17 August 2015 Study · Mobile Security · 8 min read

Static analysis of selected Android applications

Archived. This content is no longer updated. It remains available for reference.

Source https://technology2.a-sit.at/en/publications/static-analysis-of-selected-android-applications/

We analyzed how a set of selected applications is capable of overcoming real-world threats. Based on current attack vectors, we have derived concrete inspection criteria and applied them on our dataset. As a result, it was feasible to uncover deficiencies in 8 of 10 analyzed applications. The found issues significantly undermine the achievable security level and can lead to the exposure of secrets and the leak of sensitive data to unrelated parties.

Following responsible disclosure principles, a publication of particular deficiencies in the investigated applications is not considered.

1. Introduction

As more and more security-critical functionality is included in Android applications, data integrity has to be ensured by using correct implementations. Since software vendors usually provide only little details on how responsibly sensitive information is protected, an evident need evolves to assess mobile programs on how they overcome real-world threats.

In this project, we aim to inspect selected applications regarding their handling of critical data and we elaborate on possible implications resulting from found misconceptions. By focussing on predefined aspects, the main objective of our analysis consists in uncovering immediate security problems and also to pinpoint their origin.

The remainder of this document is structured as follows. In the next section, the scope of the conducted analysis is denoted. Subsequently, the inspection results are subsumed and typical problems are exemplified. Finally, the last section concludes this document.

2. Analysis Scope

In order to identify problematic code in Android applications, a static analysis procedure is carried out. In contrast to pursuing the inspection dynamically, the application under consideration is not executed on the device. Instead, the presence and constitution of particular patterns is looked up in the disassembly code of

the individual application. Using this approach, we are able to investigate all possible execution flows and pursue an analysis, independent of concrete input data.

2.2. Investigated dataset

For the current analysis scenario, we have empirically selected and downloaded a set of 50 popular Android applications from the official Google PlayStore [1] . Within the available time frame, 10 of these programs have been analysed in detail. Basically, the dataset is composed of applications belonging to different categories and have been chosen for inspection, regardless of whether the appendant description referred to the use of security properties.

2.3. Inspection criteria

The static analysis process involves the identification of predefined schemes in the considered applications. Therefore, we deduce concrete criteria from the listing of OWASP attack vectors, focussing on mobile devices [2] . By maliciously abusing the vulnerabilities, illustrated in Figure 1, sensitive data is prone to be inadvertently exposed to unrelated parties.

In the following, we explain the significance of each attack vector for our analysis:

- **M1: Weak Server-side Controls** The investigation of this attack vector is not subject of our analysis since we are targeting solely the immediate components of the applications, rather than an exposed online service or backend API.
- **M2: Insecure Data Storage** In the context of "Insecure Data Storage", we are verifying whether available security mechanisms are used for the storage of critical data. For example, sensitive information, such as passwords or cryptographic keys shall neither be stored as SharedPreferences, nor within an unprotected file.
- **M3: Insufficient Transport Layer Protection** The use of transport layer protection mechanisms shall ascertain that personal information is not leaked to unrelated parties, probably leading to account theft or fraud. By means of application analysis, we assess prevailing mechanisms which are explicitly included by the

1 https://play.google.com/ 2 https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks vendor to ensure the safety of transmitted data. In particular, we verify whether programs implement TLS certificate validation appropriately and if data transmission is aborted in case a security-related error has occurred. Furthermore, we review whether applications apply "Certificate Pinning" according to principles of this mechanism.

- **M4: Unintended Data Leakage** Exposing sensitive information to insecure locations, such as a developer console (logcat on Android), a globally accessible cache, or a clipboard may result in theft and privacy violations of confidential data. Typically, data leakage is induced unintentionally due by insecure coding practices or vulnerabilities in related components, such as the operating system or a bundled library. Considering our focus on inspecting the application binary, we aim to uncover data flows in applications in which user-entered passwords are written to log files or the file system. Likewise, we are inspecting the processing of critical information in 3 [rd] -party extensions, usually added to Android applications for gathering statistics.

- **M5: Poor Authorization and Authentication** While the complete absence of user authorization may involve the theft of confidential information, our analysis aims to uncover hard-coded usernames and passwords which are employed to authenticate with online services. In particular, we inspect applications regarding the use of login credentials for HTTP Basic authentication. Different login operations, such as parameters sent via GET or POST requests, are not considered since their embodiment might be assembled during runtime and vary depending on the individual execution context.

- **M6: Broken Cryptography** An emphasis of the application analysis has been put on the inspection of cryptographyrelated program sequences. The usage of weak algorithms, improper key management, or flawed security properties can result in severe security breaches, enabling attackers to gain access to critical information. Within the scope of our analysis, we evaluate whether applications make use of algorithms which are knowingly susceptible to attacks and whether cryptographic keys or salt values are hard-coded, or can easily be derived by attackers. Furthermore, we intend to find out whether the values of security properties used for cryptography do not fall below a threshold which impairs the achieved security level. As a result, we aim to highlight problematic cryptographic constructions and outline the resulting security impact.

- **M7: Client-Side Injection** The injection of malevolent code via Android applications can lead to the theft of critical data, such as passwords, cookies, or other personal information. The analysis of applications regarding the possible impact would require a thread model, tailored to the security requirements of the inspected applications. We, therefore, limit the analysis a circumstance which could promote code injection. Precisely, we evaluate whether used WebView components react appropriately and abort the transmission in case of TLS validation errors.

- **M8: Security Decisions via Untrusted Inputs** Typically, Android applications employ the BroadcastReceiver [3] mechanism for interprocess communication. Depending on the configuration of the receiver, an attacker might bypass protection mechanisms of an application by crafting broadcast messages with tampered parameters. As a consequence, we are verifying whether used broadcast receivers are configured to receive messages from external components.

3 http://developer.android.com/reference/android/content/BroadcastReceiver.html

- **M9: Improper Session Handling** This attack vector involves the communication of an application with an online service. As we are conducting a static inspection, the analysis of session negotiation is not feasible within the current scope.

- **M10: Lack of Binary Protections** Android applications are signed by the developer in order to prevent unauthorized modifications. In addition, many applications apply code obfuscation to impede reverse engineering. Within the scope of this analysis, we do not consider the exigence of binary protection.

Figure 1. OWASP Mobile Top 10 Risks

3. Results

The inspection of 10 applications regarding the aforementioned criteria exposed weaknesses in 8 programs. In the following, we summarize the typical misconceptions and also refer to the associated security impact.

6.1. Typical problems

In the following, three issues are explained which were found in more than half of the analysed applications:

- **Usage of a constant password, salt, or encryption key** Keeping passwords and keys secret is an essential requirement in order to impede unrelated parties from accessing private information. Statically defined passwords or keys clearly contradict to this basic rule and render the protection mechanism useless. The purpose of a randomly chosen salt value is to ensure the uniqueness of a derived encryption key and to slow down brute-force or table-based attacks.

Our inspection has revealed that the confidentiality of protected data was drastically impaired if constants were used as secrets. If not constants, it could be observed that passwords, salts or keys tended to be derived by cryptographically weak pseudorandom number generators, such as *java.util.Random* . In that case, a deterministic output was produced, unsuited for security-critical applications.



- **Storage of critical data as SharedPreferences** According to the principle that encryption is employed to protect confidential information, sensitive data shall not be saved as SharedPreferences [4] on the Android file system. As opposed to using a secure element in hardware, critical data is not adequately protected against malevolent attacks and unauthorized access if the device is rooted, stolen, or lost.

- **Flawed certificate validation** Verifying the certificate of a TLS-protected online service (correctly) is a vital requirement which needs to be satisfied in order to establish a private connection to a remote party. It has been observed that some applications either renounce entirely from checking server certificates, are weakly implemented, or silently discard the output of failing validations. As a consequence, invalid / forged server certificates might be accepted by applications. In the worst case scenario, attackers may hijack and record data, transmitted via a TLS-protected connection.

3. Conclusion

Android applications often include security-critical functionality which has to be implemented correctly in order to ensure that no sensitive data can be leaked to unrelated parties.

In this project, we have shown analysed a selection of applications and inspected whether they are able to withstand current attack vectors. The investigation has revealed weaknesses in 8 of 10 applications. The determined problems undermine the achievable security level and can potentially lead to the theft of confidential information.

4 http://developer.android.com/training/basics/data-storage/shared-preferences.html