =
function() {
        console.log("Check server trusted");
    }
})
```

For OKHTTP3-based pinning we need to combine the script of the previous video, and a small new one that prevents the SSLPinner from being added to the OkHttpClient:

```
Java.perform(() => {
    var BuilderClass = Java.use("okhttp3.OkHttpClient$Builder");
```

```
    BuilderClass.certificatePinner.implementation = function() {
        console.log("Certificate pinner called");
        return this;
    }
})
```