BUILDING SECURE MOBILE APPS

(YOU DON’T HAVE TO LEARN IT THE HARD WAY!)

SVEN SCHLEIER AND CARLOS HOGUERA
OWASP STAMMTISCH HAMBURG JAN 2021
Hi everyone, my name is Sven!

- Previous roles: Unix Admin, Penetration Tester, Security Architect for Web and Mobile Apps during SDLC
- Now Security Architect in Singapore
- Project leader together with Carlos Holguera of:
  - OWASP Mobile Security Testing Guide (MSTG) and
  - OWASP Mobile AppSec Verification Standard (MASVS)
- Blogging on [http://bsddaemonorg.wordpress.com/](http://bsddaemonorg.wordpress.com/)
Hola, my name is Carlos!

- Security Engineer & Technical Lead in Berlin:
  - Mobile & Automotive Security Testing
  - Security Testing Automation

- Project leader together with Sven Schleier of:
  - OWASP Mobile Security Testing Guide (MSTG) and
  - OWASP Mobile AppSec Verification Standard (MASVS)
AGENDA

OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

DEMOS
LET ME ASK YOU SOME QUESTIONS FIRST!

Go to www.menti.com and use the code 5192842
SECURE APPS?
FROM THE STANDARD TO THE GUIDE

MOBILE APPSEC VERIFICATION STANDARD (MASVS)

MASVS

A
B
C

https://github.com/OWASP/owasp-masvs/releases

MOBILE SECURITY TESTING GUIDE (MSTG)

MSTG

A

1
2
3
4

B

1
2

C

1
2
3

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

MOBILE APPSEC CHECKLIST

Checklist

✓ A

✓ B

X C

https://github.com/OWASP/owasp-mstg/tree/master/Checklists
OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

HANDS-ON
THE **MASVS** IS A STANDARD THAT DEFINES THE SECURITY REQUIREMENTS APPLICABLE FOR MOBILE APPS AND IS OS AGNOSTIC.

Translations available:

- Chinese (Traditional and Simplified)
- Farsi (Persian)
- French
- German
- Hindi
- Japanese
- Korean
- Portuguese (inca. Brazilian Portuguese)
- Russian
- Spanish

[https://github.com/OWASP/owasp-masvs#getting-the-masvs](https://github.com/OWASP/owasp-masvs#getting-the-masvs)
# V5: NETWORK COMMUNICATION REQUIREMENTS

<table>
<thead>
<tr>
<th>#</th>
<th>MSTG-ID</th>
<th>Description</th>
<th>L1</th>
<th>L2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>MSTG-NETWORK-1</td>
<td>Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5.2</td>
<td>MSTG-NETWORK-2</td>
<td>The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5.3</td>
<td>MSTG-NETWORK-3</td>
<td>The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>5.4</td>
<td>MSTG-NETWORK-4</td>
<td>The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>5.5</td>
<td>MSTG-NETWORK-5</td>
<td>The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>5.6</td>
<td>MSTG-NETWORK-6</td>
<td>The app only depends on up-to-date connectivity and security libraries.</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

[https://github.com/OWASP/owasp-masvs/blob/master/Document/0x10-V5-Network_communication_requirements.md](https://github.com/OWASP/owasp-masvs/blob/master/Document/0x10-V5-Network_communication_requirements.md)
KEEPING THINGS FLEXIBLE: REQUIREMENT “LEVELS”

- **L1** – Standard Security
- **L2** – Defense-in-Depth
- **L2 + R** – e.g. Banking Apps
- **L1 + R** – e.g. Game Apps
- **L2** – e.g. Health Apps
- **L1** – All mobile apps

[Image: https://github.com/OWASP/owasp-masvs/blob/master/Document/0x03-Using_the_MASVS.md#verification-levels-in-detail]
HOW TO USE THE MASVS?

- The levels and its requirements are a baseline that need to be tailored to your needs.

- Don’t blindly follow the requirements!
  - Requirements might be missing (e.g. regulations in your country/industry)
  - Requirements might not be applicable (or you may want to accept the risk)

- Usage ensures consistency of mobile app security when developing / testing an app

- Can be part of your threat model to select the requirements that address your gaps!
WHERE CAN I GET IT?

- Download it
- Read it
- Use it
- Give Feedback and create an issue!
A new release (Version 1.3) is in the making and will be published soon!
OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

HANDS-ON
THE MSTG IS A **COMPREHENSIVE MANUAL** FOR MOBILE APP SECURITY TESTING AND REVERSE ENGINEERING.

IT DESCRIBES **TECHNICAL PROCESSES** FOR VERIFYING THE CONTROLS LISTED IN THE MASVS.

https://github.com/OWASP/owasp-mstg/#reading-the-mobile-security-testing-guide
STRUCTURE OF A TEST CASE IN THE MSTG

Overview

Static Analysis (here you will also find the best practice)

Dynamic Analysis

EXAMPLE: MSTG-PLATFORM-5

Testing iOS WebViews (MSTG-PLATFORM-5)

**Enumerating WebView Instances**

Once you've identified a WebView in the app, you may inspect the heap in order to find instances of one or several of the WebViews that we have seen above.

For example, if you use Frida you can do so by inspecting the heap via `ObjC.choose()`

```javascript
ObjC.choose(Object.prototype, function
  console.log('ObjC.prototype', w);
  console.log('WPreview', w1, w2, w3);
});
```

```javascript
ObjC.choose(Object.prototype, function
  console.log('ObjC.prototype', w);
  console.log('WPreview', w1, w2, w3);
});
```
WHERE CAN I GET IT?


▸ Download it
▸ Read it
▸ Use it
▸ Give Feedback and create an issue!
OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

HANDS-ON
HOW DOES A PENETRATION TESTER EXECUTE A TEST FOR AN IOS APP?

Jailbroken Device
- Cydia App Store
- Full Root Access

Dynamic instrumentation
- Works on (non-)jailbroken devices
- Manipulate runtime behaviour of an app through Frida

See also:
- Frida iOS: https://www.frida.re/docs/ios/
WAYS TO ANALYSE LOCAL STORAGE - OBJECTION

- https://github.com/sensepost/objection

- Python based

- Can be installed and upgraded by using pip3

```
installation

Installation is simply a matter of pip3 install objection. This will give you the objection command. You can update an existing objection installation with pip3 install --upgrade objection.
```

- For more detailed update and installation instructions, please refer to the wiki page:

https://github.com/sensepost/objection/wiki/Installation
How to analyse local storage of an iOS App (Penetration Tester)
WHAT ABOUT IOS DEVS? THEY DON’T USUALLY HAVE A JAILBROKEN PHONE AND FRIDA DOESN’T SEEM TO FIT FOR THEM. THERE SHOULD BE A MORE EASY WAY, RIGHT?

Use the tools you already have: Xcode and iOS Simulator

- You have full access to the file system of the iOS Simulator
How to analyse local storage of an iOS App (Developer Perspective)
WHAT ABOUT IOS DEVS? THEY DON’T USUALLY HAVE A JAILBROKEN PHONE AND FRIDA DOESN’T SEEM TO FIT FOR THEM. THERE SHOULD BE A MORE EASY WAY, RIGHT?

Every app and simulator gets a random 128-bit UUID (Universal Unique Identifier) assigned during installation for its directory names. When using the iOS Simulator the path is:

~/Library/Developer/CoreSimulator/Devices/<Device-UUID>/data/Containers/Data/Application/<App-UUID>

A very handy way to open the data directory of our app running in the current simulator in Finder is the following:

```
$ open `xcrun simctl get_app_container booted info.s7ven.ios.data data` -a Finder
```

The bundle name would need to be explicitly specified, which is `info.s7ven.ios.data` in this case.

See also:
HOW TO DO IT RIGHT?

First reflect: Is it really necessary to store sensitive information on the device? If so, use the following:

Keychain (small bits of data)

- The iOS Keychain can be used to securely store short, sensitive bits of data, such as encryption keys and session tokens. It is implemented as an SQLite database that can be accessed through the Keychain APIs only.

iOS Data Protection APIs

- App developers can leverage the iOS Data Protection APIs to implement fine-grained access control for user data stored on the device.
BIOMETRIC AUTHENTICATION

TOUCH ID / FACE ID

- Fingerprint / facial data is stored in the Secure Enclave which is part of the processor of an iOS device (during calibration).

- The provided data (fingerprint / facial data) is sent to the Secure Enclave and compared with the stored data to authenticate the user.

- An iOS app can confirm via the LocalAuthentication (LAContext) helper class to confirm the devices passphrase, Touch ID or Face ID.
BYPASSING TOUCH ID THE EASY WAY . . .
BIOMETRIC AUTHENTICATION

BYPASSING TOUCH ID
HOW TO DO IT RIGHT?

- 2 different implementations are available:
  - Local Authentication Framework only (LAContext)
  - LAContext together with KeyChain Services

- App stores either a secret authentication token or another piece of secret data identifying the user in the Keychain.

- A valid set of biometrics must be presented before the key is released from the Secure Enclave to decrypt the keychain entry itself.

- This solution cannot be bypassed (even on jailbroken devices), as the verification is done within the Secure Enclave (SE).

See MSTG for sample implementations:

See also:
- [https://apple.co/2KUscTr](https://apple.co/2KUscTr)
HOW CAN WE MAKE SUCH ATTACKS HARDER?

- Jailbreak detection
- Detection of Dynamic Instrumentation (Frida)
- Anti Tampering
- Obfuscation
- ...

Client Side Security Controls are always a cat and mouse game!
JAILBREAK DETECTION

What does Jailbreak Detection mean?

- File-based Checks
- Checking File Permissions
- Checking Protocol Handlers (cydia://)
- Calling System APIs
- ...

See also: http://bit.ly/33oEvgR
DYNAMIC BINARY INSTRUMENTATION DETECTION (FRIDA)

How can Frida be detected?

- Checking the App Signature
- Checking For Open TCP Ports
- Scanning Process Memory
- ...

See also: bit.ly/2MfkXJx

FRIDA IS THERE!
DYNAMIC BINARY INSTRUMENTATION DETECTION (FRIDA)

Where there’s a detection, there is a bypass.


- Bypass: [https://github.com/as0ler/frida-scripts/blob/master/hooks/_jailbreak_detection.disabled](https://github.com/as0ler/frida-scripts/blob/master/hooks/_jailbreak_detection.disabled)

And all in GitHub :)
DETECTION BYPASS THROUGH BINARY PATCHING

- Patch the executable binary file (disassemble or just edit the raw file)

- The bypass might be as easy as making true (0x0) to false (0x1) or replacing some logic with a NOP!

- Repackage and re-run the app
DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

- Inject code to the running app
- Reverse engineering can help finding out which code to inject
- The bypass might be as easy as *forcing a function to return true or false*!

```javascript
setTimeout(function(){
    var MainActivity = Java.use("org.owasp.msst.antifrida.MainActivity")
    MainActivity.checkMemory.overload().implementation = function() {
        console.log("[*] checkMemory function invoked")
        return false
    }
    MainActivity.PortScanFrida.overload('java.lang.String').implementation = function() {
        console.log("[*] PortScanFrida function invoked")
        return false
    }
    MainActivity.getSignature.overload().implementation = function() {
        console.log("[*] getSignature function invoked")
        return "99slZ9rjHWOt7n68qQ3Qwvlyc="
    }
});
```
**FRIDA - MODES OF OPERATION**

**Injected** into a process by running the Frida server on the device

- Working on **only jailbroken** devices
- Frida handles the injection

**Embedded** as shared library (frida-gadget.so) into the mobile app

- Working on **non-jailbroken** devices
- Repackaging and resigning required
FRIDA DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

Inject and Attach
REVERSE ENGINEERING

FRIDA DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

Reverse Engineer
FRIDA DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

Script and Bypass
WHAT ABOUT LAYERING CLIENT SIDE CONTROLS?

- Checking if app was repackaged (Objection/Frida-Gadget)
- Checking if a debugger is used
- Checking if Reverse Engineering tools are used (Frida)
- Checking if device is jailbroken
- Usage of Obfuscation
- etc.

Makes the effort more time consuming and can be used as part of a defence in depth strategy by raising the bar and putting obstacles in the attackers way.

**Remember:** Reverse Engineering is still possible and will always be a cat and mouse game!
**V8: RESILIENCE REQUIREMENTS**

### impede Dynamic Analysis and Tampering

<table>
<thead>
<tr>
<th>#</th>
<th>MSTG-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.1</td>
<td>MSTG-RESILIENCE-1</td>
<td>The app detects, and responds to, the presence of a jailbroken device either by alerting the user or by self-protecting.</td>
</tr>
<tr>
<td>8.2</td>
<td>MSTG-RESILIENCE-2</td>
<td>The app prevents debugging and/or device fingerprinting with the debugger being attached. All available debuggers are covered.</td>
</tr>
<tr>
<td>8.3</td>
<td>MSTG-RESILIENCE-3</td>
<td>The app detects, and responds to, tampering of critical data within its own sandbox.</td>
</tr>
<tr>
<td>8.4</td>
<td>MSTG-RESILIENCE-4</td>
<td>The app detects, and responds to, the presence of reverse engineering tools and frameworks on the device.</td>
</tr>
<tr>
<td>8.5</td>
<td>MSTG-RESILIENCE-5</td>
<td>The app detects, and responds to, being attached to a Honeycomb device.</td>
</tr>
<tr>
<td>8.6</td>
<td>MSTG-RESILIENCE-6</td>
<td>The app detects, and responds to, tampering of critical device state or its own memory space.</td>
</tr>
<tr>
<td>8.7</td>
<td>MSTG-RESILIENCE-7</td>
<td>The app implements multiple mechanisms to protect itself against dynamic analysis (8.1 to 8.6). Note that resiliency scales with the originality of the mechanisms used.</td>
</tr>
<tr>
<td>8.8</td>
<td>MSTG-RESILIENCE-8</td>
<td>The detection mechanisms trigger responses including delayed and stealthy responses, such as device fingerprinting.</td>
</tr>
<tr>
<td>8.9</td>
<td>MSTG-RESILIENCE-9</td>
<td>Obfuscation is applied to programmatic defenses, which in turn impedes de-obfuscation via dynamic analysis.</td>
</tr>
</tbody>
</table>

### Device Binding

<table>
<thead>
<tr>
<th>#</th>
<th>MSTG-ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.13</td>
<td>MSTG-RESILIENCE-13</td>
<td>The app implements a device binding functionality using a device fingerprint derived from multiple properties unique to the device.</td>
</tr>
</tbody>
</table>

---

KEY TAKEAWAYS

- The MASVS defines mobile apps security requirements
- The MSTG outlines those requirements into technical test cases for Android and iOS
- Make a Threat Model of your app
- Get the basics right first (MASVS Level 1)
- Main Security belongs ALWAYS in the server. NEVER rely on client side security controls only.
- Reverse Engineering Controls NEVER go alone, layer them as a defence-in-depth strategy
- The Reverse Engineer will always win!

Thank you!

sven.schleier@owasp.org
@bsd_daemon

Carlos.Holguera@owasp.org
@grepharder