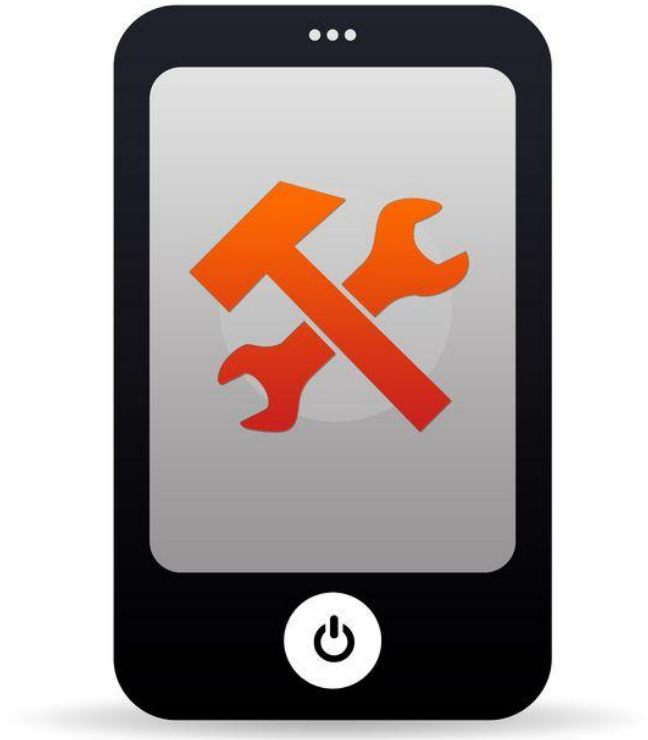

# *Smartphone Secure Development Guidelines for App Developers*

*ENISA Deliverable November 25th 2011*

## Contributors to this report

This document has been produced jointly with OWASP (Open Web Application Security Project) by the following.

- Vinay Bansal, Cisco Systems
- Nader Henein, Research in Motion
- Giles Hogben, ENISA
- Karsten Nohl, Srlabs (on SD/SIM security)
- Jack Mannino, nVisium Security Inc
- Christian Papathanasiou, Royal Bank of Scotland
- Stefan Rueping, Infineon (on SD/SIM security)
- Beau Woods, Dell Secureworks

## Acknowledgements

We would also like to thank Nick Kralevich, Google and members of the OWASP mobile security mailing list for comments and input

## About ENISA

The European Network and Information Security Agency (ENISA) is a centre of network and information security expertise for the EU, its member states, the private sector and Europe's citizens. ENISA works with these groups to develop advice and recommendations on good practice in information security. It assists EU member states in implementing relevant EU legislation and works to improve the resilience of Europe's critical information infrastructure and networks. ENISA seeks to enhance existing expertise in EU member states by supporting the development of cross-border communities committed to improving network and information security throughout the EU. More information about ENISA and its work can be found at www.enisa.europa.eu.

## Contact details

For contacting ENISA or for general enquiries on secure smartphone development, please use the following details:


- E-mail: giles.hogben [at] enisa.europa.eu
- Internet: http://www.enisa.europa.eu

## Contents



## Introduction/Background to this document

This document was produced jointly with the OWASP mobile security project. It is also published as an ENISA deliverable in accordance with our work programme 2011. It is written for developers of smartphone apps as a guide to developing secure apps. It may however also be of interest to project managers of smartphone development projects.

In writing the top 10 controls, we considered the top 10 most important risks for mobile users as described in (1) and (2). As a follow-up we are working on platform-specific guidance and code samples. We hope that these controls provide some simple rules to eliminate the most common vulnerabilities from your code.

The implementation of these controls and the amount of effort put into securing your code should always be matched to the risks you face. For example, if you are quite certain that your app is not handling any kind of personal or sensitive data (this is not always obvious), you may not need to worry about a maximum retention period for data.

## 1. Identify and protect sensitive data on the mobile device

**Risks:** Unsafe sensitive data storage, attacks on decommissioned phones unintentional disclosure: Mobile devices (being mobile) have a higher risk of loss or theft. Adequate protection should be built in to minimize the loss of sensitive data on the device.

1.1 In the design phase, classify data storage according to sensitivity and apply controls accordingly (e.g. passwords, personal data, location, error logs, etc.). Process, store and use data according to its classification. Validate the security of API calls applied to sensitive data.

1.2 Store sensitive data on the server instead of the client-end device. This is based on the assumption that secure network connectivity is sufficiently available and that protection mechanisms available to server side storage are superior. The relative security of client vs server-side security also needs to be assessed on a case-by-case basis (see ENISA cloud risk assessment (3) or the OWASP Cloud top 10 (4) for decision support).

1.3 When storing data on the device, use a file encryption API provided by the OS or other trusted source. Some platforms provide file encryption APIs which use a secret key protected by the device unlock code and deleteable on remote kill. If this is available, it should be used as it increases the security of the encryption without creating extra burden on the end-user. It also makes stored data safer in the case of loss or theft. However, it should be born in mind that even when protected by the device unlock key, if data is stored on the device, its security is dependent on the security of the device unlock code if remote deletion of the key is for any reason not possible.

1.4 Do not store/cache sensitive data (including keys) unless they are encrypted and if possible stored in a tamper-proof area (see control 2).

1.5 Consider restricting access to sensitive data based on contextual information such as location (e.g. wallet app not usable if GPS data shows phone is outside Europe, car key not usable unless within 100m of car etc...).

1.6 Do not store historical GPS/tracking or other sensitive information on the device beyond the period required by the application (see controls 1.7, 1.8).

1.7 Assume that shared storage is untrusted - information may easily leak in unexpected ways through any shared storage. In particular:

- Be aware of caches and temporary storage as a possible leakage channel, when shared with other apps.
- Be aware of public shared storage such as address book, media gallery and audio files as a possible leakage channel. For example storing images with

location metadata in the media-gallery allows that information to be shared in unintended ways.

- Do not store temp/cached data in a world readable directory.

1.8 For sensitive personal data, deletion should be scheduled according to a maximum retention period, (to prevent e.g. data remaining in caches indefinitely).

1.9 There is currently no standard secure deletion procedure for flash memory (unless wiping the entire medium/card). Therefore data encryption and secure key management are especially important.

1.10 Consider the security of the whole data lifecycle in writing your application (collection over the wire, temporary storage, caching, backup, deletion etc)

1.11 Apply the principle of minimal disclosure - only collect and disclose data which is required for business use of the application. Identify in the design phase what data is needed, its sensitivity and whether it is appropriate to collect, store and use each data type.

1.12 Use non-persistent identifiers which are not shared with other apps wherever possible - e.g. do not use the device ID number as an identifier unless there is a good reason to do so (use a randomly generated number – see 4.3). Apply the same data minimization principles to app sessions as to http sessions/cookies etc.

1.13 Applications on managed devices should make use of remote wipe and kill switch APIs to remove sensitive information from the device in the event of theft or loss. (A kill-switch is the term used for an OS-level or purpose-built means of remotely removing applications and/or data).

1.14 Application developers may want to incorporate an application-specific "data kill switch" into their products, to allow the per-app deletion of their application's sensitive data when needed (strong authentication is required to protect misuse of such a feature).

## 2. Handle password credentials securely on the device

**Risks:** Spyware, surveillance, financial malware. A user's credentials, if stolen, not only provide unauthorized access to the mobile backend service, they also potentially compromise many other services and accounts used by the user. The risk is increased by the widespread of reuse of passwords across different services.

2.1 Instead of passwords consider using longer term authorization tokens that can be securely stored on the device (as per the OAuth model). Encrypt the tokens in transit (using SSL/TLS). Tokens can be issued by the backend service after verifying

the user credentials initially. The tokens should be time bounded to the specific service as well as revocable (if possible server side), thereby minimizing the damage in loss scenarios. Use the latest versions of the authorization standards (such as OAuth 2.0). Make sure that these tokens expire as frequently as practicable.

2.2 In case passwords need to be stored on the device, leverage the encryption and key-store mechanisms provided by the mobile OS to securely store passwords, password equivalents and authorization tokens. Never store passwords in clear text. Do not store passwords or long term session IDs without appropriate hashing or encryption.

2.3 Some devices and add-ons allow developers to use a Secure Element e.g. (5) (6) – sometimes via an SD card module - the number of devices offering this functionality is likely to increase. Developers should make use of such capabilities to store keys, credentials and other sensitive data. The use of such secure elements gives a higher level of assurance with the standard encrypted SD card certified at FIPS 140-2 Level 3. Using the SD cards as a second factor of authentication though possible, isn't recommended, however, as it becomes a pseudo-inseparable part of the device once inserted and secured.

2.4 Provide the ability for the mobile user to change passwords on the device.

2.5 Passwords and credentials should only be included as part of regular backups in encrypted or hashed form.

2.6 Smartphones offer the possibility of using visual passwords which allow users to memorise passwords with higher entropy. These should only be used however, if sufficient entropy can be ensured. (7)

2.7 Swipe-based visual passwords are vulnerable to smudge-attacks (using grease deposits on the touch screen to guess the password). Measures such as allowing repeated patterns should be introduced to foil smudge-attacks. (8)

2.8 Check the entropy of all passwords, including visual ones (see 4.1 below).

2.9 Ensure passwords and keys are not visible in cache or logs.

2.10 Do not store any passwords or secrets in the application binary. Do not use a generic shared secret for integration with the backend (like password embedded in code). Mobile application binaries can be easily downloaded and reverse engineered.

## 3. Ensure sensitive data is protected in transit

**Risks:** Network spoofing attacks, surveillance. The majority of smartphones are capable of using multiple network mechanisms including Wi-Fi, provider network (3G, GSM, CDMA and

others), Bluetooth etc. Sensitive data passing through insecure channels could be intercepted. (9) (10)

3.1 Assume that the provider network layer is not secure. Modern network layer attacks can decrypt provider network encryption, and there is no guarantee that the Wi-Fi network will be appropriately encrypted.

3.2 Applications should enforce the use of an end-to-end secure channel (such as SSL/TLS) when sending sensitive information over the wire/air (e.g. using Strict Transport Security - STS (11)).This includes passing user credentials, or other authentication equivalents. This provides confidentiality and integrity protection.

3.3 Use strong and well-known encryption algorithms (e.g. AES) and appropriate key lengths (check current recommendations for the algorithm you use e.g. (12) page 53).

3.4 Use certificates signed by trusted CA providers. Be very cautious in allowing self-signed certificates. Do not disable or ignore SSL chain validation.

3.5 For sensitive data, to reduce the risk of man-in-middle attacks (like SSL proxy, SSL strip), a secure connection should only be established after verifying the identity of the remote end-point (server). This can be achieved by ensuring that SSL is only established with end-points having the trusted certificates in the key chain.

3.6 The user interface should make it as easy as possible for the user to find out if a certificate is valid.

3.7 SMS, MMS or notifications should not be used to send sensitive data to or from mobile end-points.

## 4. Implement user authentication and authorization and session management correctly

**Risks:** Unauthorized individuals may obtain access to sensitive data or systems by circumventing authentication systems (logins) or by reusing valid tokens or cookies. (13)

4.1 Require appropriate strength user authentication to the application. It may be useful to provide feedback on the strength of the password when it is being entered for the first time. The strength of the authentication mechanism used depends on the sensitivity of the data being processed by the application and its access to valuable resources (e.g. costing money).

4.2 It is important to ensure that the session management is handled correctly after the initial authentication, using appropriate secure protocols. For example, require authentication credentials or tokens to be passed with any subsequent request (especially those granting privileged access or modification).

4.3 Use unpredictable session identifiers with high entropy. Note that random number generators generally produce random but predictable output for a given seed (i.e. the same sequence of random numbers is produced for each seed). Therefore it is important to provide an unpredictable seed for the random number generator. The standard method of using the date and time is not secure. It can be improved, for example using a combination of the date and time, the phone temperature sensor and the current x,y and z magnetic fields. In using and combining these values, well-tested algorithms which maximise entropy should be chosen (e.g. repeated application of SHA1 may be used to combine random variables while maintaining maximum entropy – assuming a constant maximum seed length).

4.4 Use context to add security to authentication - e.g. IP location, etc...

4.5 Where possible, consider using additional authentication factors for applications giving access to sensitive data or interfaces where possible - e.g. voice, fingerprint (if available), who-you-know, behavioural etc.

4.6 Use authentication that ties back to the end user identity (rather than the device identity).

## 5. Keep the backend APIs (services) and the platform (server) secure

**Risks:** Attacks on backend systems and loss of data via cloud storage. The majority of mobile applications interact with the backend APIs using REST/Web Services or proprietary protocols. Insecure implementation of backend APIs or services, and not keeping the back-end platform hardened/patched will allow attackers to compromise data on the mobile device when transferred to the backend, or to attack the backend through the mobile application. (14)

5.1 Carry out a specific check of your code for sensitive data unintentionally transferred, any data transferred between the mobile device and web-server back-ends and other external interfaces - (e.g. is location or other information included within file metadata).

5.2 All backend services (Web Services/REST) for mobile apps should be tested for vulnerabilities periodically, e.g. using static code analyser tools and fuzzing tools for testing and finding security flaws.

5.3 Ensure that the backend platform (server) is running with a hardened configuration with the latest security patches applied to the OS, Web Server and other application components.

5.4 Ensure adequate logs are retained on the backend in order to detect and respond to incidents and perform forensics (within the limits of data protection law).

5.5 Employ rate limiting and throttling on a per-user/IP basis (if user identification is available) to reduce the risk from DDoS attack.

5.6 Test for DoS vulnerabilities where the server may become overwhelmed by certain resource intensive application calls.

5.7 Web Services, REST and APIs can have similar vulnerabilities to web applications:

- Perform abuse case testing, in addition to use case testing.
- Perform testing of the backend Web Service, REST or API to determine vulnerabilities.

## 6. Secure data integration with third party services and applications

**Risks:** Data leakage. Users may install applications that may be malicious and can transmit personal data (or other sensitive stored data) for malicious purposes.

6.1 Vet the security/authenticity of any third party code/libraries used in your mobile application (e.g. making sure they come from a reliable source, with maintenance supported, no backend Trojans)

6.2 Track all third party frameworks/APIs used in the mobile application for security patches. A corresponding security update must be done for the mobile applications using these third party APIs/frameworks.

6.3 Pay particular attention to validating all data received from and sent to non-trusted third party apps (e.g. ad network software) before processing within the application.

## 7. Pay specific attention to the collection and storage of consent for the collection and use of user's data

**Risks:** Unintentional disclosure of personal or private information, illegal data processing. In the European Union, it is mandatory to obtain user consent for the collection of personally identifiable information (PII). (15) (16)

7.1 Create a privacy policy covering the usage of personal data and make it available to the user especially when making consent choices.

7.2 Consent may be collected in three main ways:

1. At install time.
2. At run-time when data is sent.
3. Via "opt-out" mechanisms where a default setting is implemented and the user has to turn it off.

7.3 Check whether your application is collecting PII - it may not always be obvious - for example do you use persistent unique identifiers linked to central data stores containing personal information?

7.4 Audit communication mechanisms to check for unintended leaks (e.g. image metadata).

7.5 Keep a record of consent to the transfer of PII. This record should be available to the user (consider also the value of keeping server-side records attached to any user data stored). Such records themselves should minimise the amount of personal data they store (e.g. using hashing).

7.6 Check whether your consent collection mechanism overlaps or conflicts (e.g. in the data handling practices stated) with any other consent collection within the same stack (e.g. APP-native + webkit HTML) and resolve any conflicts.

## 8. Implement controls to prevent unauthorised access to paid-for resources (wallet, SMS, phone calls, etc...)

**Risks:** Smartphone apps give programmatic (automatic) access to premium rate phone calls, SMS, roaming data, NFC payments, etc. Apps with privileged access to such API's should take particular care to prevent abuse, considering the financial impact of vulnerabilities that giveattackers access to the user's financial resources.

8.1 Maintain logs of access to paid-for resources in a non-repudiable format (e.g. a signed receipt sent to a trusted server backend – with user consent) and make them available to the end-user for monitoring. Logs should be protected from unauthorised access.

8.2 Check for anomalous usage patterns in paid-for resource usage and trigger re-authentication. E.g. when significant change in location occurs, user-language changes etc.

8.3 Consider using a white-list model by default for paid-for resource addressing - e.g. address book only unless specifically authorised for phone calls.

8.4 Authenticate all API calls to paid-for resources (e.g. using an app developer certificate).

8.5 Ensure that wallet API callbacks do not pass cleartext account/pricing/ billing/item information.

8.6 Warn user and obtain consent for any cost implications for app behaviour.

8.7 Implement best practices such as fast dormancy (a 3GPP specification), caching, etc… to minimise signalling load on base stations.

## 9. Ensure secure distribution/provisioning of mobile applications

**Risks:** Use of secure distribution practices is important in mitigating all risks described in the ENISA top 10 risks.

9.1 Applications must be designed and provisioned to allow updates for security patches, taking into account the requirements for approval by app-stores and the extra delay this may imply.

9.2 Most app-stores monitor apps for insecure code and are able to remotely remove apps at short notice in case of an incident. Distributing apps through official app-stores therefore provides a safety-net in case of serious vulnerabilities in your app.

9.3 Provide feedback channels for users to report security problems with apps – e.g. a security@ email address.

## 10. Carefully check any runtime interpretation of code for errors

**Risks:** Runtime interpretation of code may give an opportunity for untrusted parties to provide unverified input which is interpreted as code. For example, extra levels in a game, scripts, interpreted SMS headers. This gives an opportunity for malware to circumvent walled garden controls provided by app-stores. It can lead to injection attacks leading to Data leakage, surveillance, spyware, and diallerware.

Note that it is not always obvious that your code contains an interpreter. Look for any capabilities accessible via user-input data and use of third party API's which may interpret user-input - e.g. javascript interpreters.

10.1 Minimise runtime interpretation and capabilities offered to runtime interpreters: run interpreters at minimal privilege levels.

10.2 Define comprehensive escape syntax as appropriate.

10.3 Fuzz test interpreters.

10.4 Sandbox interpreters.

# 1 Annex A – relevant general coding best practices:

Some general coding best practices are particularly relevant to mobile coding. We have listed some of the most important tips here:

- Perform abuse case testing, in addition to use case testing.
- Validate all input.
- Minimise lines and complexity of code. A useful metric is cyclomatic complexity (17).
- Use safe languages (e.g. from buffer-overflow).
- Implement a security report handling point (address) security@example.com
- Use static and binary code analysers and fuzz-testers to find security flaws.
- Use safe string functions, avoid buffer and integer overflow.
- Run apps with the minimum privilege required for the application on the operating system. Be aware of privileges granted by default by APIs and disable them.
- Don't authorize code/app to execute with root/system administrator privilege
- Always perform testing as a standard as well as a privileged user
- Avoid opening application-specific server sockets (listener ports) on the client device. Use the communication mechanisms provided by the OS.

- Remove all test code before releasing the application
- Ensure logging is done appropriately but do not record excessive logs, especially those including sensitive user information.

## 2 Annexe B: Enterprise-specific guidelines

- If a business-sensitive application needs to be provisioned on a device, applications should enforce of a higher security posture on the device (such as PIN, remote management/wipe, app monitoring)
- Device certificates can be used for stronger device authentication.

## 3 References

1. **ENISA.** Top Ten Smartphone Risks . [Online] http://www.enisa.europa.eu/act/application-security/smartphone-security-1/top-ten-risks.

2. **OWASP.** Top 10 mobile risks. [Online] https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_Ten_Mobile_Risks.

3. Cloud Computing: Benefits, Risks and Recommendations for information security. [Online] 2009. http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment.

4. OWASP Cloud Top 10. [Online] https://www.owasp.org/index.php/Category:OWASP_Cloud_%E2%80%90_10_Project.

5. Blackberry developers documents. [Online] http://www.blackberry.com/developers/docs/7.0.0api/net/rim/device/api/io/nfc/se/SecureElement.html,.

6. Google Seek For Android. [Online] http://code.google.com/p/seek-for-android/.

7. Visualizing Keyboard Pattern Passwords. [Online] cs.wheatoncollege.edu/~mgousie/comp401/amos.pdf.

8. *Smudge Attacks on Smartphone Touch Screens.* **Adam J. Aviv, Katherine Gibson, Evan Mossop, Matt Blaze, and Jonathan M. Smith.** s.l. : Department of Computer and Information Science – University of Pennsylvania.

9. Google vulnerability of Client Login account credentials on unprotected . [Online] http://www.uni-ulm.de/in/mi/mitarbeiter/koenings/catching-authtokens.html.

10. SSLSNIFF. [Online] http://blog.thoughtcrime.org/sslsniff-anniversary-edition.

11. [Online] http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-02.

12. NIST Computer Security. [Online] http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_PART3_key-management_Dec2009.pdf.

13. Google's ClientLogin implementation . [Online] http://www.uni-ulm.de/in/mi/mitarbeiter/koenings/catching-authtokens.html.

14. [Online] https://www.owasp.org/index.php/Web_Services.

15. EU Data Protection Directive 95/46/EC. [Online] http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML.

16. [Online] http://democrats.energycommerce.house.gov/sites/default/files/image_uploads/Testimony_05.04.11_Spafford.pdf.

17. [Online] http://www.aivosto.com/project/help/pm-complexity.html.

18. [Online] http://code.google.com/apis/accounts/docs/AuthForInstalledApps.html.

19. Google Wallet Security. [Online] http://www.google.com/wallet/how-it-works-security.htm.


P.O. Box 1309, 71001 Heraklion, Greece
www.enisa.europa.eu