A Summary of Danger is My Middle Name: Experimenting with SSL Vulnerabilities in Android Apps

1 Abstract

This paper will cover information leaks and SSL vulnerabilities in Android apps, including both static and dynamic analysis on 100 surveyed Android apps. Overall, Android apps are very insecure, with 32/100 apps that accept all certificates and hostnames, 4/100 that transmit sensitive, unencrypted data, and that 91/100 were vulnerable to man in the middle attacks with an installed adversary certificate.

2 Introduction

73% of mobile phone users use their mobile phone to regularly access the Internet, and this is increasing. SSL security for mobile devices is more prone to security vulnerabilities than browsers. The authors of this article used dex2jar to decompile the applications. Three Man in the Middle attacks scenarios were tested on the applications, including an advanced adversary that has actually installed a certificate on a user's phone, an app with an SSL implementation that accepts all certificates, and app that does not correctly implement performing hostname verification. 93 of the 100 apps used SSL and 78 of those used custom SSL code. Using static analysis, the authors found that half of the apps accept all certificates, half of the apps fail hostname verification, 90 of the 100 apps establish http connections under attack of the first MITM Attack scenario and about one quarter of the apps establish http connections under the second and third scenarios. 4 of the 100 apps established login sessions over HTTP.

3 Related Work

See the related work section for further reading on permissions, privilege escalation, detecting malicious apps, information flow tracking, and SSL implementation details. The authors point out that while the related works warn users and developers of security vulnerabilities similar to that of this paper, this paper was published 1 to 2 years later when the security threat has remained virtually the same.

4 Background

SSL Handshake - To begin an SSL session you need to have a handshake between the client and the server. If the handshake is not implemented properly an adversary could impersonate a server. To ensure that the server is verified the client has to check the Chain of Trust for the server's certificate and hostname.

SSL MiTM Attacks - A man in the middle attack is exactly what it sounds like. A man in the middle attack is successful if:

- The client accepts all certificates as the server's
- The client doesn't validate
- The client accepts expired certificates
- An adversary fakes the server's certificate
- The client accepts all self-signed certificates

SSL Implementation in Android - Android apps rely on android.net, android.webkit, java.net, javax.net, java.security, jvax.security.cert, and org.apache.http packages from the Android SDK to supply end to end security. The SSK certificate validation logic uses the TrustManager and HostnameVerifier interfaces provided by these packages. TrustManager manages the CA certificates. HostnameVerifier is used for hostname verification for urls.

SSL Pinning - This is a bit of an extra step that involves coupling the host's trusted credential to the host's identity.

5 Methodology

Experimental testbed - The authers used an LG Nexus 4 smartphone running Android KitKat 4.4.4 with a Lenovo laptop running Windows 8.1 acting as a Wi-Fi Access Point. The authors Wireshark and Fiddler2 to simulate man in th emiddle attacks.

Timeline - The study presented took place in 2014 and 2015.

Analyzed Apps - The apps that the authors chose to survey were those that requested full network permissions and had at least 10 million downloads. The apps chosen roughly represent 10% of all apps with 10 million downloads or more.

Sensitive information - includes "login credentials, device information, location data, chat and email messages, financial information, calendars, contact list, files, and so on." The authors do not consider data transmitted by embedded ad libraries.

Static Analysis - dex2jar and JDGUI were used to decompile the apps. The authors searched for key terms such as, "HttpsURLConnection, HostnameVerifier, and TrustManager" to find the SSL code. The authors then analyzed the TrustManager and HostnameVerifier implementations.

Dynamic Analysis - For dynamic analysis, the authors assumed that the adversary has control over the Wi-Fi access point and simulated the following, verbatim:

- **S1:** The adversary has his certificate installed on the user's device (to simulate this, we install a Fiddler certificate as root CA);
- **S2:** The adversary presents an invalid, self-signed certificate;
- **S3:** The adversary presents a certificate with a wrong Common Name (CN) and/or SubjectAltName, signed by a root CA.

Ethics - The authors did not capture any user's traffic and all tests were observed in a controlled environment.

6 Results

Static Analysis

93/100 apps include SSL code. 78 of these 93 implement their own TrustManager or HostnameVerifier. 48 of these 93 allow HostameVerifier to accept all hostnames. 41 of these 93 define a verifier that always returns true without a check. 46 of these 93 include a TrustManager that accepts all certificates.

Dynamics Analysis

The authors first observe memory leakage and then man in the middle attacks.

Unencrypted traffic - 3 apps sent unencrypted usernames and passwords, 4 gps location, 2 IMEI number, and 1 IMSI number.

MiTM - 91 of the 100 apps give access to sensitive data due to their implementation of the login connection. 9 of the apps do not connect due to SSL pinning. This is for scenario 1. For scenario 2, 23 of the apps complete the connection despite the adversary presenting an invalid certificate. 9 of these 23 leaked sensitive information. For scenario 3, when the adversary uses a certificate with incorrect CN or SubjectAltName, 29 apps established a connection with 11 of these 29 leaking sensitive information. 20 apps were vulnerable in all three scenarios. 9 of these 20 leaked sensitive information. Only 3 apps present an SSL certificate error. Others continuously load, display an incorrect error message, or crash.

7 Analysis

”Secure” Apps

Even when the tested apps were secure, the in app purchase screens were not.

Google Apps

Even the Google Apps that were preloaded on the Android phone were vulnerable to attacks in scenario 1. Even non-Google apps that use Google’s in app purchase API were vulnerable. 40 of the 100 apps tested were vulnerable due to this.

Vulnerable Apps

59 of the apps were vulnerable in scenario 1 but not scenarios 2 and 3. This includes apps like Netflix and Facebook. This concludes that scenario 1 is the hardest attack to avoid. To avoid this attack programmers can use SSL pinning, but this can add extra costs to development. After contacting the developers of some of these applications, the authors revisited the tests and the same apps no longer necessarily fall vulnerable to the same attack scenarios.

Static vs Dynamic Analysis

There was an inconsistency between the author’s static and dynamic analysis.

Possible false negatives in dynamic analysis - Some of the tests in dynamic analysis may have missed the improper SSL implementations that the static analysis was able to pick up on due to actualy viewing the code.

Possible false positives in static analysis - Some of the implementation for SSL that proved vulnerable may have been added for testing purposes and then removed for production.

8 Discussion

Self-signed and testing

The authors recommend that developers stay away from most TrustManagers due to their acceptance of self-signed certificates. It is possible that developers accept self signed certificates for testing purposes during development but forget to switch this capability off for production. They recommend that developers use SSL pinning rather than customizing a TrustManager.

Open Problems

Even after attention to improper SSL implementation through research done 1 to 2 years before the publication of this study, there are still apps that allow for self signed certificates. The authors argue that developers should be presented with better tools for testing apps and their SSL implementation before the app is released for production. They also argue that checking for SSL implementation vulnerabilities and for information leakage should be enforced by the Google Play store. These could even be made public, encouraging safer app development. An interesting point that the authors bring up is properly indicating that a connection is secure, such as the security lock displayed by most browsers. Or at the very least, the error messages should provide a bit more context.