Introduction to OWASP Mobile Application Security Verification Standard (MASVS)

OWASP Geneva

12/12/2016 – Jérémy MATOS

whois securingapps

- Developer background
- Spent last 10 years working between Geneva and Lausanne on security products and solutions
  - Focus on mobile since 2010
- Now software security consultant at my own company

http://www.securingapps.com

- Provide services to build security in software
  - Mobile
  - Web
  - Cloud
  - Internet Of Things
  - Bitcoin/Blockchain

@SecuringApps

Introduction

- Providing mobile apps is required by business
- Native is often the choice
  - Usability
  - Performance
  - Access to sensors
  - Connectivity issues
- A traditional web security assessment only applies to webview integrations
- A mobile application is a fat client and hence has a totally different threat model

Some of the most significant differences

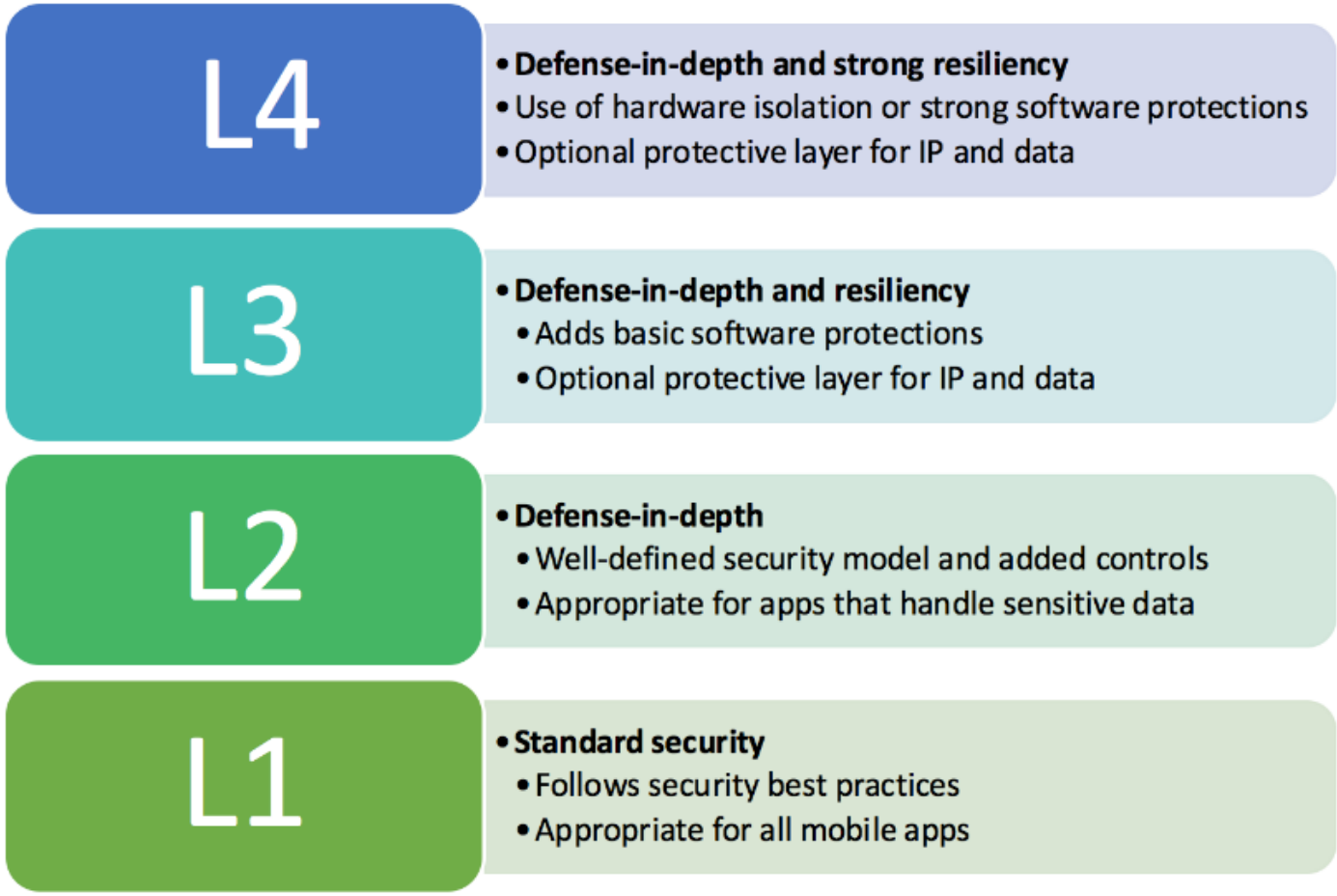
- Code running client side
  - Real local storage
  - Lots of APIs, including for security (e.g encryption)
- Mobile OS are sandboxed
  - Much more clear than Same Origin Policy
- «Trusted» download: applications stores + signature
- Not a HTML hack
  - XSS and CSRF not issues anymore
- But access to many user data

What should we check then ?

- SSL and certificate pinning ?
- Clear text storage in SQLlite database ?
- Obfuscation ?
- Anti-debugging ?
- Encryption in Trusted Excution Environment (TEE) ?

- This is the goal of OWASP Mobile Application Security Verification Standard (MASVS)
  - https://github.com/OWASP/owasp-masvs
  - Project leaders: Bernard Mueller & Sven Schleier
    - http://www.vantagepoint.sg/blog

Security Verification levels 1/3



OWASP MASVS LEVELS

Security Verification levels 2/3

Level 1: Standard Security

An application that achieves MASVS level 1 adheres to mobile application security best practices. It fulfills basic requirements in terms of code quality, handling of sensitive data, and interaction with the mobile environment. A testing process must be in place to verify the security controls. This level is **appropriate for all mobile applications**.

Level 2 : Defense-in-Depth

Level 2 introduces advanced security controls that go beyond the standard requirements. To fulfill L2, **a threat model must exist, and security must be considered during the design phase**. The effectiveness of the controls must be **verified using white-box testing**. This level is **appropriate for applications that handle sensitive data**, such as mobile banking.

Security Verification levels 3/3

Level 3 : Defense-in-Depth and resiliency

Level 3 adds mechanisms that increase the cost of reverse engineering the application. It can be applied to add an additional layer of protection for apps that process sensitive data. Vendors may also opt to implement the L3 requirements as a means of protecting their intellectual property and to prevent tampering with the app.

Level 4 : Defense-in-Depth and strong resiliency

An application that achieves MASVS level 4 has both state-of-the-art security and strong software protections. Such an application **leverages hardware security features** or strong obuscation techniques and is highly resilient against attacks and reverse engineering attempts. L4 is applicable to apps that handle **highly sensitive data**. The L4 controls may also serve as a means of protecting intellectual property or tamper-proofing an app.

Industry specific guidance 1/2

Industry	Threat Profile	L2 Recommendation	L3+ Recommendation
Finance and Insurance	Although this segment will experience attempts from opportunistic attackers, it is often viewed as a high value target by motivated attackers and attacks are often financially motivated. Commonly, attackers are looking for sensitive data or account credentials that can be used to commit fraud or to benefit directly by leveraging money movement functionality built into applications. Techniques often include stolen credentials, application-level attacks, and social engineering. Some major compliance considerations include Payment Card Industry Data Security Standard (PCI DSS), Gramm Leech Bliley Act and Sarbanes-Oxley Act (SOX).	Apps that enable access to highly sensitive information like credit card numbers, personal information, or that can move limited amounts of money in limited ways. Examples include: (i) transfer money between accounts at the same institution or(ii) a slower form of money movement (e.g. ACH) with transaction limits or(iii) wire transfers with hard transfer limits within a period of time.	Apps that enable access to large amounts of sensitive information or that allow either rapid transfer of large sums of money (e.g. wire transfers) and/or transfer of large sums of money in the form of individual transactions or as a batch of smaller transfers.
Healthcare	Most attackers are looking for sensitive data like personally identifiable information (PII) and payment data that can be used for financial gain. Often the data can be used for identity theft, fraudulent payments, or a variety of fraud schemes. For the US healthcare sector, the Health Insurance Portability and Accountability Act (HIPAA) Privacy, Security, Breach Notification Rules and Patient Safety Rule.	Apps that enable access to sensitive medical information (Protected Health Information), Personally Identifiable Information, or payment data.	Apps used to control medical equipment, devices, or records that may endanger human life. Payment and Point of Sale systems (POS) that contain large amounts of transaction data that could be used to commit fraud.

Industry specific guidance 2/2

Gaming	Cheating is an important issue in online games, as a large amount of cheaters leads to a disgruntled the player base and can ultimately cause a game to fail. Popular games usually result in a modding scene that releases patched versions of games that enable features such as infinite in-game currency, threatening the business model on which the game is built.	Online games that use a client-server infrastructure where cheating is a potential issue. Games that process personally identifiable information or sensitive data, such as credit card information.	Games with an essential need to prevent modding and cheating, such as competitive online games.
Manufacturing, professional, transportation, technology, utilities, infrastructure, and defense	BYOD exposes sensitive corporate data in ways that are difficult to control. Lost or compromised mobile devices may grant adversaries access to intellectual property or application functionality that could influence or disrupt sensitive systems. Data obtained from devices may be used for identity theft, fraudulent payments, or a variety of fraud schemes. To mitigate this threat, it is often desirable to employ containerization solutions that improve in an untrusted environment.	Apps that enable access to internal information or information about employees that may be leveraged in social engineering. Apps that enable access to intellectual property or trade secrets.	Apps that enable access to highly sensitive intellectual property, trade secrets, or government secrets (e.g. in the United States this may be anything classified at Secret or above) that is critical to the survival or success of the organization. Applications controlling sensitive functionality (e.g. transit, manufacturing equipment, control systems) where tampering could threaten safety of life.

Detailed verification requirements

- V1 Architecture, design and threat modelling
- V2 Data storage and privacy
- V3 Cryptography verification
- V4 Authentication and session management
- V5 Network communication
- V6 Interaction with the environment
- V7 Code quality and build setting
- V8 Resiliency against reverse engineering

V1 Architecture,design & threat modelling

- At level 1, components of the application are identified and have a reason for being in the app
- At level 2 and higher, the architecture has been defined and the code adheres to the architecture. Additionally, a threat model exists that identifies potential

#	Description	1	2	3	4
1.1	Verify that all app components are identified and are known to be needed.	✓	✓	✓	✓
1.2	Verify that all third party components used by the application, such as libraries and frameworks, are identified, and tested for known vulnerabilities.	✓	✓	✓	✓
1.3	Verify that a high-level architecture for the mobile application and any connected remote services has been defined and that security has been addressed in that architecture.		✓	✓	✓
1.4	Verify that a threat model for the mobile app and any connected remote services has been produced to identify potential threats and countermeasures to work against them.		✓	✓	✓
1.5	Verify that all third party components are assessed and evaluated (associated risks) before being used or implemented. Whenever a security update is published of an implemented third party component, the change has to be inspected and the risk has to be evaluated.		✓	✓	✓
1.6	Verify that all security controls (including libraries that call external security services) have a centralized implementation.		✓	✓	✓

V2 Data storage and privacy

#	Description	1	2	3	4
2.1	Verify that system credential storage facilities are used appropriately to store sensitive data, such as user credentials or cryptographic keys.	✓	✓	✓	✓
2.2	Verify that no sensitive data is written to application logs.	✓	✓	✓	✓
2.3	Verify that no sensitive data leaks to cloud storage.	✓	✓	✓	✓
2.4	Verify that no sensitive data is sent to third parties.	✓	✓	✓	✓
2.5	Verify that the keyboard cache is disabled on text inputs that process sensitive data.	✓	✓	✓	✓
2.6	Verify that the clipboard is deactivated on text fields that may contain sensitive data.	✓	✓	✓	✓
2.7	Verify that no sensitive data is exposed via IPC mechanisms.	✓	✓	✓	✓
2.8	Verify that sensitive data, such as passwords and credit card numbers, is not exposed through the user interface, and does not leak to screenshots.	✓	✓	✓	✓
2.9	Verify that sensitive data does not leak to backups.		✓	✓	✓
2.10	Verify that the app removes sensitive data from views when backgrounded.		✓	✓	✓
2.11	Verify that the app does not hold sensitive data in memory longer than necessary, and that the memory is cleared explicitly after use.		✓	✓	✓
2.12	Verify that the app only runs on operating system versions that offer a hardware-backed keystore, and that the device supports the hardware-backed keystore. Alternatively, verify that encryption has been implemented according to the controls in MASVS V3.			✓	✓

V3 Cryptography verification

#	Description	1	2	3	4
3.1	Verify that that the application does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption.	✓	✓	✓	✓
3.2	Verify that the app uses proven implementations of cryptographic functions.	✓	✓	✓	✓
3.3	Verify that the app does not use cryptograpic protocols or algorithms are used that are widely considered depreciated.	✓	✓	✓	✓
3.3	Verify that cryptographic modules use parameters that adhere to current industry best practices. This includes key length and modes of operation.	✓	✓	✓	✓
3.4	Verify that the application doesn't re-use the same cryptographic key for multiple purposes.	✓	✓	✓	✓
3.5	Verify that all random numbers, random file names, random GUIDs, and random strings are generated using a secure random number generator.	✓	✓	✓	✓
3.6	Verify that all keys and passwords are changeable, and are generated or replaced at installation time.		✓	✓	✓
3.7	Verify that random numbers are created with proper entropy during the application lifecycle.			✓	✓
3.8	Verify that cryptographic controls explicitly clear memory containing (working-)keys after they have been used.			✓	✓
3.9	Verify that consumers of cryptographic services do not have direct access to key material if the keying material is locally generated. Isolate cryptographic processes, including master secrets through the use of strong software protections (MASVS L4) or a hardware key vault (HSM).				✓

V4 Authentication and session mgmt

#	Description	1	2	3	4
4.1	If the app requires access to a remote service, verify that an acceptable form of authentication such as username/password authentication is performed at service endpoints.	✓	✓	✓	✓
4.2	Verify that a password policy exists and is enforced at the remote endpoint.	✓	✓	✓	✓
4.3	Verify that the remote service terminates an existing session when the user logs out.	✓	✓	✓	✓
4.4	Verify that sessions are terminated at the remote end after a predefined period of inactivity.	✓	✓	✓	✓
4.5	Verify that the remote service implements an exponential back-off, or temporarily locks the user account, when incorrect authentication credentials are submitted an excessive number of times .	✓	✓	✓	✓
4.6	If biometric authentication is used, verify that it is not event-bound (e.g.: "fingerprint verification passed"), but that the actual authentication happens based on keying material which can only be unlocked by that specific biometric authentication method.	✓	✓	✓	✓
4.7	Verify that a second factor of authentication exists at the remote endpoint, and that the 2FA requirement is enforced at the remote end.		✓	✓	✓
4.8	Verify that the remote service generate short-lived access token to authenticate client requests without sending the user's credentials.		✓	✓	✓
4.9	Verify that step-up authentication is required to enable actions that deal with sensitive data or transactions.			✓	✓

V5 Network communication

#	Description	1	2	3	4
5.1	Verify that sensitive data is encrypted on the network using secure communication protocols. Verify that the secure channel is used consistently throughout the app.	✓	✓	✓	✓
5.2	Confirm that the app verifies the identity of the remote endpoint when the secure channel is established. Verify that this is done using X.509 certificates, and that only certificates signed by a valid CA are accepted.	✓	✓	✓	✓
5.3	Confirm that communication of 3rd party libraries that send data to their own hosts (e.g. tracking libraries, crash reporter) uses secure communication protocols.	✓	✓	✓	✓
5.4	Verify that the app either uses its own certificate store, or pins the endpoint certificate or the public key.		✓	✓	✓
5.5	Verify that the secure communcation protocol used enables perfect forward secrecy.		✓	✓	✓
5.6	Verify that the app pins the endpoint certificate or the public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA.			✓	✓
5.7	Confirm that the remote endpoint verifies the identity of the app when secure channel is established (PKI mutual authentication).			✓	✓
5.8	Verify that the app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and step-up authentication.			✓	✓
5.9	Verify that the app uses additional payload encryption.				✓

V6 Interaction with the environment

#	Description	1	2	3	4
6.1	Verify that that the app does not request any unnecessary permissions.	✓	✓	✓	✓
6.2	Verify that all inputs from external sources is validated. This includes data received via the GUI, IPC mechanisms such as intents, custom URLs, and network communication.	✓	✓	✓	✓
6.3	Verify that all user input is sanitized and verified (both when using input fields, as well as barcode/QR code data, NFC-related data, etc.).	✓	✓	✓	✓
6.4	Verify that the app does not export sensitive functionality via custom URL schemes.	✓	✓	✓	✓
6.5	Verify that the app does not export sensitive functionality through IPC facilities.	✓	✓	✓	✓
6.6	Verify that Javascript is disabled in all WebViews unless explicitly required.	✓	✓	✓	✓
6.7	Verify that file access is disabled in all WebViews unless explicitly required.	✓	✓	✓	✓
6.8	If Javascript is required in a WebView, ensure that the WebView is restricted to a specific URL, and that no unfiltered user input is rendered in the WebView.	✓	✓	✓	✓
6.9	Verify that the app does not load user-supplied local resources into WebViews.	✓	✓	✓	✓
6.10	If Java objects are exposed in a WebView, verify that the WebView only renders JavaScript contained within the APK (Android).	✓	✓	✓	✓
6.11	Verify that the app leverages operating system features that allow updating of outdated system components.		✓	✓	✓
6.12	Verify that the app provides a custom keyboard whenever sensitive data is entered.			✓	✓
6.13	Verify that custom ui-components are used when sensitive data is displayed. The UI-component should not rely on immutable data structures.			✓	✓

V7 Code quality and build setting

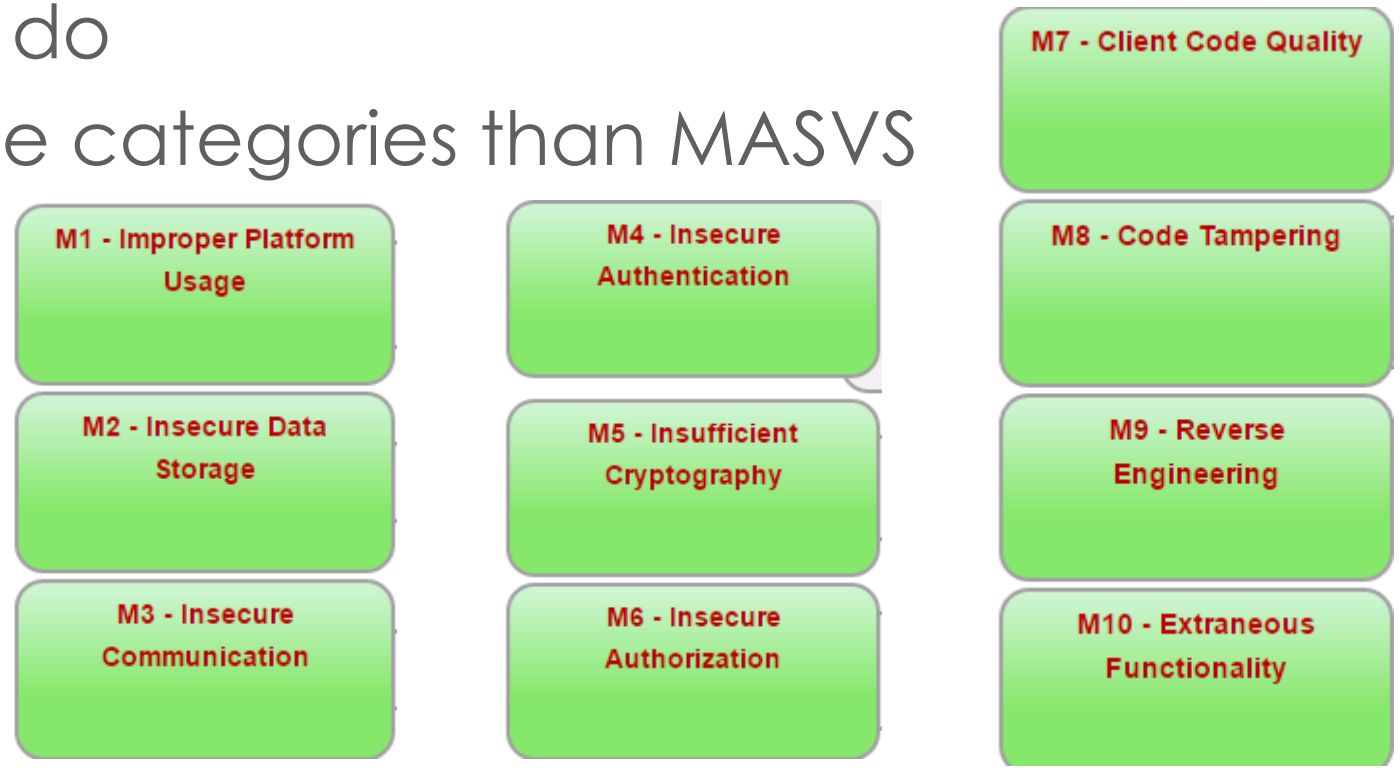
#	Description	1	2	3	4
7.1	Verify that the application catches and handles possible exceptions.	✓	✓	✓	✓
7.2	Verify that all debugging code is removed from the release build, and that the app does log detailed error messages.	✓	✓	✓	✓
7.3	Verify that error handling logic in security controls denies access by default.	✓	✓	✓	✓
7.4	Do not concatenate untrusted external input into database queries or dynamically executed code.	✓	✓	✓	✓
7.5	If the app contains unmanaged code, verify that memory is allocated, freed and used securely.	✓	✓	✓	✓
7.6	Verify that the app is marked as a release build.	✓	✓	✓	✓
7.7	Verify that security features offered by the compiler, such as stack protection, PIE support and automatic reference counting, are activated.	✓	✓	✓	✓
7.8	Verify that static and dynamic application security testing are performed as part of the development lifecycle, and that the configuration of the SAST and DAST tools is tailored to the app.		✓	✓	✓

V8 Reverse engineering resiliency

#	Verified	1	2	3	4
8.1	Verify that debugging symbols have been removed from native binaries.		✓	✓	✓
8.2	Verify that Java bytecode has been obscured through identifier renaming.		✓	✓	✓
8.3	Verify that the application detects whether it is being executed on a rooted or jailbroken device. Depending on the business requirement, users should be warned, or the app should terminate if the device is rooted or jailbroken.		✓	✓	✓
8.4	Verify that the app checks its installation source, and only runs if installed from a trusted source (e.g. Google Play Store / Apple App Store).		✓	✓	✓
8.5	Verify that the app has some form of debugger detection and terminates when a debugger is detected, or prevents attaching a debugger using any method. All available means of debugging must be covered (e.g. JDWP and native).		✓	✓	✓
8.6	Verify that the app implements two or more functionally independent methods of root detection and responds to the presence of a rooted device either by alerting the user or terminating the app.			✓	✓
8.7	Verify that the app implements multiple defenses that result in *strong* resiliency against debugging. All available means of debugging must be covered (e.g. JDWP and native).*			✓	✓

OWASP Mobile Top 10 2016

- https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10
  - Still release candidate. Really alive ?
- More a classification of issues
- Provides high level info on what not to do, rather than detailed info of what to do
- Somehow same categories than MASVS



Conclusion

- MASVS provides clear guidance of what to check in a mobile application
- Really interesting definition of security levels
  - And industry specific advice
- Actionnable
- Reasonable number of controls
- Strong security requirements in general

- Do not hesitate to provide feedback to the project leaders : https://github.com/OWASP/owasp-masvs

Thank you !

Any question 

contact@securingapps.com