# Practical Attacks against Digital Wallets

Date: Apr 2017

**OPCDE/Dubai**

**Yinkozi Middle East**

**Loïc Falletta - Owner & Principal Consultant**

e-mail: loic.falletta@yinkozi.com

mobile: +971 568 348 047

# Who I Am

**LOÏC FALLETTA**
Owner and Principal Security Consultant at Yinkozi

- Review board ENISA (Mobile Security)
- Review board Cloud Alliance (IoT)
- Digital breaker
- Contacts:
  - loic@yinkozi.com
  - Twitter: @zavidan

# Agenda

1. Digital Wallet?
2. Threat Modeling/Attack Surface
3. Security Levels
4. Let’s break it
6. Questions?

45min

# 1 - DIGITAL PAYMENT?

# What do you mean by **Digital Wallet?**

*"They store value in digital form and allow an individual to purchase an item online or send funds to friends or family. Depending on the type of digital wallet used, the information stored might include debit, credit, prepaid or loyalty card data as well as personal information of the card holder such as driver's license, health card, loyalty card(s) and other ID documents. "* - ENISA

*The talk will not cover:*

- *Mobile Wallet such as Android Pay, Apple Pay and Samsung Pay*
- *Digital Currency Wallets*
- *Security of Mobile Payment Providers, Card Issuers, Payment Network Providers, Payment Service, Acquirers,…*

***The talk will cover:***

- ***Custom Digital wallet***
- ***Contactless Payment Communication Technologies: Magnetic Secure Transmission (MST), Near Field Communication (NFC), Quick Recognition (QR) Code, Bluetooth, …***
- ***Perspective of Mobile Device, Mobile Application and Communication Layers***

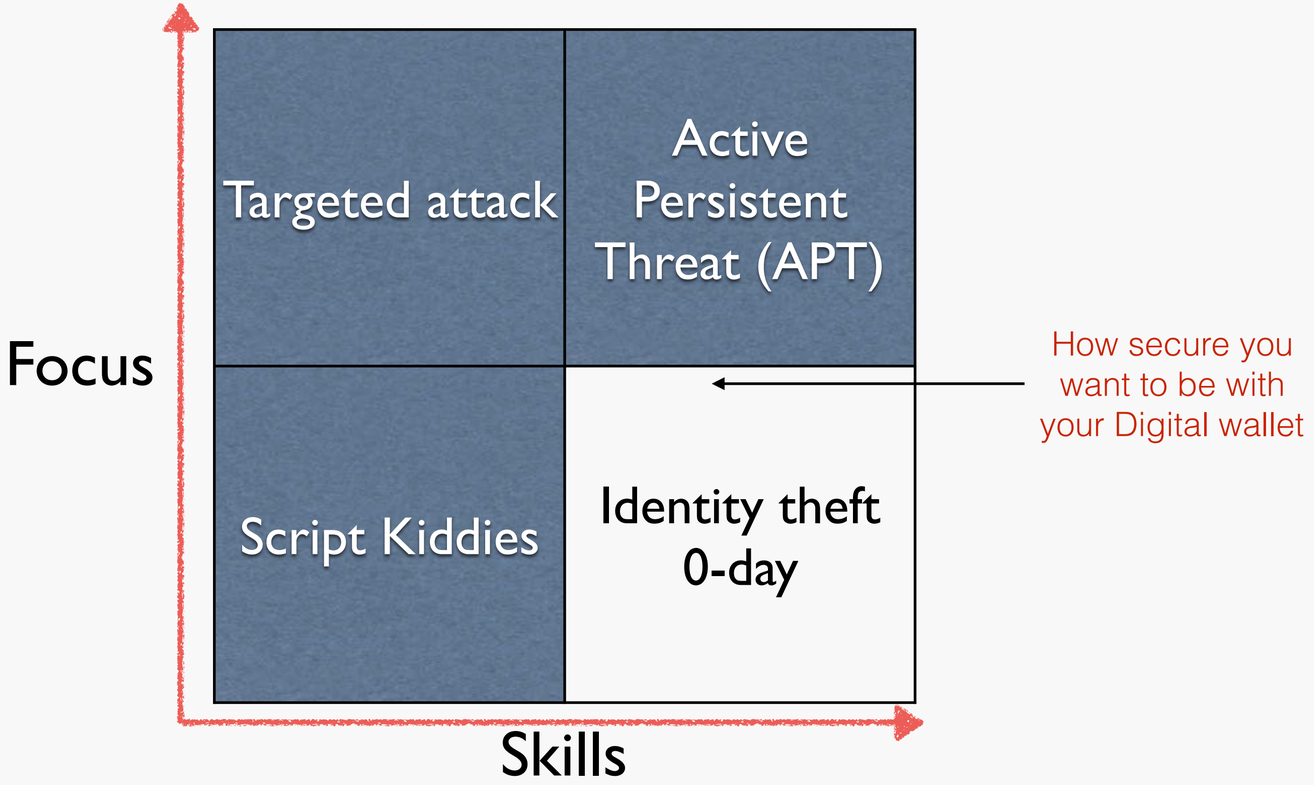
# Threat Modelling / **Attack Surface**

***Script Kiddies:*** *Are you really storing your card's information in plain text?*

***Targeted Attack:*** *Thanks for trusting any Point Of Sale. That makes my life easier.*

***Identify theft, 0-day:*** *I can see that you are doing SSL pinning, too bad that you are using a vulnerable version of openssl.*

***APT:*** *I want you and I will get you, even if I will need a year to do so.*



# Threat Modelling / **Attack Surface**

- *Installation of rogue applications and malware*
- *Unauthorized access to lost or stolen mobile device*
  - *Locked/Unlocked*
  - *Malware installation on the device*
    - *With root permission or not*
- *Reverse engineering*
  - *Static analysis*
  - *Dynamic analysis*
  - *Tampering with the mobile application*
    - *Actively*
    - *Passively (wait for action)*
  - *Mobile application vulnerability*
    - *Race conditions (bad network )*
    - *Static keys*
  - *Compromised third party*
    - *SDK/Contactless POS/Dependency/Device*
  - *Compromised environment*
    - *Relay attacks against NFC enabled POS contactless terminal*
- *Data connectivity compromise*

1 - Social Engineering

2 - Lost or stolen mobile device

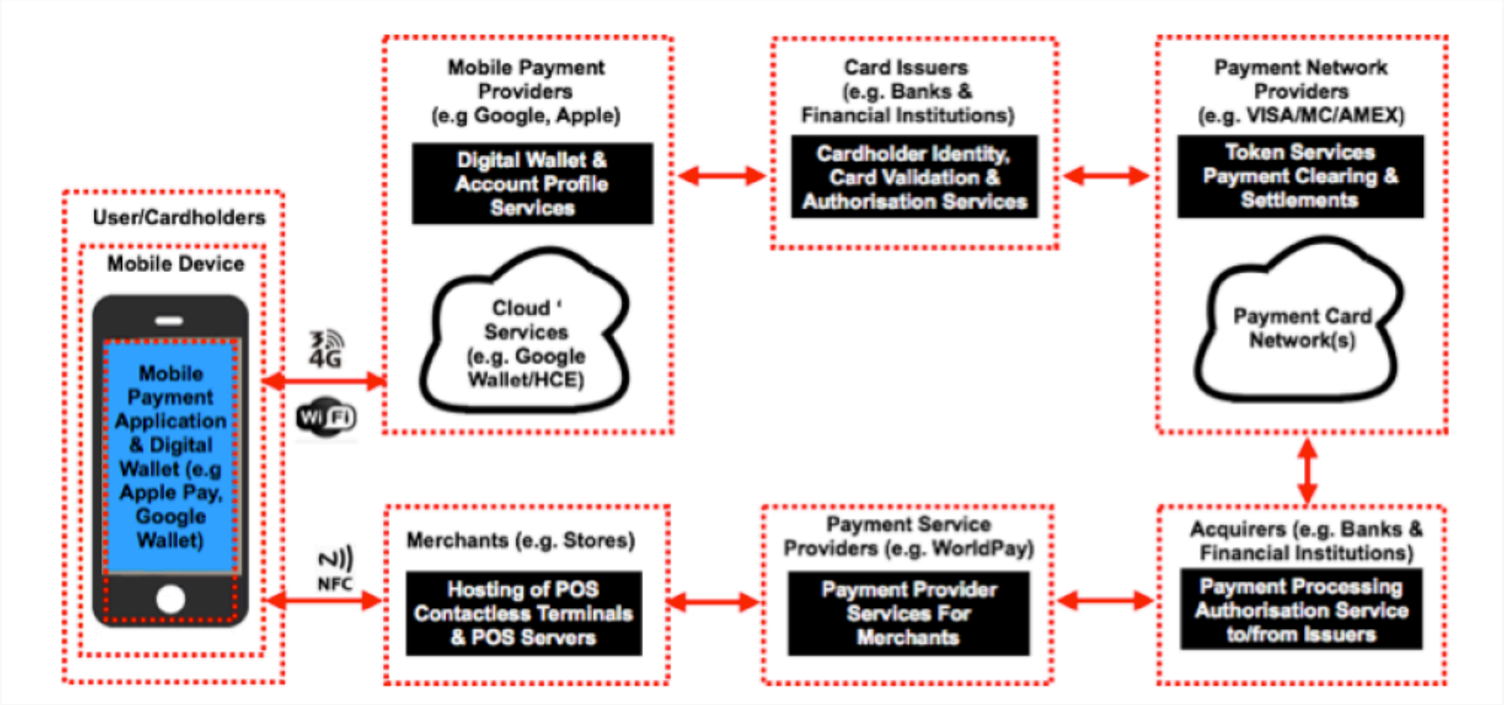
3 - Third party

4 - Environment

# Threat Modelling / **Attack Surface**

Feature Targeted on a Digital Wallet:

- *Enrollment*
- *Credit Card entry*
- *User authentication*
- *Third party trust*
- *Mobile Device configuration*
- *Implementation issues*




source: ENISA

# Threat Modelling / **Attack Surface**

## Paranoid approach for mobile security:

*Be able to protect user's data even if:*

- *The device is compromised*
- *Dependencies are compromised*
- *POS is compromised*
- *Network is compromised*

*Don't forget that:*

- *The application can still be loaded in memory when the malware is loaded*
- *An attacker can wait for you to use the app*

# Security levels **requirements**

## Level 1: Native application

- *SSL pinning  (field based)*
- *Permission enforcement*
- *SQLITE with symmetric encryption*
- *Basic jailbreak/root detection*

***For most apps, this is enough… not for a digital wallet.***

# Security levels **requirements**

## Level 2: Native application and critical checks in native

- *Native application and critical checks in native (like jailbreak/ root detection)*
- *SSL pinning with SSL stack checks*
- *SQLITE with complex encryption*
- *Device linked with the session*

***-Critical checks in native can be easily bypassed (example)***
***-Things are getting serious with SSL pinning. That will slow down most of the attackers. What do you do when the certificate expire…***
***-Linking the device with the session will make it hard to duplicate your phone. But if you don't have a process on mobile banking to approve a phone, this is useless.***
***- Public tools can bypass all these security features***

# Security levels **requirements**

## Example with Root Detection in Native

```
1  -- smali/com/client/android/native/NativeLib.smali --
2  .method public checkRoot()V
3      .locals 5
4      const/4 v4, 0x0
5      invoke-virtual {p0}, Lcom/client/android/native/NativeLib;->check()I
6      move-result v1
7      if-lez v1, :cond_0
```

Check() inside lib_tests.so

- Easy to bypass/reverse if the check is simple
- Easy to bypass if the return value is easy to guess

# Security levels **requirements**

## Level 3: Native application with critical and some business logic in native

- *SSL pinning with native code. Low level checks*
- *SQLITE with complex encryption*
- *Secure container*
- *Device linked with the session*
- *Behavior based on device model*
- *Secure enclave*


- ***If business logic code is written in native. This is a pain for the attacker. He will need more time to understand how the application works***
- ***RSA encryption or others will make it hard to access without having a good understand on how it works***
- ***Once the jailbreak detection is broken, most of the time we still find a way to get infos such as What's in, what's out. And that's enough some time.***
- ***If the device is linked an approval process in performed we can do things. But it's getting harder***
- ***Most of the time integrity checks are performed when the application starts… Root the device, re-link your library in memory and start playing.***

# Security levels **requirements**

## Level 3: Native application with critical and some business logic in native

```
04-26 08:36:24.497 15121 15155 I YKZ  : octopus-com.test.example.android:{"class":"javax.crypto.spec.PBEKeySpec","
method":"javax.crypto.spec.PBEKeySpec","timestamp":1493188584495,"type":"crypto","args":[["6","5","9","e","e","8",
"5","w","e","3","3","9","c","5","1","1","7","c","2","2","8","c","1"
,"f","3","c","a","9","e","e","2","b"],"\n0x00000000 A5 EE 0A 5B 95 13 E2 67 C3 EA C2 AC C2 92 F8 A5
...[...g........\n0x00000010 40 36 B3 FE 47 07 75 A7 62 89 16 F6 3D E5 24 E2 @6..G.u.b...=.$.","10000","256"]}

04-26 08:36:25.000 15121 15155 I YKZ  : octopus-com.test.example.android:{"class":"javax.crypto.Mac","method":"doF
inal","timestamp":1493188585000,"type":"crypto","result":"\n0x00000000 F1 70 47 99 AD 3E 85 9C A1 2D E4 99 87 F1
15 B1 .pG..>...-......\n0x00000010 6C A1 5F 8A A3 17 9F 19 5A C3 22 46 46 CF 17 1D l._.....Z.\"FF..."}

04-26 08:36:25.005 15121 15155 I YKZ  : octopus-com.test.example.android:{"class":"android.content.ContentValues",
"method":"put","timestamp":1493188585005,"type":"globals","args":["DBLOB","\n0x00000000 57 38 40 81 3D 8D 74 A1
0D EA CE FE A1 CC 49 FC W8@.=.t.......I.\n0x00000010 1C 84 95 AE 71 6F B3 30 D8 14 E5 D3 0A 67 45 71
....qo.0.....gEq\n0x00000020 3E 7E DA DF E6 32 E9 B5 89 9C DA 31 12 A9 C4 55 >....2.....1...U\n0x00000030 F1 70
47 99 AD 3E 85 9C A1 2D E4 99 87 F1 15 B1 .pG..>...-......\n0x00000040 6C A1 5F 8A A3 17 9F 19 5A C3 22 46 46 CF
17 1D l._.....Z.\"FF..."]}

04-26 08:36:25.018 15121 15155 I YKZ  : octopus-com.test.example.android:{"class":"android.database.sqlite.SQLiteD
atabase","method":"update","timestamp":1493188585013,"type":"content","args":["DATASTORE_TABLE",{"mValues":{"DBLOB
":{}}},"_ID=?",["1"]],"result":"1"}
04-26 08:36:25.042 15121 15155 I YKZ  : octopus-com.test.example.android:{"class":"javax.crypto.Mac","method":"doF
inal","timestamp":1493188585042,"type":"crypto","result":"\n0x00000000 F1 70 47 99 AD 3E 85 9C A1 2D E4 99 87 F1
15 B1 .pG..>...-......\n0x00000010 6C A1 5F 8A A3 17 9F 19 5A C3 22 46 46 CF 17 1D l._.....Z.\"FF..."}
```

**Public constructors**

`PBEKeySpec(char[] password)`

Constructor that takes a password.

`PBEKeySpec(char[] password, byte[] salt, int iterationCount, int keyLength)`

Constructor that takes a password, salt, iteration count, and to-be-derived key length for generating PBEKey of variable-key-size PBE ciphers.

`PBEKeySpec(char[] password, byte[] salt, int iterationCount)`

Constructor that takes a password, salt, iteration count for generating PBEKey of fixed-key-size PBE ciphers.

- Reversing is not always required
- Event Based on native methods works just fine
- All you need is the input/ output
- FYI: Octopus is a fork of DroidMon

# Security levels **requirements**

## Level 3++: Level 3 + obsfucation, Strong integrity checks, Secure container with anti-tampering

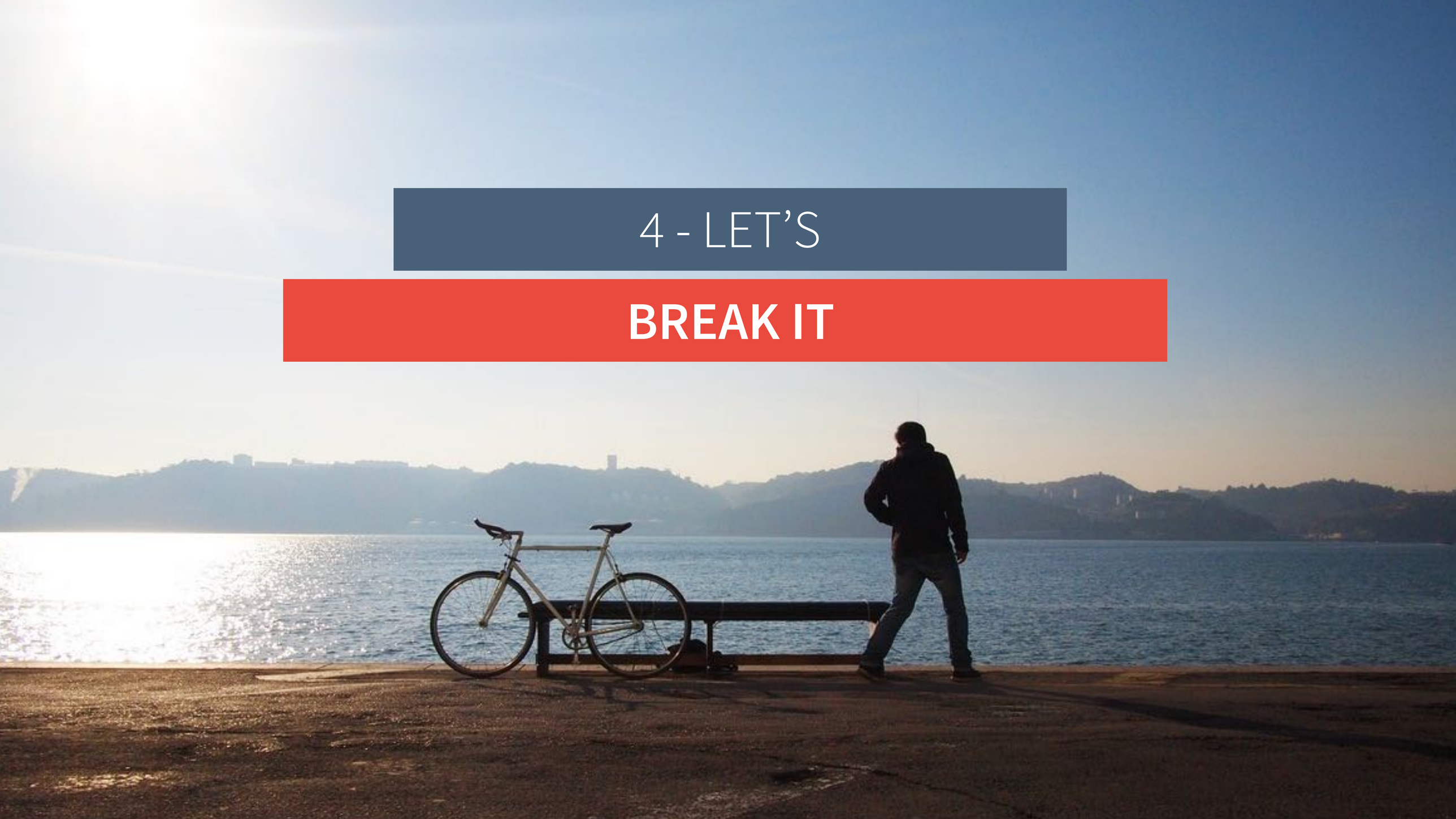
- Obsfucation is only good on critical components. Business does not want a big application.
- Method name are not always obsfucated. Really hard to manage dependencies if the project is large. Secure Container with binary modifications are more efficient.
- When anti-tampering is performed, it's only performed on critical methods not on UI components. As an attacker you can't modify the method but you can get the data.
- Anti-tampering. A method is not called direct but through a secure method. Quite heavy and make the application slow.
- Return to method attack (input/output management).
- Some devices are excluded


- ***Obsfucation is good… and bad. You will find String obsfucation yes but a fully obsfucated binary.. never. Digital Wallet are already big, the business never accept to make it 10 times bigger. So they only obfuscate the "sensitive" features… You can then spot them easily.***
- ***But globally this is where you want to be. It takes time to design and update it properly***

# Security levels **requirements**

## Level 3++: Level 3 + obsfucation, Strong integrity checks, Secure container with anti-tampering

```
Processing com.google.gson.internal.bind.MapTypeAdapterFactory
Processing com.google.gson.internal.bind.ObjectTypeAdapter
Processing com.google.gson.internal.bind.ReflectiveTypeAdapterFactory
Processing com.google.gson.internal.bind.SqlDateTypeAdapter
Processing com.google.gson.internal.bind.TimeTypeAdapter
Processing com.google.gson.internal.bind.TreeTypeAdapter
Processing com.google.gson.internal.bind.TypeAdapterRuntimeTypeWrapper
Processing com.google.gson.internal.bind.TypeAdapters
Processing com.google.gson.internal.bind.util.ISO8601Utils
Processing com.google.gson.stream.JsonScope
Processing com.google.gson.stream.JsonToken
Processing com.google.gson.stream.MalformedJsonException
Processing iiiiii.aaaooa
Processing iiiiii.aooaoa
Processing iiiiii.ffffgf
Processing iiiiii.fffgff
Processing iiiiii.ffgfff
Processing iiiiii.ffggff
Processing iiiiii.fggfff
Processing iiiiii.gggggf
Processing iiiiii.fgggff
Processing iiiiii.fggggf
```



# 4 - LET'S

# BREAK IT

# Techniques to **tools**

Most of the tools you know will not work (Integrity checks, dynamic linker restrictions,…)
Running the application inside an Emulator is not a good idea. NFC, SIM Card communications,.. are not supported properly.

**Two approaches that currently works to bypass most of the protections:**

1) Systemless root for Android:
Since Android 4.3, the “su” daemon the process that handles requests for root access—has to run at startup, and it has to do so with enough permissions to effectively perform the tasks requested of it. This was traditionally accomplished by modifying files found on Android’s /system partition.
In the early days of Lollipop, there was no way to launch the su daemon at boot, so a modified boot image was used (beginning of Systemless root). FYI: Another way has been found later.
- Marshmallow, make it unfeasible to launch the su daemon with the required permissions just by modifying the /system partition. The systemless method was resurrected, and that’s now the default rooting method for phones running Marshmallow. It’s also worth mentioning that this is also true for Android N, as well as Samsung devices running 5.1 (or newer).

# Techniques to **tools**

*2) In memory/runtime modifications to the rescue:*

- *On IOS with Mach Ports and/or with an out-of-process dynamic linker (map your .dylib into sandboxed processes) - Check Frida-gum project (https://github.com/frida/frida-gum/). By experience cyscript is great as a standalone but quickly detected. Things like accessing a bad pointer in Frida gives you a Javascript exception instead of crashing the application which is very useful.*
- *Modify app_process to add additional class path and calls method. Just after the VM has been created, even before the main method of Zygote has been called. Similar to Xposed framework approach.*

*3) Avoid binary modifications as much as you can*

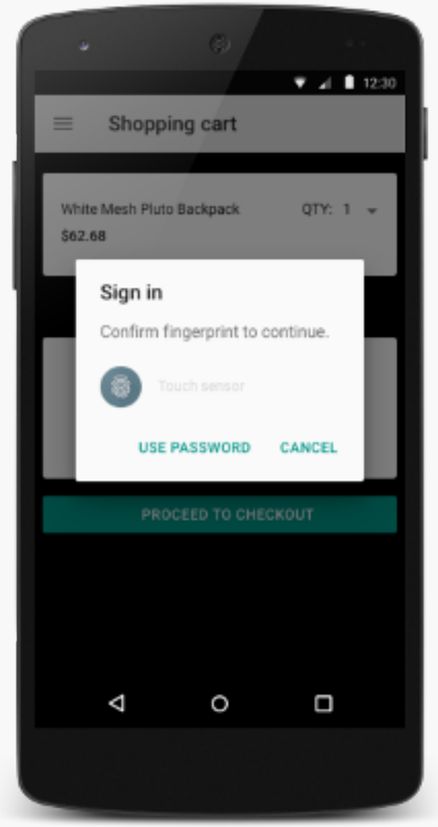
# Useful methods to hooks in **Android**

android.view.Window
android.view.SurfaceView
android.app.ActivityManager
android.os.Process
android.content.ContentResolver
android.accounts.AccountManager
android.location.Location
android.content.ContentResolver
android.media.AudioRecord
android.media.MediaRecorder
android.app.ApplicationPackageManager
android.content.ClipboardManager
android.content.ContextWrapper
java.net.HttpURLConnection
android.content.ContextWrapper
android.app.Activity
android.content.ContextWrapper
android.net.Uri
android.os.Process
android.database.sqlite.SQLiteDatabase
android.content.ContextWrapper
net.sqlcipher.database.SQLiteDatabase
java.io.ObjectInputStream
android.webkit.WebView
libcore.io.IoBridge
android.hardware.fingerprint.FingerprintManager
android.hardware.fingerprint.FingerprintManager
android.hardware.fingerprint.FingerprintManager.AuthenticationCallback
android.webkit.WebView

android.telephony.TelephonyManager
android.net.wifi.WifiInfo
android.telephony.TelephonyManager
android.os.Debug
android.app.SharedPreferencesImpl$EditorImpl
android.content.ContentValues
org.apache.http.impl.client.AbstractHttpClient
android.app.ContextImpl
android.app.ActivityThread
android.app.Activity
dalvik.system.BaseDexClassLoader
dalvik.system.DexFile
dalvik.system.DexClassLoader
dalvik.system.BaseDexClassLoader
dalvik.system.DexFile
dalvik.system.PathClassLoader
javax.crypto.spec.SecretKeySpec
javax.crypto.spec.PBEKeySpec
javax.crypto.Cipher
javax.crypto.Mac
android.app.ApplicationPackageManager
android.app.NotificationManager
android.util.Base64
android.telephony.TelephonyManager
android.util.Base64
android.net.ConnectivityManager
android.content.BroadcastReceiver
android.telephony.SmsManager
java.lang.ProcessBuilder

# Android/IOS **in-app fingerprint**

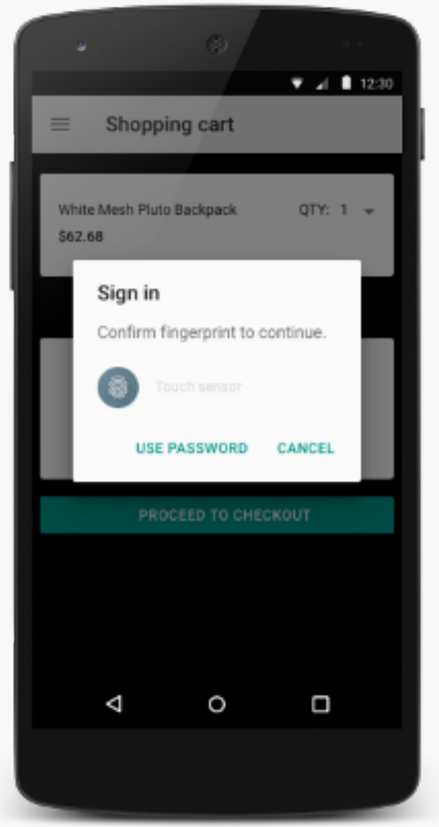
- *Different than the screen unlock*
- *The fingerprint is used to unlock a secret*
- *The secret is stored in the secure enclave, without the good fingerprint you can't unlock the secret*
- *However, by waiting for the user to unlock his app, you can get the secret*



# Samsung **PASS framework**

## Android in-app fingerprint

- *The fingerprint is used to unlock a secret*
- *The secret is stored in the secure enclave, without the good fingerprint you can't unlock the secret*
- *You can have one catalogue managed by the application separated from the system*
- *If the fingerprint authentication is successful, the Android return onAuthenticationSucceeded through an AuthenticationCallback and onAuthenticationFailed otherwise*
- *Event based so if the device is compromised, you can wait for a user to unlock his app, you can get the secret*




## Samsung PASS framework

- *One unique catalog for all fingerprints*
- *The application can detect if a fingerprint is installed*
- *The application can detect if a newFingerprint has been installed since the last time the application was running through in index id table*
- *If the situation did not change, then the authentication process can start.*

Cf: http://developer.samsung.com/onlinedocs/sms/pass/com/samsung/android/sdk/pass/SpassFingerprint.html

# Evil **Partner attack on Samsung S5**

- *One unique catalog for all fingerprints*
- *The application can detect if a fingerprint is installed*
- ***The application can detect if a new Fingerprint has been installed since the last time the application was running through in index id table***
- *If the situation did not change, then the authentication process can start.*


***Let's explore this:***

- ***It can detect some kinds of fingerprint catalog change like adding a new fingerprint***
- ***But… DEVICE_FINGERPRINT_UNIQUE_ID is not supported on Samsung S5.***
- ***Is that meant they only check the size of the index table?***

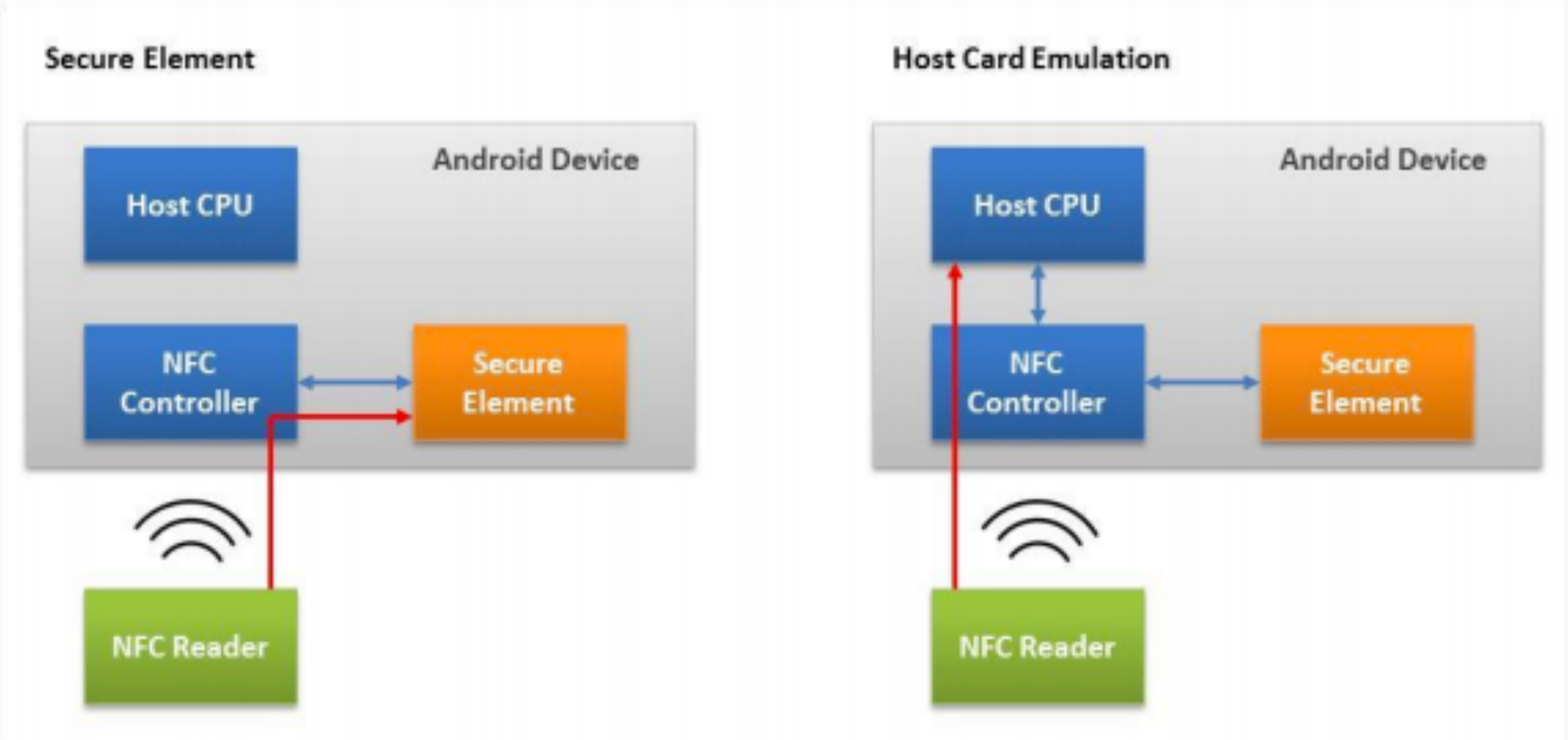
Cf: http://developer.samsung.com/onlinedocs/sms/pass/com/samsung/android/sdk/pass/SpassFingerprint.html

# Evil **Partner attack on Samsung S5**

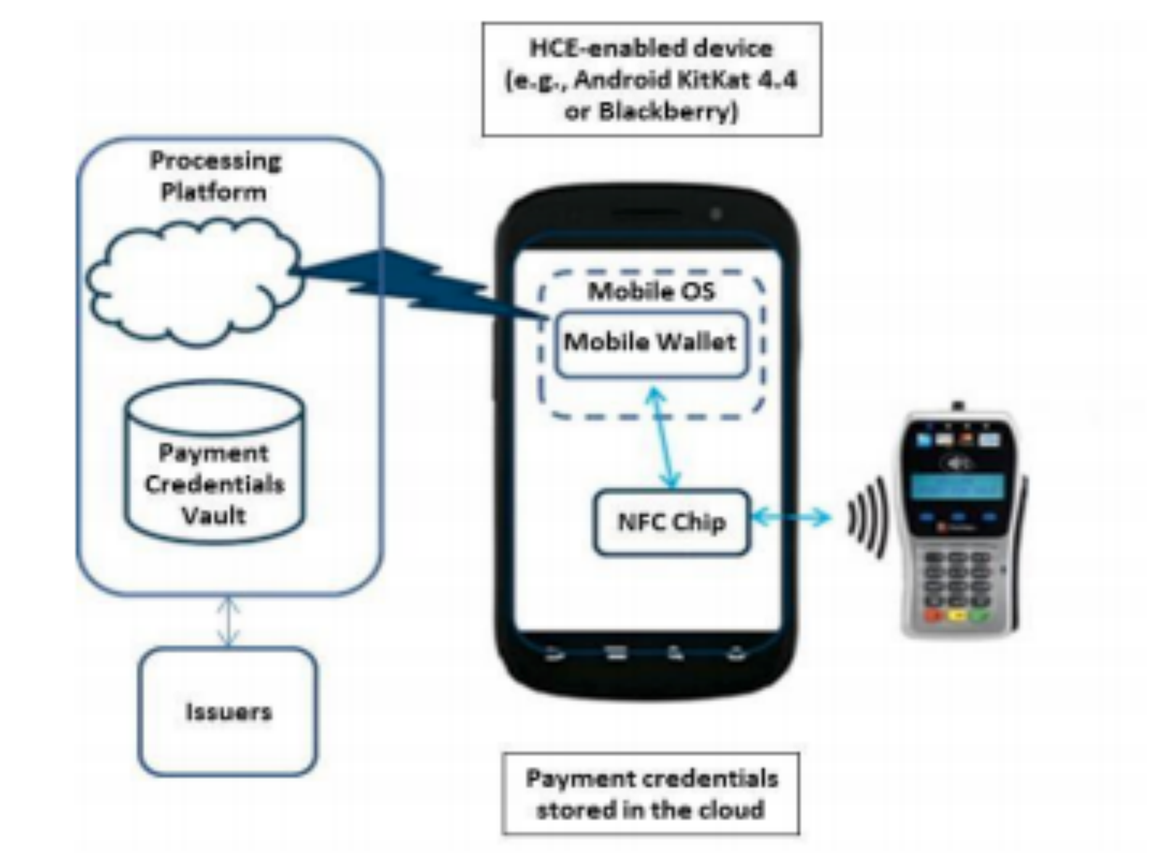
- *So if a partner knows your pattern/pin to unlock your phone he can:*
  - *Unlock your phone*
  - *Go to your fingerprint management menu*
  - *Remove one of the fingerprint*
  - *Add her fingerprint*
  - *Go to the Digital Wallet application*
  - *Put his finger on the Digital Wallet authentication page… he will be authenticated.*


Cf: http://developer.samsung.com/onlinedocs/sms/pass/com/samsung/android/sdk/pass/SpassFingerprint.html

# HCE(Host Based Card Emulation) **attacks**



# HCE(Host Based Card Emulation) **attacks**




1. The customer registers and acquires the card credentials either through a mobile app or using the provider's secure Web-based service (such as a bank's Internet banking site).
2. At time of payment, the customer is authenticated by the cloud (using the credentials entered on the mobile app). The customer then selects a card to use for payment from the mobile wallet app.
3. The payment credentials are sent to the customer's mobile device to initiate the transaction. The device transmits the payment credentials to the merchant using NFC

# HCE(Host Based Card Emulation) **attacks**

## ISO7816

- APDU Command sent by read to the card
- Header, 4 bytes
- Class instruction (CLA)
- Code instruction (INS)
- Parameters: P1 and P2
- Optional body (random size)
- Lc = Length of body in bytes
- Le = Length of response to the command (bytes)
- The data field contains data to be sent to the card, to process instructions specified in header

# HCE(Host Based Card Emulation) **attacks**

## What can we do?

- Some code instructions (INS) can be accepted by the wallet but not referenced
- Some code instructions (INS) can be proprietary. This could lead to some interesting behavior (however, this is usually blocked over the contactless interface). You can brute force force and if it's valid you get a Conditions of use not satisfied (0x6985) as a status response
- Card agents are (sometime) implementing some checks such as validating for transactions that are not exceeding a certain amount threshold or valid only at a given merchant name
- Card agent can crash…
- Card agent can leak information disclosure through debugging feature

# How to **assess a digital wallet**

- *Use physical devices (emulator will not help for SIM Card validation, KYC, NFC, HCE….) a lot of them*
- *Create a set of tools based on in memory modifications.*
- *Log input/output of interesting methods*
- *Do a data lifetime analysis (how long your credit card number stay in memory?)*
- *Use Qark and MobSF to catch low hanging fruits.*
- *Don’t forget that a component can be compromised at any time. Then it’s only a matter of risk management.*

# CONCLUSION

# Conclusion

- **Custom Digital Wallet require strong security requirements.**
- Thinking that any components can be compromised at any time
- Whatever happened, the data stored or transited through the wallet should be safe
- Time of exposure should be evaluated for each sensitive information
- Card agent is still an attacker vector neglected.
- Custom checks slow down the attacker (like hiding part of the UI with Safetynet checks, add business logic into security checks)
- With PSD2 coming in Europe in 2018 and globally Open Banking, we will see more custom Digital Wallet on the Market.
- I will release some of the scripts/tools on https://github.com/Yinkozi

# 6 - THANKS FOR LISTENING!

## QUESTIONS?

# References


- ***Security of Mobile Payments and Digital Wallets - ENISA*** - https://www.enisa.europa.eu/publications/mobile-payments-security/at_download/fullReport
- **Building Secure Containers for Mobile Devices - Ron Gutierrez** - http://2013.appsecusa.org/2013/wp-content/uploads/2013/12/appsec-securecontainers.pptx
- **IOS Security - Apple (Update 2017)** - https://www.apple.com/business/docs/iOS_Security_Guide.pdf
- **TrustZone TEEs: An attacker Perspective - Gal Benjamin -** https://microsoftrnd.co.il/Press%20Kit/BlueHat%20IL%20Decks/GalBeniamini.pdf
- **Mobile Platform Security. Trusted Execution Environments - N. Asokan -** http://asokan.org/asokan/Padova2014/tutorial-mobileplatsec.pdf
- **Hacking Soft Tokens: Advanced Reverse Engineering on Android - Bernhard Mueller -** http://gsec.hitb.org/materials/sg2016/D1%20-%20Bernhard%20Mueller%20-%20Attacking%20Software%20Tokens.pdf

# References


- **IOS 10: Security and Privacy Changes - Alban Diquet -** https://nabla-c0d3.github.io/documents/ios10_security_changes.pdf
- **Pandora's Digital Box: Digital Wallets and the Honor All Wallets Rules - Adam J. Levitin -** https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2819146
- **Android Security Analysis: Final report - MITRE** - https://www.mitre.org/sites/default/files/publications/pr-16-0202-android-security-analysis-final-report.pdf
- **Qark (Quick Android Review Kit) - Linkedin** - https://github.com/linkedin/qark
- **MobSF (Mobile Security Framework) - MobSf** - https://github.com/MobSF/Mobile-Security-Framework-MobSF
- **Low-level code instrumentation library used by Frida-core - Oleavr -** https://github.com/frida/frida-gum