OWASP
Open Web Application Security Project

Let me introduce you the OWASP Mobile App Security Testing

How to test your mobile applications against security vulnerabilities

OWASP Italy Day
Cagliari, 19th October 2018

WHOAMI

Giuseppe Porcu

Software Security Consultant

@MindedSecurity

- Site: https://www.mindedsecurity.com
- Blog: https://blog.mindedsecurity.com
- Email: giuseppe.porcu@mindedsecurity.com
- Telegram: @GTechGuy

AGENDA

- Introduction to OWASP Mobile App Security Testing
- Key Areas of Mobile App Security Testing
  1. Data Storage
  2. Sensitive Data Exposure
  3. Cryptographic Functions
  4. Endpoint Identity Verification
  5. App Permissions
  6. App Signature & Tampering
  7. Anti-Reversing Defense
  8. Anti-Debug Defense
- Conclusions


I SEE CYBER RISKS
EVERYWHERE

FOCUS ON THE PROBLEM

Portable devices

- stolen
- lost

Lot of apps installed on it

- app security is often only presumed

Rooted devices

Testing phase

- often tested against usability and functionality, not security

OWASP MOBILE SECURITY TESTING GUIDE

- Describes processes and techniques for verifying the requirements listed in the Mobile Application Security Verification Standard
- Can be used as a baseline for complete and consistent security tests

- Divided in 3 main sections:
  - General Guide
  - Android Guide
  - iOS Guide

KEY AREAS OF MOBILE TESTING

Similarities with:

- Web App Testing
- Network Testing

Additionally, there are specific key areas related to the mobile environment

KEY AREAS OF MOBILE TESTING

- Local Data Storage
- Communication with Trusted Endpoints
- Authentication & Authorization
- Interaction with the Mobile Platform
- Code Quality & Exploit Mitigation
- Anti-Tampering & Anti-Reversing

OWASP TOP 10 MOBILE RISKS

M1 - Improper Platform Usage
M2 - Insecure Data Storage
M3 - Insecure Communication
M4 - Insecure Authentication
M5 - Insufficient Cryptography
M6 - Insecure Authorization
M7 - Client Code Quality
M8 - Code Tampering
M9 - Reverse Engineering
M10 - Extraneous Functionality

REFERENCES

OWASP Mobile Security Project

https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

OWASP Mobile Security Testing Guide

https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide

Android Developer Security Tips

https://developer.android.com/training/articles/security-tips

TESTING FOR SECURITY 101

Black-Box vs White-Box vs Gray-Box Testing

Static Analysis vs Dynamic Analysis

False Positive Problem

Penetration Test

Reporting

ANDROID TESTING

ANDROID ATTACK SURFACE

- Insecure/compromised storage
- Unsafe input:
  - by means of IPC communication or URL-schemes
  - by the user to input fields
  - to a Webview by a user or by having insecure code loaded into the webview
- Insecure/compromised responses from a server:
  - MITM attack
- Compromised runtime or repackaged app:
  - method hooking and other attacks

1. DATA STORAGE

Public data should be available to everyone, sensitive/private data must be protected, or kept out of device storage

In real scenarios, some type of user data must be stored:

- authentication tokens
- personally identifiable information

Identify:

- **Information processed** by the app
- **Input** by the user
- **Information** that may be **valuable** to **attackers**

1. DATA STORAGE IN ANDROID

Shared Preferences

- collections key/value
- written to a plain-text XML file

SQLite/Realm Databases

- databases unencrypted
- hard-coded password

Internal/External Storage

- “world-readable” files
- rooted devices

1. DATA STORAGE: TESTING

Check AndroidManifest.xml for read/write storage permission

- uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"

Check source code for keyword and API calls

- MODE_WORLD_WRITEABLE | MODE_WORLD_READABLE
- SharedPreference, FileOutPutStream class
- getReadableDatabase, getWritableDatabase functions
- getExternal* functions

Check source code for:

- Sensitive information encrypted via simple bit operations (XOR, bit flipping)
- Keys used/created without Android onboard feature
- Key hard-coded

1. DATA STORAGE: TESTING

Check for common locations of secrets:

- res/values/strings.xml
- build configs: local.properties, gradle.properties

The good way:

- encrypt sensitive data using keys provided by AndroidKeyStore
- do not use Shared Preferences (insecure/unencrypted by default)
- do not use External Storage for sensitive information (data are not removed by default when uninstalling the app)

1. DATA STORAGE: EXAMPLE

Shared Preferences example:

```
SharedPreferences sharedPref = getSharedPreferences("key", MODE_WORLD_READABLE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("username", "administrator");
editor.putString("password", "supersecret");
editor.commit();
```

SQLite Database example:

```
SQLiteDatabase db = openOrCreateDatabase("privateNotSoSecure",MODE_PRIVATE,null);
db.execSQL("CREATE TABLE IF NOT EXISTS Accounts(Username VARCHAR, Password
VARCHAR);");
db.execSQL("INSERT INTO Accounts VALUES('admin','AdminPass');");
db.close();
```

1. DATA STORAGE: EXAMPLE

Resources file example:

```
<resources>
   <string name="app_name">SuperApp</string>
   <string name="hello_world">Hello world!</string>
   <string name="action_settings">Settings</string>
   <string name="secret_key">My_Secret_Key</string>
</resources>
```

Build file example:

```
buildTypes {
  debug {
    minifyEnabled true
    buildConfigField "String",  "hiddenPassword", "\"${hiddenPassword}\""
  }
}
```

2. SENSITIVE DATA IN LOGS

Often developers use logs for debug purpose

However, logging sensitive data may expose the data to attackers or malicious apps

Example:

```
Log.e("Private key [byte format]: " + key);
```

2. SENSITIVE DATA IN LOGS: TESTING

Check source code for:

- Log, Logger classes
- Log.d, Log.e, Log.i … functions
- System.out.print, System.err.print functions
- printStackTrace

Tools like ProGuard (included in Android Studio) can be used to delete logging-related code in production release

2. SENSITIVE DATA AND 3RD PARTIES

Sometimes developers use third-party service for various reasons:

- tracker services
- sell banner ads
- improve user experience

Downside: you can't know exactly what the libraries execute!

Usually included as Jars, API calls or full SDKs

2. SENSITIVE DATA AND 3RD PARTIES: TESTING

Check for necessary permissions in AndroidManifest.xml:

- READ_SMS
- READ_CONTACTS
- ACCESS_FINE_LOCATION

Check source code for:

- API calls
- Third-party library functions
- SDKs

Check if third-party libraries are necessary and whether they are out of date or contain known vulnerabilities

2. SENSITIVE DATA AND IPC

As part of the IPC mechanisms, content providers allow app's stored data to be accessed and modified by other apps

They have to be properly configured or they may leak sensitive data

They are defined inside the AndroidManifest.xml file

2. SENSITIVE DATA AND IPC: TESTING

Check AndroidManifest.xml for providers:

- identified by <provider> tag
- **android:exported** should be explicitly set to “false” if the content is meant to be accessible only by the app itself, otherwise define proper read/write permissions
- **android:permission** tags must be used to limit exposure to others
- **android:protectionLevel** should be set to “signature” (content accessible only by apps signed with the same key)

Check source code for:

- android.content.ContentProvider, android.database.Cursor, android.database.sqlite, .query, .update, .delete

2. SENSITIVE DATA AND IPC: EXAMPLE

Example (AndroidManifest.xml):

```
<provider android:authorities="com.mwr.example.sieve.DBContentProvider" android:exported="true"
android:multiprocess="true" android:name=".DBContentProvider">
   <path-permission android:path="/Keys" android:readPermission="com.mwr.example.sieve.READ_KEYS"
android:writePermission="com.mwr.example.sieve.WRITE_KEYS"/>
</provider>
<provider android:authorities="com.mwr.example.sieve.FileBackupProvider" android:exported="true"
android:multiprocess="true" android:name=".FileBackupProvider"/>
```

Some automatic tools can be used to inspect the app and identify content provider URIs, for example Drozer with scanner.provider.finduris module

2. SENSITIVE DATA AND IPC: EXAMPLE

```
dz> run scanner.provider.finduris -a com.mwr.example.sieve
Scanning com.mwr.example.sieve...
Unable to Query content://com.mwr.example.sieve.DBContentProvider/
...
Unable to Query content://com.mwr.example.sieve.DBContentProvider/Keys
Accessible content URIs:
content://com.mwr.example.sieve.DBContentProvider/Keys/
content://com.mwr.example.sieve.DBContentProvider/Passwords
content://com.mwr.example.sieve.DBContentProvider/Passwords/
```

```
dz> run app.provider.query content://com.mwr.example.sieve.DBContentProvider/Passwords/ --vertical
_id: 1
service: Email
username: incognitoguy50
password: PSFjqXIMVa5NJFudgDuuLVgJYFD+8w== (Base64 - encoded)
email: incognitoguy50@gmail.com
```

2. SENSITIVE DATA IN SCREENSHOTS

Android offers a screenshot feature that it's used:

- by user, taking explicitly a screenshot
- by system for the recent apps view

The screenshot feature is useful but it can leak sensitive data, for example in a banking app it can reveal user's account, info etc…

The feature can be explicitly disabled in the Activities that shown sensitive data

2. SENSITIVE DATA IN SCREENSHOTS: TESTING

Identify in the App the Activities that shown sensitive data

Check source code for:

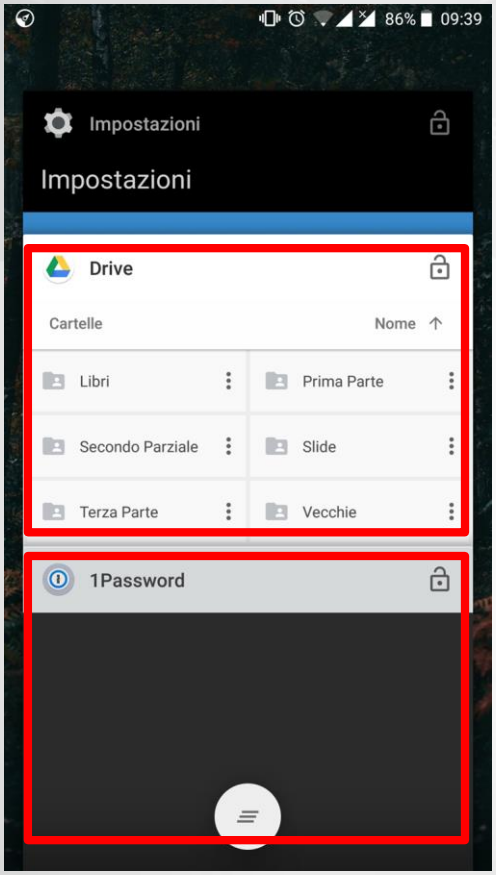
- FLAG_SECURE must be present in the sensitive Activities

Example:

```
getWindow().setFlags(WindowManager.LayoutParams.FLAG_SECURE,
        WindowManager.LayoutParams.FLAG_SECURE);

setContentView(R.layout.activity_main);
```

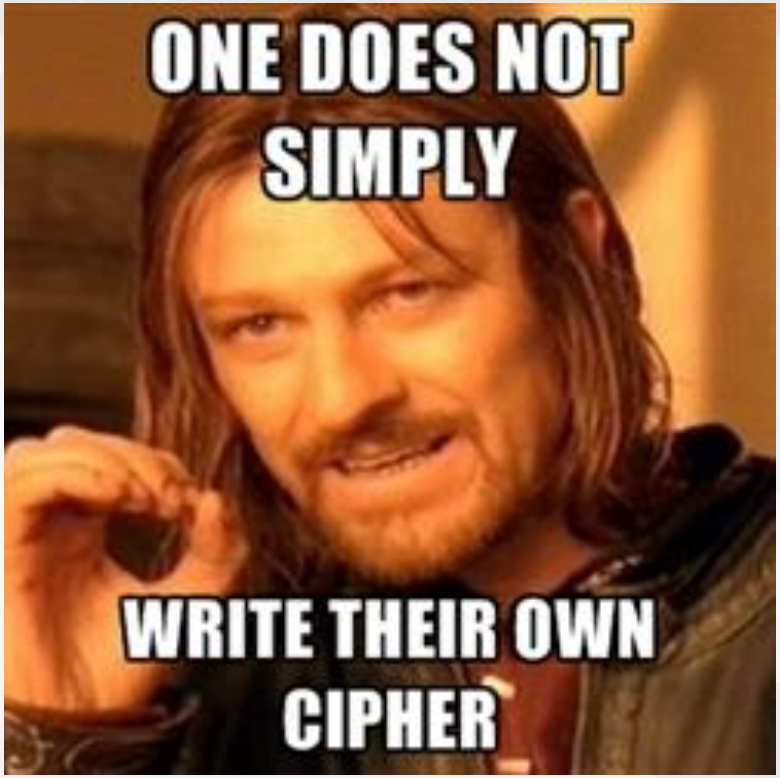
2. SENSITIVE DATA IN SCREENSHOTS: EXAMPLE



Google Drive App not protected against screenshot leakage

1Password App is a Password Manager and it is protected against screenshot leakage

User cannot take screenshot inside the app or see the preview in the recent apps


ONE DOES NOT
SIMPLY
WRITE THEIR OWN
CIPHER

3. CRYPTOGRAPHIC API

Android cryptography APIs are base on JCA (Java Cryptography Architecture)

JCA separates interfaces and implementations, so it's possible to define different security providers

Most of JCA interfaces and classes are defined in:

- java.security.*, javax.crypto.* packages
- android.security.*, android.security.keystore.* packages (Android specific packages)

3. CRYPTOGRAPHIC API: TESTING

The set of existing providers can be listed as follows:

```
StringBuilder builder = new StringBuilder();
for (Provider provider : Security.getProviders()) {
   builder.append("provider: ")
          .append(provider.getName())
          .append(" ")
          .append(provider.getVersion())
          .append("(")
          .append(provider.getInfo())
          .append(")\n");
}
String providers = builder.toString();
//now display the string on the screen or in the logs for debugging.
```

3. CRYPTOGRAPHIC API: TESTING

Check source code for:

- Cipher
- Mac
- MessageDigest
- Signature
- Key, PrivateKey, PublicKey, SecretKey
- java.security.*, javax.crypto.*

Verify that the configuration of cryptographic algorithms used are aligned with best practices from NIST and BSI

3. CRYPTOGRAPHIC API: EXAMPLE

Example of cryptographic functions usage:

```
String keyAlias = "MySecretKey";

KeyGenParameterSpec keyGenParameterSpec = new KeyGenParameterSpec.Builder(keyAlias,
        KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PURPOSE_DECRYPT)
        .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_PKCS7)
        .setRandomizedEncryptionRequired(true)
        .build();

KeyGenerator keyGenerator = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,
        "AndroidKeyStore");
keyGenerator.init(keyGenParameterSpec);

SecretKey secretKey = keyGenerator.generateKey();
```

3. RANDOM NUMBER GENERATION: TESTING

Cryptography requires secure pseudo random number generation (PRNG).

Standard Java classes do not provide sufficient randomness:

- possible for an attacker to guess the next value that will be generated

Identify all the instances of random number generators and look for either custom or known insecure java.util.Random class

Identify all instances of SecureRandom that are not created using the default constructor

3. RANDOM NUMBER GENERATION: EXAMPLE

Example of insecure random generation:

```
import java.util.Random;
// ...
Random number = new Random(123L);
//...
for (int i = 0; i < 20; i++) {
  // Generate another random integer in the range [0, 20]
  int n = number.nextInt(21);
  // ...
}
```

4. ENDPOINT IDENTITY VERIFICATION

Using TLS to transport sensitive data over the network is essential for security

Encrypting communication between app and backend API is not trivial

Developers often decide simpler, less secure solutions (e.g. accept any certificate) to facilitate development process

App is exposed to Man-in-the-middle attacks or MITM

4. ENDPOINT IDENTITY VERIFICATION: TESTING

The App should check:

- certificate signed by a “trusted CA”
- certificate expiration
- self-signed certificate

Note that some frameworks will ignore TLS errors under particular conditions (e.g. Apache Cordova)

Identity Verification could be tested dynamically with a tool like Burp, intercepting the app traffic and changing the option for the certificate

4. CERTIFICATE PINNING: TESTING

Certificate pinning is the process of associating the backend server with a particular X509 certificate or public key

The certificate can be pinned and hardcoded into the app or retrieved the first time the app connects to the host

To customize the network security settings in a safe, declarative configuration file it can be used the **Network Security Configuration** (version > 7.0)

4. CERTIFICATE PINNING: TESTING

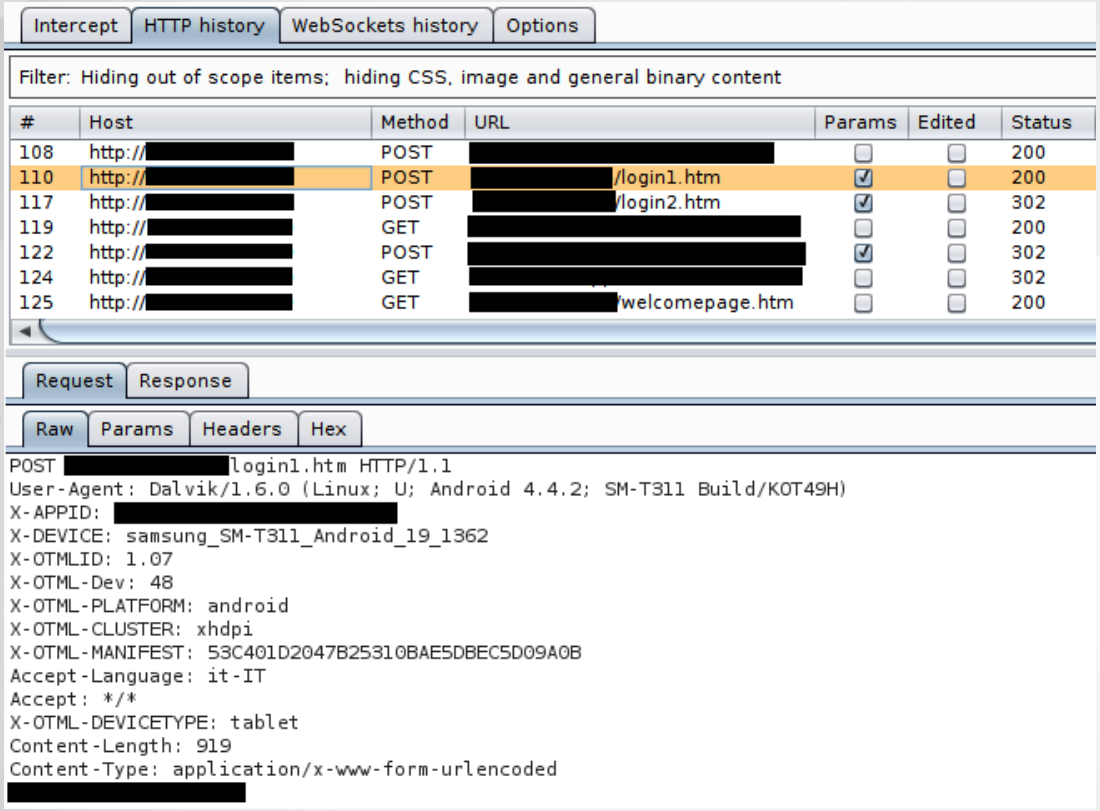
Official Android developer guide describe how to configure correctly the NSC

OWASP Mobile App Security Testing describe how to configure certificate pinning with NSC and others libraries

Reference:

- https://developer.android.com/training/articles/security-config

4. CERTIFICATE PINNING: EXAMPLE



Intercepting traffic of an app without certificate pinning enabled

On rooted device users can install malicious certificate in order to intercept and decrypt the https traffic

5. ANDROID APP PERMISSIONS

Each Android app operates in a process sandbox, so apps must explicitly request access to resources and data outside their sandbox.

Request is made by declaring the permission they need to use system data and feature

Depending on how sensitive or critical the data of the feature is, the Android system will grant automatically or ask the user to approve the request

5. ANDROID APP PERMISSIONS: CATEGORIES

- Normal
  - Access to isolated application-level feature; minimal risk; granted automatically (or approved, depending on SDK level) at install time; android.permission.INTERNET
- Dangerous
  - Control over user data/device in a way that impacts the user; asked to user; android.permission.RECORD_AUDIO
- Signature
  - Granted only if the requested app was signed with the same certificate used to sign the app that declared the permission; granted automatically at install time; android.permission.ACCESS_MOCK_LOCATION

5. ANDROID APP PERMISSIONS: CATEGORIES

- SystemOrSignature
  - Granted only to applications embedded in the system image or signed with the same certificate used to sign the app that declared the permission; android.permission.ACCESS_DOWNLOAD_MANAGER

A list of all permissions is available in the Android developer documentation

Reference:

- https://developer.android.com/guide/topics/permissions/overview

5. ANDROID APP PERMISSIONS: TESTING

Check permissions to make sure that the app really needs them and remove unnecessary permissions

Check the AndroidManifest.xml for permissions

It can also be used the Android Asset Packaging tool

```
$ aapt d permissions com.owasp.mstg.myapp
uses-permission: android.permission.WRITE_CONTACTS
uses-permission: android.permission.CHANGE_CONFIGURATION
uses-permission: android.permission.SYSTEM_ALERT_WINDOW
uses-permission: android.permission.INTERNAL_SYSTEM_WINDOW
```

6. APP SIGNATURE

Android requires all APKs to be digitally signed with a certificate before they are installed or run

This process can prevent an app from being tampered with or modified to include malicious code

Two APK signing schemes are available:

- JAR signing
- APK Signature Scheme v2 (only by Android 7.0 and above)

6. APP SIGNATURE: TESTING

The v2 signature offers improved security and performance

Release builds should always be signed with both schemes

APK signatures can be verified with the apksigner tool (located in [SDK-Path]/build-tools/[version]):

```
$ apksigner verify --verbose Desktop/example.apk
Verifies
Verified using v1 scheme (JAR signing): true
Verified using v2 scheme (APK Signature Scheme v2): true
Number of signers: 1
```

6. DEBUGGABLE PROPERTY: TESTING

The android:debuggable attribute determines whether the app can be debugged or not

With debugger attached apps can leak sensitive informations such as Logs etc..

Verify in AndroidManifest.xml that android:debuggable is set to false for release builds!

```
<application android:allowBackup="true" android:debuggable="true"
android:icon="@drawable/ic_launcher" android:label="@string/app_name"
android:theme="@style/AppTheme">
```

7. ANTI-REVERSING DEFENSE

The goal is to make running the app on rooted devices a bit more difficult

Implementing multiple root checks can improve the effectiveness of the overall anti-tampering scheme

Root detection can be implemented by APIs, libraries or in a programmatic way

7. ANTI-REVERSING DEFENSE: TESTING

SafetyNet

- provides a set of services; creates profiles according to device info
- profile is compared to a list of whitelisted device models
- recommended by Google

Example of attestation:

```
{
  "nonce": "R2Rra24fVm5xa2Mg",
  "timestampMs": 9860437986543,
  "apkPackageName": "com.package.name.of.requesting.app",
  "apkCertificateDigestSha256": ["base64 encoded, SHA-256 hash of the
                    certificate used to sign requesting app"],
  "apkDigestSha256": "base64 encoded, SHA-256 hash of the app's APK",
  "ctsProfileMatch": true,
  "basicIntegrity": true,
}
```

7. ANTI-REVERSING DEFENSE: TESTING

Other ways to check for rooted devices include searching for particular packages, apps and binaries usually associated with rooted devices or searching for ‘su’ in the PATH
Example code:

```
public static boolean checkRoot(){
   for(String pathDir : System.getenv("PATH").split(":")){
      if(new File(pathDir, "su").exists()) {
         return true;
      }
   }
   return false;
}
```

8. ANTI-DEBUG DEFENSE

Debugging is a highly effective way to analyze run-time app behavior.

It allows the reverse engineer to step through the code, stop app execution at arbitrary points, inspect the state of variables, read and modify memory, and a lot more

To check if an app is running inside a debug environment we can check for android:debuggable property or use static functions

8. ANTI-DEBUG DEFENSE: EXAMPLE

Example code:

```
public static boolean isDebuggable(Context context){

   return ((context.getApplicationContext().getApplicationInfo().flags &
ApplicationInfo.FLAG_DEBUGGABLE) != 0);
}

// …

public static boolean detectDebugger() {
   return Debug.isDebuggerConnected();
}

// ...
```

CONCLUSIONS

The OWASP Mobile Security Testing Guide is an open, agile, crowd-sourced effort, made of the contributions of dozens of authors and reviewers from all over the world

It was designed to provide standards for the purpose of verifying the security of mobile applications and develop more secure apps

If you have feedback or suggestions, or want to contribute, you can create an issue on the GitHub or join the Slack channel!

- https://www.github.com/OWASP/owasp-mstg/

CONCLUSIONS

"Don't just follow the OWASP Mobile Security Testing Guide. True excellence at mobile application security requires a deep understanding of mobile operating systems, coding, network security, cryptography, and a whole lot of other things, many of which we can only touch on briefly in this book.

Don't stop at security testing.

Write your own apps, compile your own kernels, dissect mobile malware, learn how things tick.

And as you keep learning new things, consider contributing to the MSTG yourself"

OWASP
Open Web Application Security Project

Thank for your attention!!

Questions??

OWASP Italy Day

Cagliari, 19th October 2018

TIME FOR A CUP OF