A client-side analysis of TLS usage in mobile apps

Abbas Razaghpanah
Stony Brook University

Narseo Vallina-Rodriguez
ICSI

Phillipa Gill
Stony Brook University

1 Introduction

As mobile applications become more pervasive, they provide us with a variety of online services that range from social networking to banking and credit card management. Since many of these services involve communicating and handling of private user information – and also due to increasing security demands from users – the use of TLS connections has become a necessity for today's mobile applications. However, an improper use of TLS and failure to adhere to TLS security guidelines by app developers, exposes users to agents performing TLS interception thus giving them a false sense of security. Unfortunately, researchers and users alike lack of information and easy-to-deploy mechanisms to analyze how securely mobile apps implement TLS.

Hence, in order to understand and assess the security of mobile app communications, it is crucial to study their use of the TLS protocol. In this poster we present a method to study the use of TLS in mobile apps using the data provided by the ICSI Haystack app [2], a mobile measurement platform that enables on-device analysis of mobile traffic without requiring root access. The unique vantage point provided by the Haystack platform enables a variety of measurements from the edge of the network with real user workload and the added bonus of having contextual information on the device to supplement the data collection.

2 Previous Work

Studies that use instrumented test-beds [1] or use static analysis usually have smaller scale and lack the organic user and network-stimuli required to study apps in real-world situations. Additionally, these approaches do not cover vendor and carrier-specific apps that are not on the Play Store. On the other hand, approaches that analyze ISP traces operate a VPN server outside the device [3] have larger scale, but lack the contextual information that can be obtained on the device (*e.g.,* traffic-to-app mapping) thanks to information provided by the operating system.

Previous work has looked at the problem of TLS communications on mobile devices from different angles and from different vantage points, but there has not been a comprehensive on-device study conducted at large scale.

3 Methodology

We make use of Haystack, a general-purpose on-device mobile measurement platform, to study TLS usage by apps. Haystack doesn't require root access to monitor app traffic and is available to download from the Play Store. Data collected by Haystack is approved by IRB [1] and comes from real users interacting with their apps. Haystack uses the VPN interface to gain access to apps' traffic and allows us to capture and analyze TLS usage by these apps (Figure 1), including pre-installed applications and others not available on popular app-stores. Haystack has been downloaded by more than 500 users and has collected TLS information from around 1,700 apps in the wild. Right now Haystack app focuses on providing transparency to users about what information their mobile apps are exfiltrating about them over the Internet, particularly information leakages to third-party services, tracking activity, and privacy leaks.

3.1 TLS Proxy

Haystack supports an opt-in TLS proxy to study and analyze TLS traffic. With user's approval, it injects a self-signed root certificate on Android's cert store. Haystack parses packets before forwarding them to the outbound Java sockets, packets that look like TLS Client Hello are parsed to extract relevant information (e.g., SNI field), uploaded to the server in raw binary format, and for-

[1]Due to the data anonymization and reduction process taking place on the handset to avoid collecting personal information, UC Berkeley's IRB considered this study as a non-human subject research.

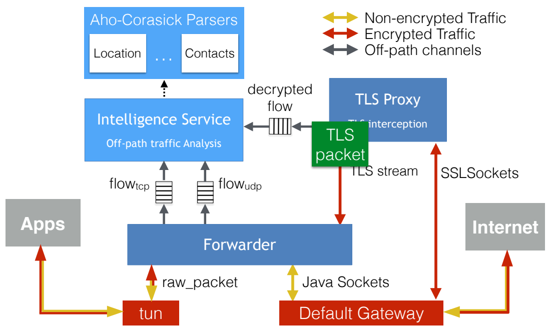

Figure 1: The Haystack architecture, highlighting system components and data forwarding channels.

warded to the TLS proxy. The TLS proxy will try to connect to the remote server on behalf of the app, and upon successful handshake with the server, will finish handshake with the client. If the handshake with the server fails, the exception information is stored and uploaded to our server for further analysis. Server certificate blobs are uploaded to our server in the event of a successful proxy connection.

We use client-side and server-side TLS handshake exceptions to determine which apps bundle CA certificate stores, hard-code certificate signatures, or pin certificates using other methods.

Server-side failures. If a server-side cert can't be verified by the proxy, it can be one of the following: either the certificate is invalid or it is self-signed. When this type of failure happens, the connection is handed over to the app and is not proxied. That way we can follow up on the connection and see if the client accepts the certificate. If the certificate is accepted, it means that either the client doesn't do certificate checking, or that it checks the certificate using hard-coded information such as a custom CA cert store or a certificate fingerprint used to pin the certificate. In case the app doesn't accept the certificate, we conclude that the certificate is invalid.

Client-side failures. If the handshake fails on the client-side of the proxied connection, it could point to certificates being pinned or CA stores being bundled with the app. It could also mean that some of the extensions that were used in the original handshake to the server are not supported by the on-device TLS proxy server.

3.2 Handshake Message Analysis

Since Haystack uploads Client Hello packets from the handshake process, we can analyze and fingerprint different aspects of the apps' TLS usage. Supported cipher suites and extensions present on the Client Hello can be utilized to determine which libraries are used for TLS sockets. Apps that still use older TLS versions, less secure cipher suites, or deprecated parameters can be detected. The certificates that are collected by Haystack are analyzed to extract information such as key type and length, validity period, signature and signature algorithm, and the CA.

The data collected in the wild by Haystack so far (more than 2.6 million TLS flows) shows that while the vast majority of TLS flows (90.26%) use TLSv1.2, there are still some that use TLSv1.0 (8.29%) and TLSv1.1 (0.80%), with a smaller percentage using the vulnerable SSLv3 (0.63%). Moreover, RC4 ciphers that have been shown [4] to be weak, as well as MD5 hashing algorithm that is vulnerable to collision, are supported in 76.11% and 8.08% of the flows studied respectively. This shows that the majority of the apps that use TLS use Android's default library. Therefore, newer versions of Android use newer versions of the protocol and announce support for cipher suites that were not known to be unsecure at the time of OS release; which demonstrates the importance of upgrading the OS version as apps are dependent on OS support for TLS security.

4 Future Work

In addition to collecting Client Hello packets from TLS connections, we plan to extend our data collection process to gather Server Hello messages to extract more information from the server-side of the connection. This will allow us to identify the final cipher suite chosen for the connection and other parameters set by the server which will help us learn more about the server-side of the TLS connection and get a better picture of how securely either end of the connection really is. Additionally, we can use this information to warn the user about vulnerable TLS connections and expired certificates in real-time. As a way to improve the state of TLS usage, we plan to create a census of apps with improper TLS usage to encourage better and more secure TLS implementation by app developers.

References

[1] Fahl, S., Harbach, M., Muders, T., Smith, M., Baumgärtner, L., and Freisleben, B. Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *ACM CCS* (2012).

[2] Razaghpanah, A., Vallina-Rodriguez, N., Sundaresan, S., Kreibich, C., Gill, P., Allman, M., and Paxson, V. Haystack: In situ mobile traffic analysis in user space. *CoRR abs/1510.01419* (2015).

[3] Ren, J., Rao, A., Lindorfer, M., Legout, A., and Choffnes, D. ReCon: Revealing and Controlling PII Leaks in Mobile Network Traffic . In *ACM MobiSys* (2016).

[4] Vanhoef, M., and Piessens, F. All your biases belong to us: Breaking rc4 in wpa-tkip and tls. In *24th USENIX Security Symposium (USENIX Security 15)* (Washington, D.C., Aug. 2015), USENIX Association, pp. 97–112.