LABORATORIO PER L'INFORMATICA LIBERA SANNITA ONLUS

Deep dive into mobile app

scopri se un’app ti spia

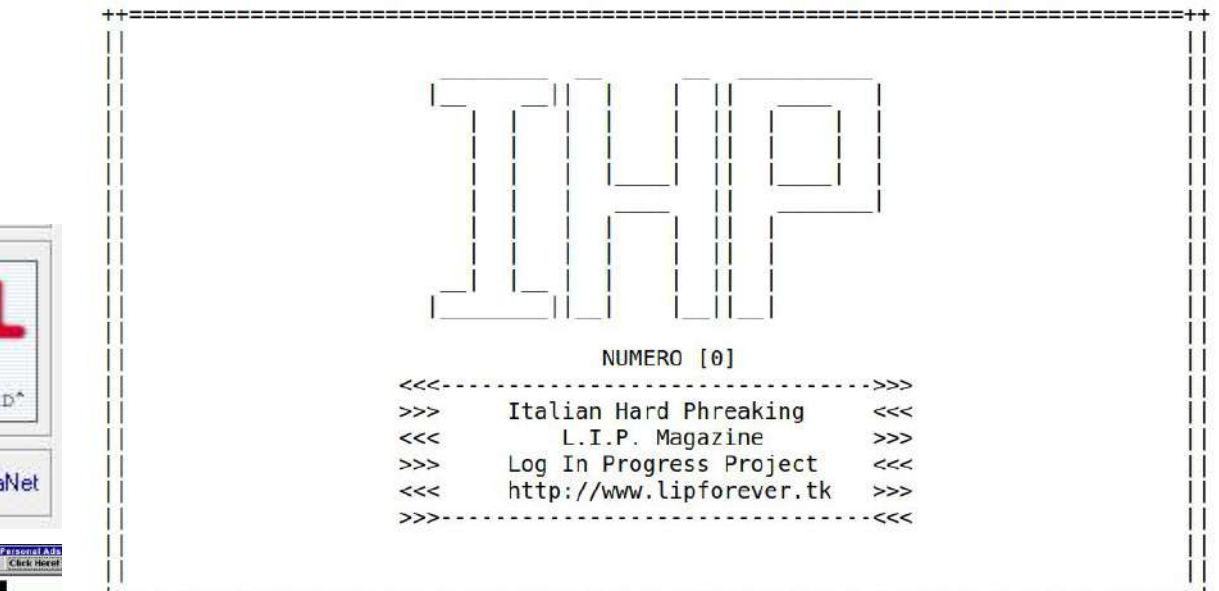
pasquale ‘sid’ fiorillo

> whoami

- Penetration Tester @ ISGroup
- IT Consultant @ rooters >_
- Teacher @ IFTS - Scuola la Tecnica
- Independent Security Researcher
- Reverse Engineering addicted
- Hacking enthusiast

LINUX DAY

> cat /underground



INDICE:

cat /security-researcher

The vulnerability allows any local user, such as “httpdusr” used to run web application, to escalate to Domain Administrator if the NAS is a domain member.

Pasquale ‘sid’ Fiorillo from ISGroup (www.isgroup.biz), an Italian Security Company, and Guido ‘go’ Oricchio of PCego (www.pcego.com), a System Integrator, have just released a critical security advisory for any version of QNAP NAS prior to 4.2.4 Build 20170313 (https://www.qnap.com/en/support/con_show.php?cid=113).

The vulnerability allows a local unprivileged user of a Windows guest to gain Local and/or Domain Administrator access when VeeamVixProxy is active, the de-facto default in VMWare and Hyper-V environments.

Pasquale `sid` Fiorillo, Francesco `ascii` Ongaro from ISGroup, an Italian Security firm, and Antonio `s4tan` Parata from ush team, have just released a critical security advisory for any version of Veeam Backup & Replication prior to 8 Update 3 (released today, October 8th, 2015).

Veeam Software provides backup, disaster recovery and virtualizationmanagement software for the VMware and Hyper-V environments. The ISGroup team has discovered this 0day in the Veeam Software while performing a Penetration Test for a customer.

"The vulnerability allows a local unprivileged user of a Windows guest to gain Local and/or Domain Administrator access when VeeamVixProxy is active, the de-facto default in VMWare *and Hyper-V environments." states the advisory.*

The issue potentially involves 157,000 customers and 9.1 million Virtual Machines worldwide and could lead to full Domain Administrator compromise of the affected infrastructures.

```
sid@zen:~/veeam$ cat VeeamVixProxy_16072015.log | grep "01/07/2015 1.33.42" | cut
-d ' ' -f 6 | base64 -d | hexdump -C | lolcat
base64: invalid input
00000000  23 00 00 00 0a 00 00 00  56 00 65 00 65 00 61 00  |#.......V.e.e.a.|
00000010  6d 00 55 00 73 00 65 00  72 00 10 00 00 00 55 00  |m.U.s.e.r.....U.|
00000020  32 00 56 00 6a 00 63 00  6d 00 56 00 30 00        |2.V.j.c.m.V.0.|
0000002e
sid@zen:~/veeam$ echo -en "U2VjcmV0" | base64 -d | xargs -I {} echo {} | lolcat
Secret
sid@zen:~/veeam$
```

OBJ
OBJ

```
> cat /app/index
```

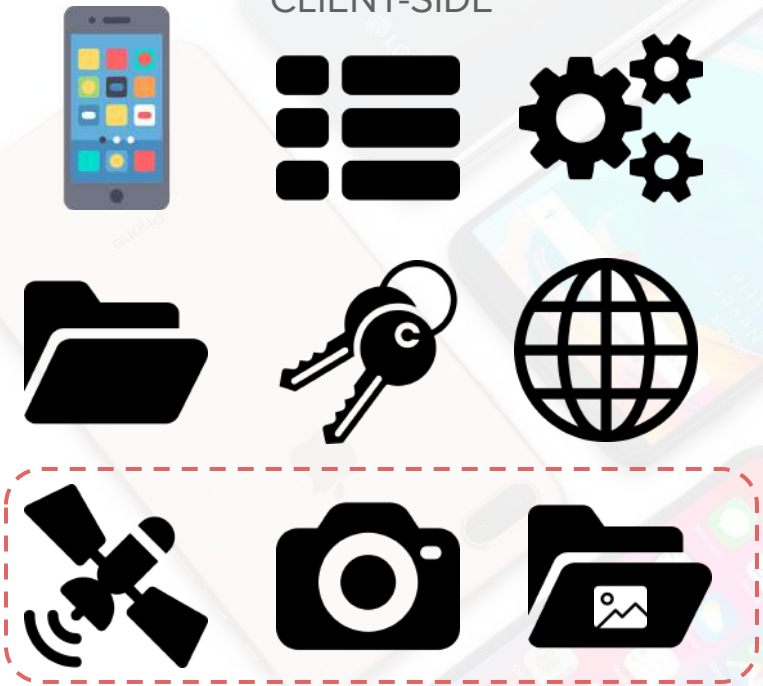
> cat /app/anatomy

CLIENT-SIDE

SERVER-SIDE

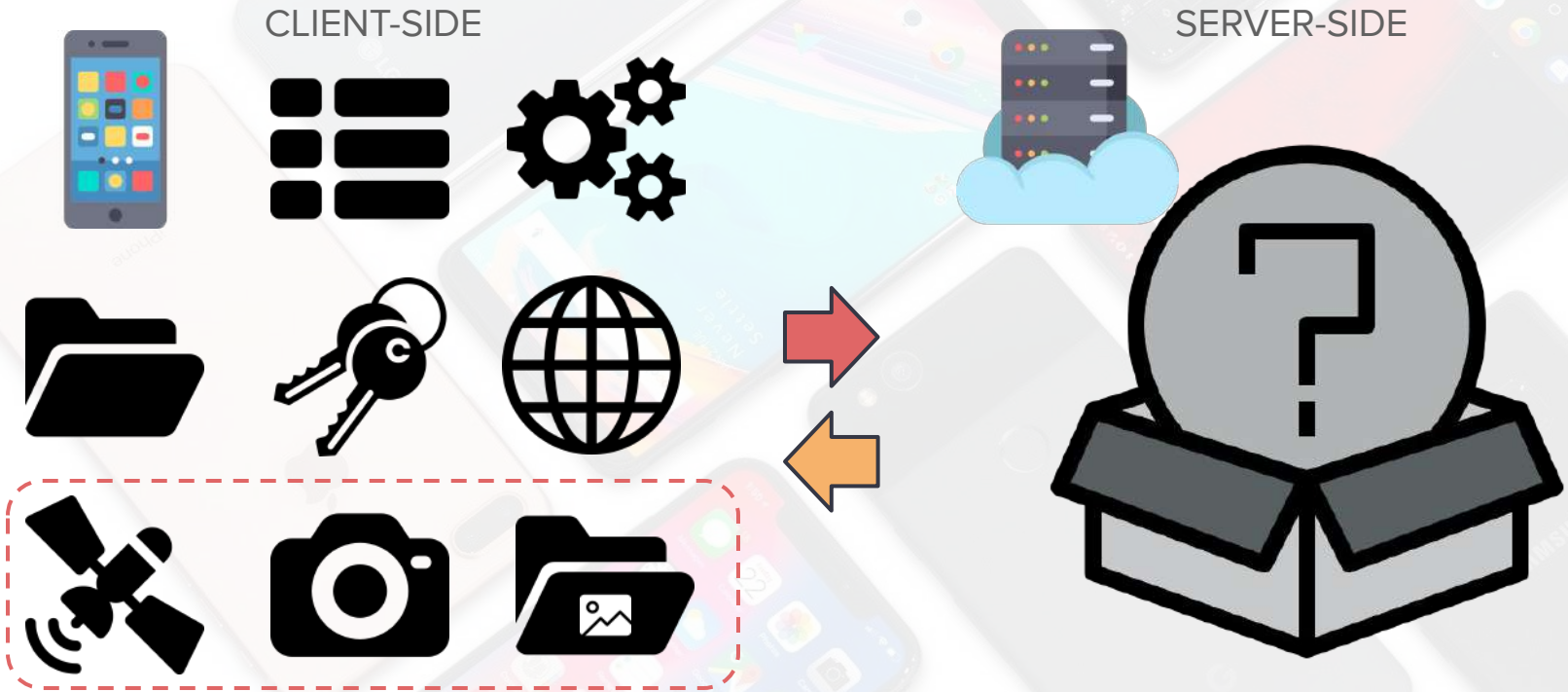
> cat /app/anatomy

CLIENT-SIDE



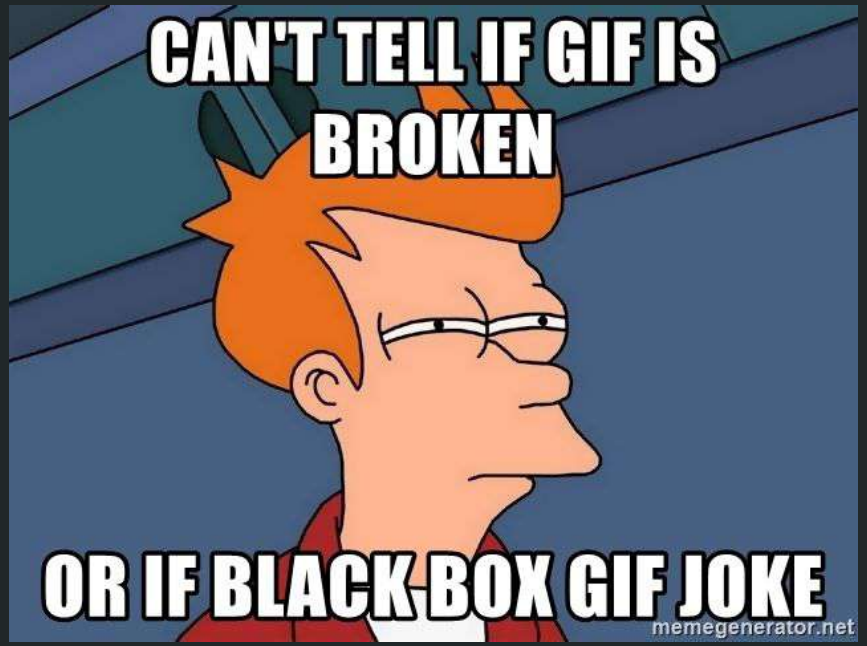
- foreground code
- background code
- internal storage
- keychain
- network access
- sensors, bluetooth, etc
- camera, microphone
- external storage

> cat /app/anatomy

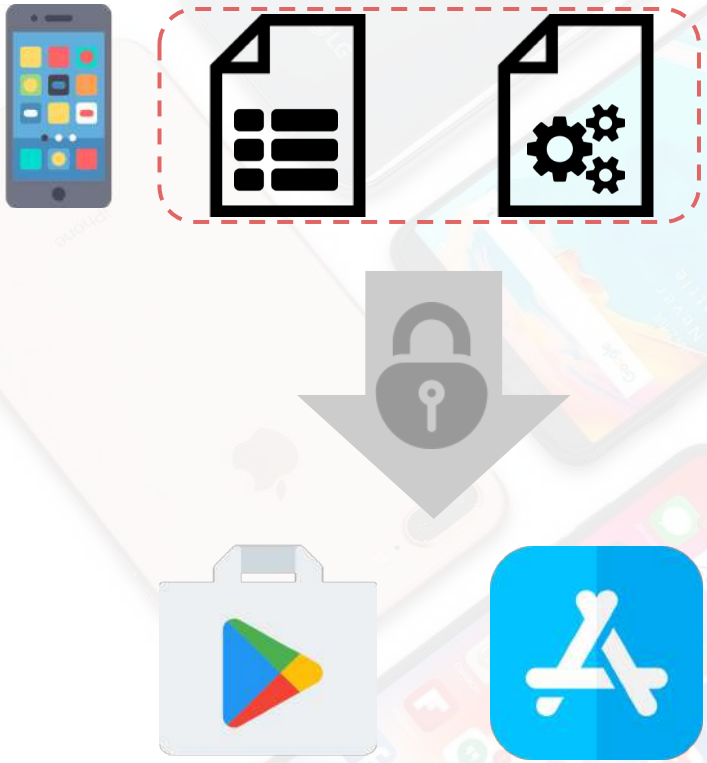


> cat /app/anatomy

SERVER-SIDE

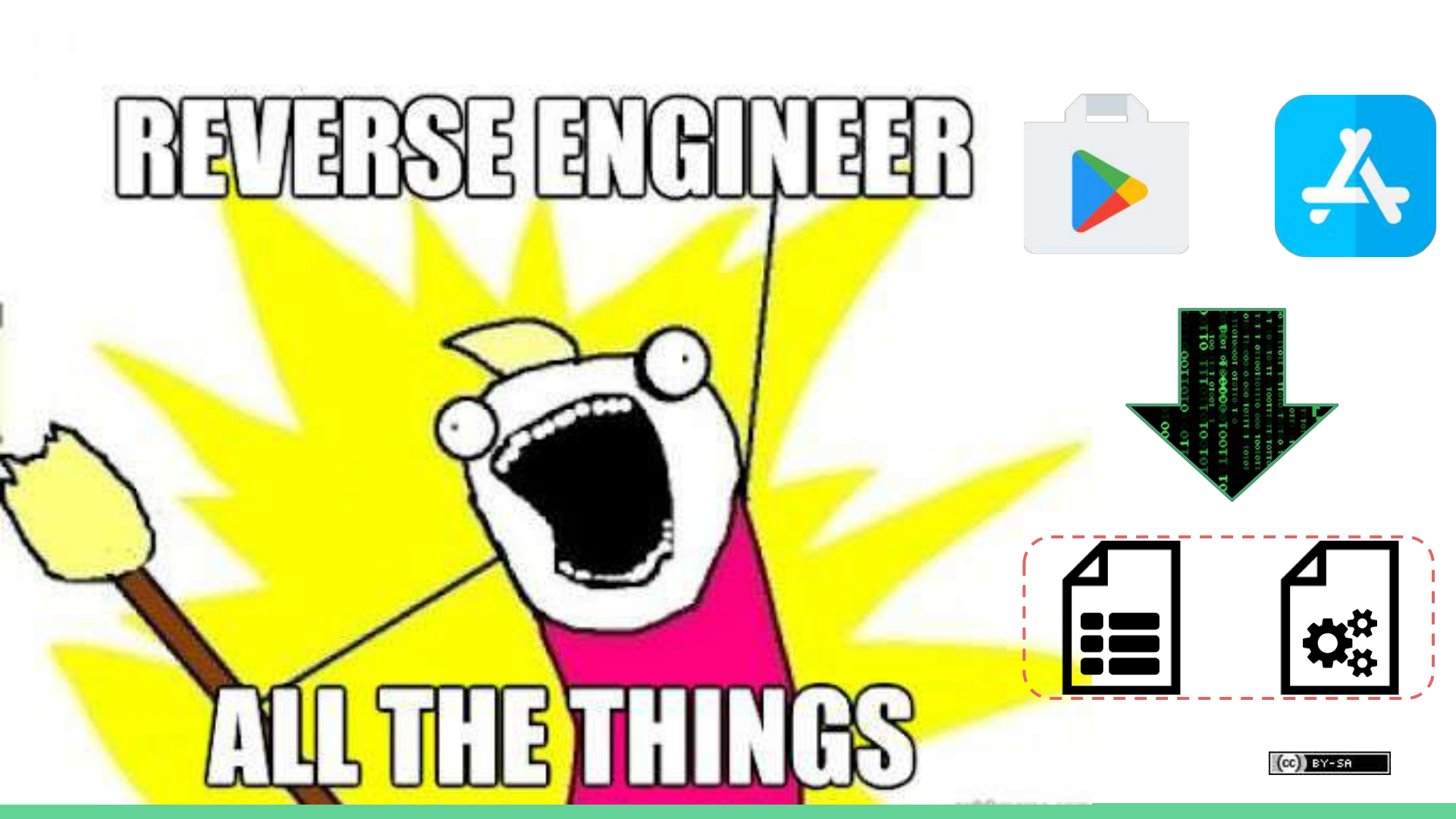


```
> cat /app/package
```



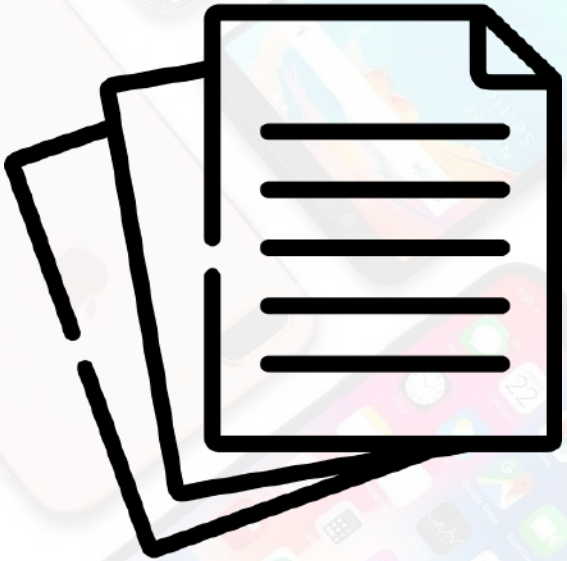
- source code & assets
- build process
- digital signature
- publish

- ipa
- apk/aab



> cat /hack/index

STATIC ANALYSIS



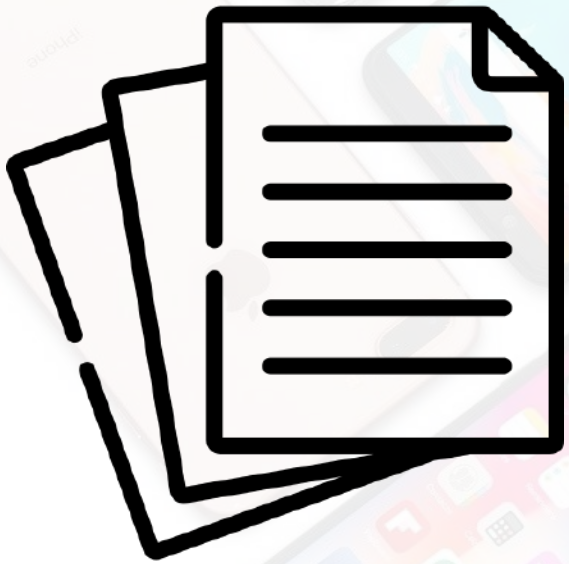
DYNAMIC ANALYSIS

reverse engineering

static analysis

> cat /hack/static

STATIC ANALYSIS

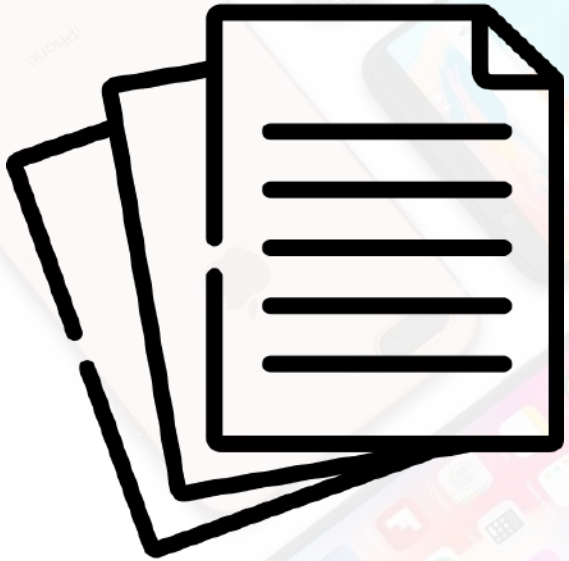


- download/dump the apk
- unpack it
- decode manifest XML
- decompile the code

Android: apk/aab

> cat /hack/static

DOWNLOAD/DUMP THE APK

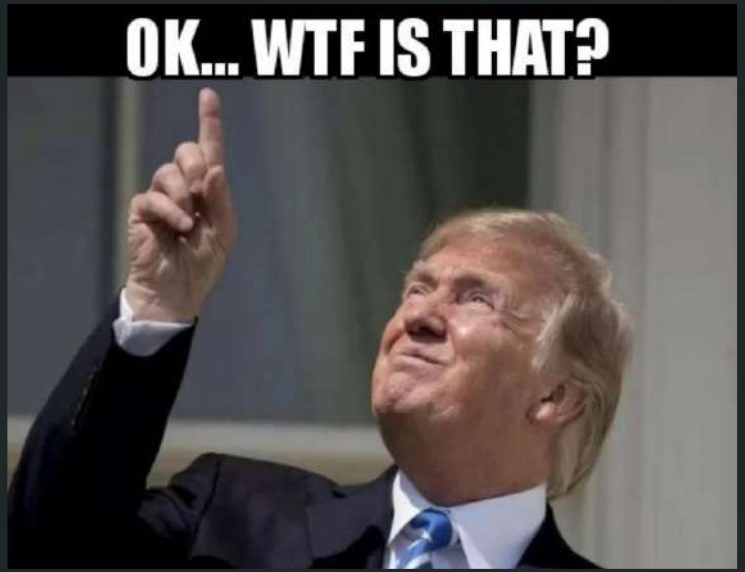


- py googleplay-api
- online downloader
- adb

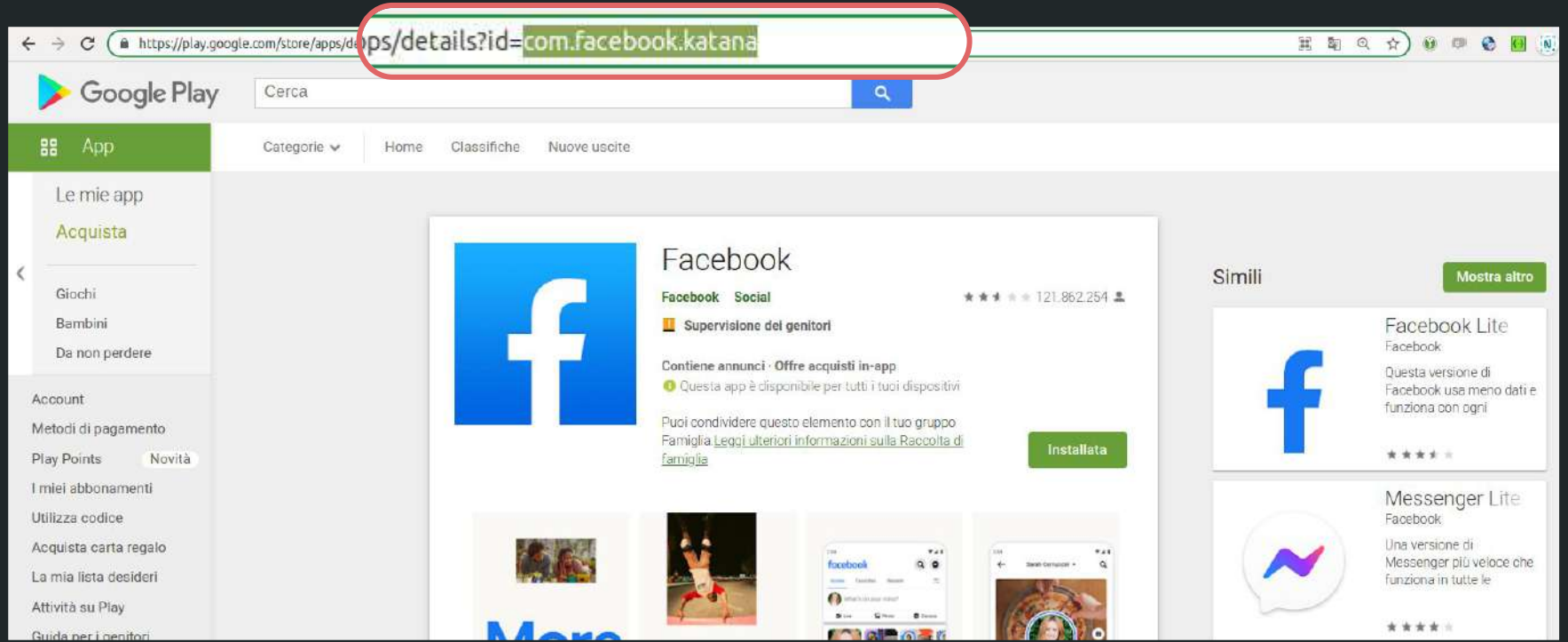
1 obtain the binary package

EXAMPLE: dump the APK from the device with adb

```
$ adb shell pm list packages

$ adb shell pm path com.facebook.katana

$ adb pull /data/app/com.facebook.katana.apk ./
[...]
5185 KB/s (15106048 bytes in 2.844s)
```

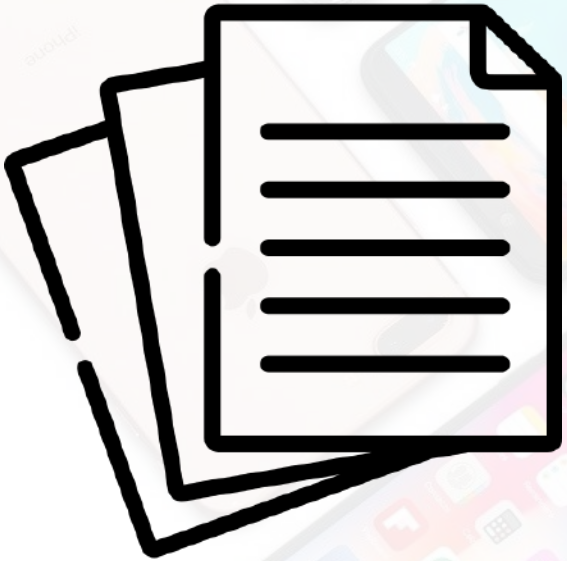


EXAMPLE: getting the package name from Google Play



> cat /hack/static

(UN)PACK THE APK



- apktool
- APK Studio

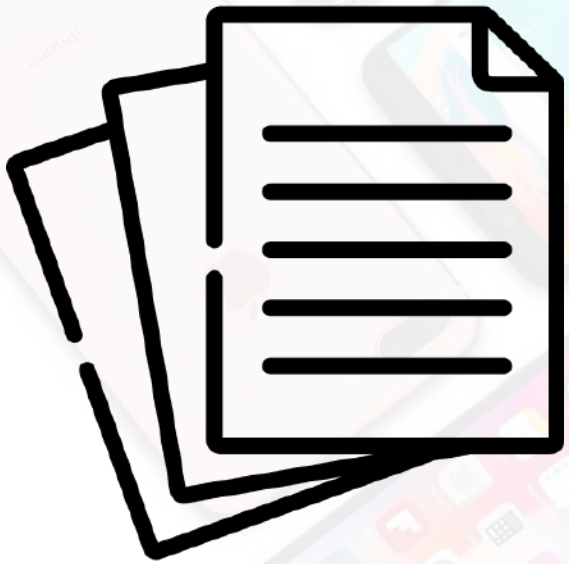
1. obtain the SMALI code
2. obtain the assets
3. obtain AndroidManifest.xml

EXAMPLE: unpack and pack an apk

```
$ apktool d com.facebook.katana.apk

[...]
I: Decoding AndroidManifest.xml with resources...

$ apktool b facebook
[...]
I: Building apk file...
```

> cat /hack/static

AndroidManifest.xml

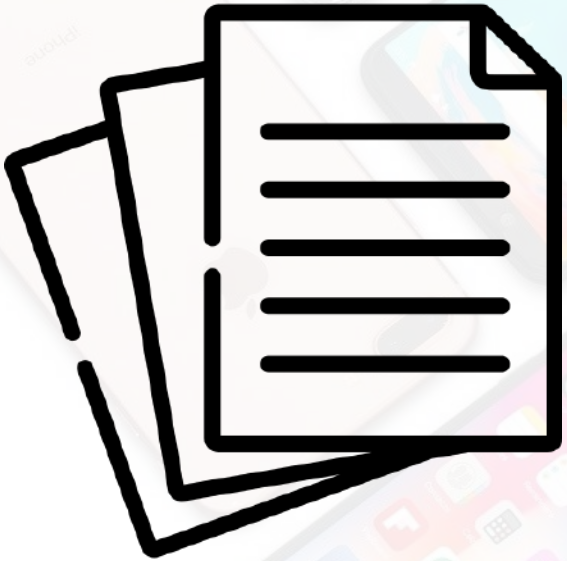


- just a text editor

1. main Activity
2. services
3. permissions
4. ...

> cat /hack/static

DECOMPILE THE SMALI CODE



- jadx (Java native)
- dnSpy / ilSpy (Xamarin C#)
- NOTHING (JS framework)

1 obtain the source code

EXAMPLE: how decompiled code looks like

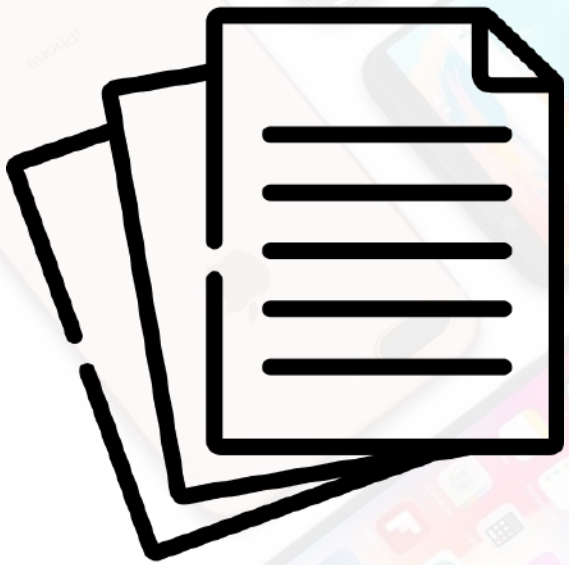
```
14
15      // Token: 0x0600166D RID: 5741 RVA: 0x00081194 File Offset:
          0x0007F394
16      public Position(string deviceId, Location location)
17      {
18          if (location == null)
19          {
20              return;
21          }
22          this.deviceId = deviceId;
23          this.latitude = location.Latitude;
24          this.longitude = location.Longitude;
25          this.altitude = location.Altitude;
26          this.speed = (double)location.Speed;
27          this.course = (double)location.Bearing;
28          if (location.Provider != null && location.Provider != "gps")
29          {
30              this.accuracy = (double)location.Accuracy;
31          }
32          if (Build.VERSION.SdkInt >= 18)
33          {
34              this.mock = location.IsFromMockProvider;
35          }
36      }
```

reverse engineering

static analysis

> cat /hack/static

STATIC ANALYSIS

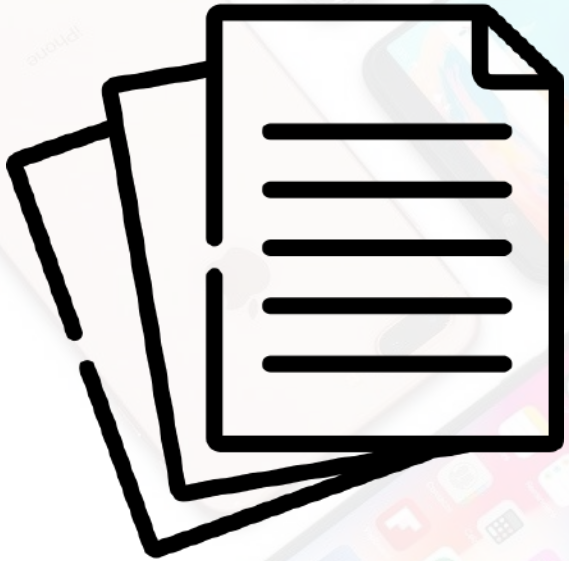


- download/extract the ipa
- unpack it
- decode info.plist
- decompile the code

iOS: ipa

> cat /hack/static

DOWNLOAD/DUMP THE IPA



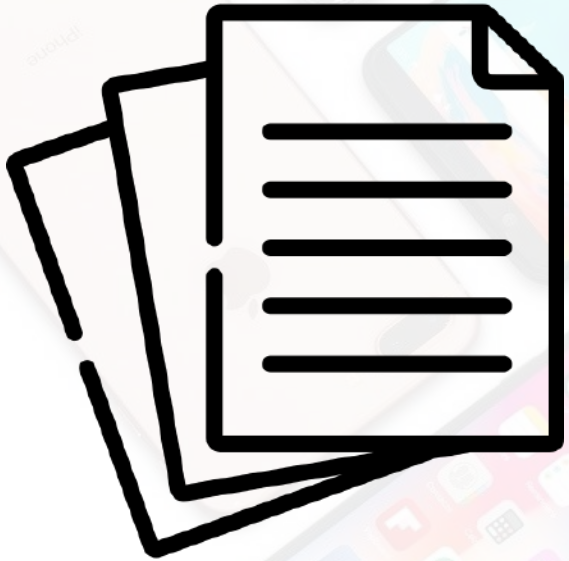
- apple configurator 2
- frida-ios-dump

EXAMPLE: dump the IPA from the device with frida-ios-dump

```
$ ./dump.py Facebook

Start the target app Facebook

Dumping Facebook to ./

[...]

Generating Facebook.ipa

Done.
```

> cat /hack/static

UNPACK THE IPA

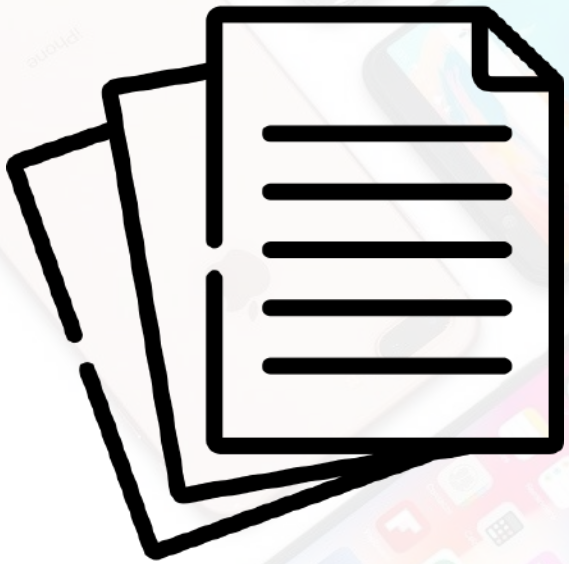


just a zip file

1. obtain the binary code
2. obtain the assets
3. obtain info.plist

> cat /hack/static

info.plist

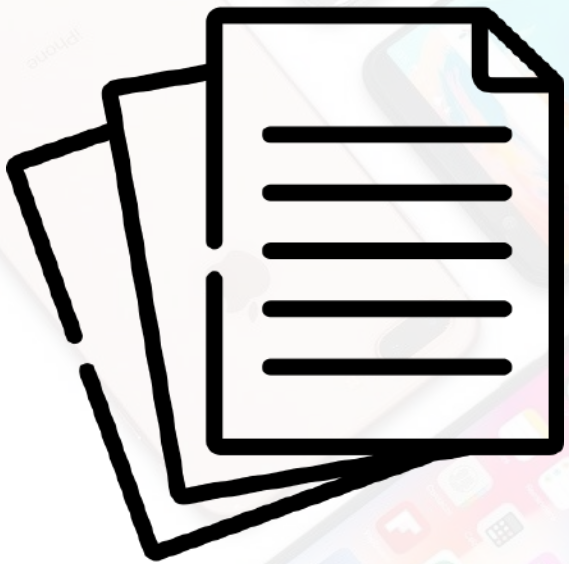


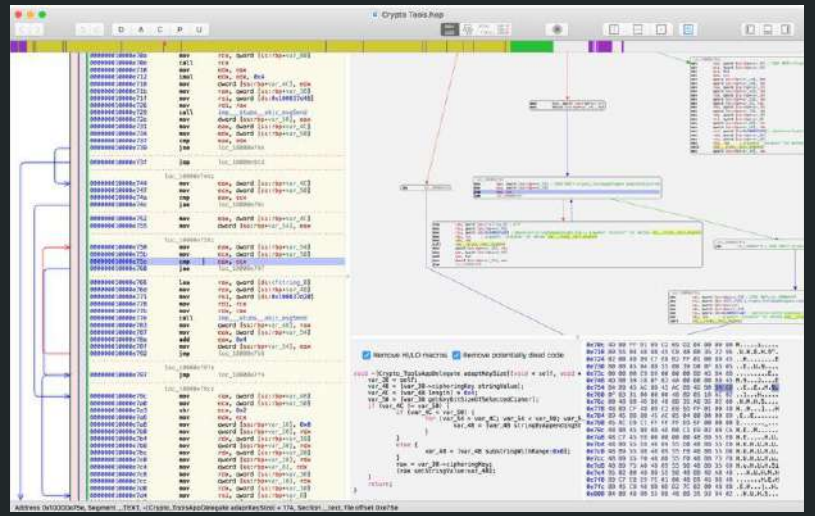
just a text editor

1. some useful app informations
2. supported devices
3. permissions
4. ...

> cat /hack/static

DECOMPILE THE BINARY CODE





- $ Hopper
- $ IDA Pro

1. obtain the assembly code
2. obtain dirty object-c code

reverse engineering

dynamic analysis

> cat /hack/dynamic

CODE INJECTION

FRIDA

```
> cat /hack/dynamic
```

NETWORK MitM

OWASP
Zed Attack Proxy

BURPSUITE

> cat /hack/dynamic

NETWORK MitM

- with an interceptor proxy

you can:

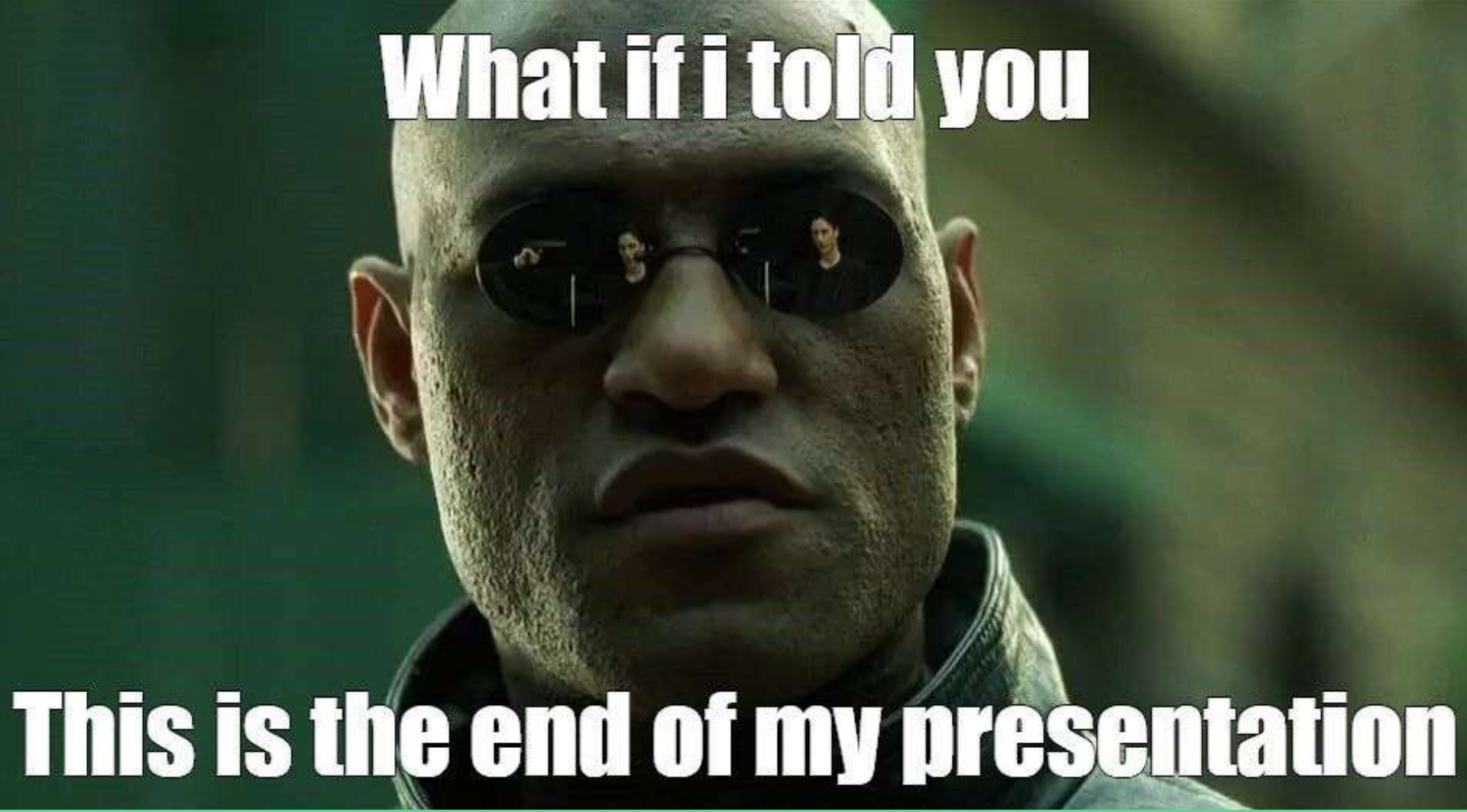
1. sniffing TLS communication
2. intercept and modify data
3. attack the back-end

EXAMPLE: how network communication looks like

Request Response

Pretty Raw Hex \n

```
1 POST /admin/Device/SaveDevice HTTP/2
2 Host: [redacted]
3 Accept: application/json, text/json, text/x-json, text/javascript, application/xml, text/xml
4 User-Agent: Dalvik/2.1.0 (Linux; U; Android 9; [redacted]
5 Content-Type: application/json; charset=utf-8
6 Content-Length: 656
7 Expect: 100-continue
8 Connection: keep-alive
9 Cookie: ASP.NET_SessionId=[redacted]; NID=[redacted]
10 Accept-Encoding: gzip, deflate
11
12 {
   "UserAgent":"Dalvik/2.1.0 (Linux; U; Android 9; [redacted]
   "Size":{
     "WidthPixels":0,
     "HeightPixels":0
   },
   "Oid":0,
   "Identifier":"android-[redacted]",
   "Name":null,
   "Model":null,
   "Serial":null,
   "Latitude":41.14[redacted],
   "Longitude":14.7[redacted]3,
   "Altitude":228.37009742271337,
   "DeviceLocation":{
     "Latitude":41.1[redacted],
     "Longitude":14.[redacted],
     "Altitude":228.37009742271337,
     "Address":", ",
     "City":"Benevento",
     "Province":"Provincia di Benevento",
     "Zip":"82100",
     "Region":"Campania",
     "Nation":"Italia",
     "NationCode":"IT"
   }
```


What if i told you
This is the end of my presentation