The Use of Certificate Pinning in Android Apps

Clifford Bakalin, Alex Brassel, Preston Tong

ABSTRACT
Certificate pinning has been established for years, yet many apps do not integrate it. Although cert pinning is an effective way to prevent MITM attacks, a prior study has discovered that many apps lack this safety mechanism. This study determines that there has been a rise in applications that use cert pinning in the last couple of years. Furthermore, our study is a measurement analysis on how Android applications have changed to use cert pinning overtime, whether it be changes in cert pinning or implementation or the advent of cert pinning integration with the app. We find that 3 of the 9 apps which we identified as having cert pinning have started using cert pinning in the last 2 years while 2 of our studied applications have changed how they handle the cert pinning protocol. We have discovered apps not listed in the prior study and discrepancies in the study from our results within the two year gap that they have occurred.

1 INTRODUCTION

As phones become more ubiquitous in modern society, mobile security becomes even more prevalent. With unsecure public WiFi access points and unsuspecting users, man-in-the-middle (MITM) attacks can be easily executed. MITM attacks are when malicious actors intercept traffic between a user and a server with intent to collect/alter/spoof data not meant for them. In order to defend from MITM attacks, apps can implement cert pinning to ignore malicious certs and only trust "pinned" certs. Cert pinning is when apps are configured with a set of "pinned" certs that the app trusts. As a result, the app will not have to rely on other installed certificate authorities (CAs) to verify the validity of connections and communication with intended servers. Malicious CAs can be installed on mobile devices that sign data and effectively allow attacks to perform MITM attacks. With cert pinning, these malicious CAs will no longer be used in the verification process and MITM attacks will no longer work.
There are incentives and disincentives with this approach. On one hand, as discussed above, it defends against MITM attacks. On the other hand, whenever the cert changes or expires, an update must be pushed to the app. Additionally, cert pinning must be done manually for each domain an app would like to access, which can quickly become impractical.

A compromising alternative that appears several times in this survey is "CA pinning" where, rather than pinning a specific cert, a root certificate authority is selected and the app will always attempt to authenticate a cert through this root.

How many apps employ either CA pinning or cert pinning? When did they start to do so? These are the motivating questions for this survey.

2 RELATED WORK

A prior 2017 study consists as the primary base of our research [2]. The study includes an overview of certificate pinning on android apps. Out of 7,258 mobile apps, 244, or 3.36%, of the apps were found to utilize pinning [2]. The researchers had more than 5,000 users download their custom app titled Lumen Privacy Monitor. The app collected data from user phones and sent them back to the researcher's servers for analysis. Unfortunately, we were unable to have such a broad range of users generate traffic data for our study. Instead, we ran Lumen on top of automatic app download and traffic generation. The previous 2017 study performs analysis on which types of apps use cert pinning and how many. However, our study aims to measure how apps have changed over time in regards to cert pinning. This includes how long they have used cert pinning and any changes to their methods.

3 DESIGN

This section discusses the process of collecting APKs, detecting apps for cert pinning, and analyzing APK versions.

3.1 APK Scraping

In order to perform our analysis, we first needed to obtain a source of data with which to do so. We were specifically interested in historical trends of cert pinning; consequently, we needed to obtain historical versions of any APK we observed using cert pinning in our study. This ruled out using the Google Play store as our primary source, as we could not find a way to download past versions at will; rather, we ended up investigating various APK mirroring sites, before committing to using APKPure, due to their easy interface for searching, as well as an ability to discriminate between versions based on architecture and DPI.

In order to begin our search, we needed to obtain a list of app ids. Based on the "top" categories available from the Google Play store [4]. we spawned an instance of a scraper per category. On each scraping thread, we collected app ids as follows. First, we hooked into a new instance of chromedriver using Selenium's python bindings. Then, we scrolled through the available app ids, recording them and their category information as went. Each thread piped their gathered information in chunks to a shared queue. Finally, a static API was made which, when called, generates app ids and metainfo one by one.

Next, we used a similar scraper to download app versions from a given app id on APKPure. It worked by navigating to the search page for an app id, and then inspecting the links for the results for a link to an app with the correct id. If none were found, it reported no result. Then, it tried to navigate to the appropriate versions page using the requests library. If it received a 404 response, it noted that no historical versions were available for the app in question, and skipped the app id. This did lead to some unfortunate trade-offs; for example, Facebook was not on the site, but Facebook lite was. However, in order to remain consistent, and because we had no automatic way of detecting this, we skipped such close mis-matches. After finding a versions page, it was presented with one of two options. The first option was a direct link of different versions per version (dpi, architectures, etc). If it found such a page, it filtered based on the specs given to the scraper at initialization. For example, we didn't allow certain DPIs or architectures. The second option was a list of downloads for past versions without any direct filtering options. In this case, we merely directly downloaded the page and filtered manually at analysis time. The different concurrent searches were fed to a single shared downloader which was also responsible for rate-limiting downloads; we did not wish to be throttled by APKPure itself. We limited our downloads to 3 concurrent downloads per minute.

The overall process is approximately as follows. First, an instance of the Google play scraper is launched which populates a producer-consumer queue with app ids in chunks. Then the aforementioned downloader is launched, which consumes from a second producer-consumer queue as download links become available. Finally, the APKPure scraper worker threads are launched which consume from the first queue and populate the second queue as they find appropriate links.

At first, we set the downloader to only download the most recent version from each app; it saved the other versions in order to cut down on future costs. Next, we examined each of the downloaded apps to determine whether the most recent versions do cert pinning. From the subset of results that we then determined *were* doing cert pinning, we conducted an additional round of downloads to collect all historic versions. We used this data to determine *when* the cert pinning began.

3.2 Cert Pinning Detection

As a product of the 2017 study, researchers developed Lumen, a mobile network analysis app that has proven to be useful for our current study [3]. Lumen is an android app that runs a proxy in the background and collects app network traffic. The collected data is logged within its designated path at /data/data/edu.berkeley.icsi.haystack/databases/haystack.db. Lumen is able to act as a MITM proxy after requesting the user to install and trust a CA specifically for signing intercepted traffic. Because Lumen requires a GUI for creating and trusting the CA, in addition to starting traffic collection, we were unable to seamlessly link cert pinning detection to our headless ubuntu server, and in turn, developed a script to pull APKs from the server onto a separate test environment, while maintaining a list of previously tested APKs that should not be pulled.

An android emulator is installed on the test environment running API level 23 (Marshmallow) and x86 architecture. As of the introduction of API level 24 (Nougat), apps no longer trust user installed CAs by default. Apps API level 24 and above have to specifically be configured to trust user-installed CAs [2]. This new setting impedes installed malicious CAs from allowing MITM attacks. As a result, we have decided to only test apps targeting API level 23 and below. Although we are only testing this subset of apps, we have found that many current versions accommodate API levels under 23.

In order to distinguish cert pinning, we can check Lumen for apps that throw a CERTIFICATE_UNKNOWN Java exception. When an app uses cert pinning, the app will terminate its connection with Lumen's proxy and Lumen's proxy will throw an Android TLS exception [2]. This exception is mapped to the CERTIFICATE_UNKNOWN exception in Lumen's logs [2]. Apps that do not use cert pinning will not throw the specified Java exception.

To facilitate testing of many APKs, we developed a script to automatically install the app from the APK, run the app with random input, and finally delete the app to clear up storage. The android emulator's default RAM setting of 1 GB was not enough to run Lumen safely in the background without crashing, so we bumped the android emulators RAM up to 4 GB.

Data can be retrieved by locating Lumen's database file in Android Studio's device file explorer. We created another python script to pull the haystack.db from the emulator and select Android apps that contained a tls_error column with the CERTIFICATE_UNKNOWN error. The script utilizes the SQLite3 library for integration with python. These apps can then be noted for later analysis.

3.3 Analysis

After we determined which applications utilized cert pinning, we had to find the certs that were pinnined. We hit a major obstacle right away as there is no standard for cert pinning. This meant that with so many ways to pin a cert, automating the process would be difficult. Eventually, we decided to brute force the process, which allowed us to determine three main ways application developers decided to pin their certs. After finding the certs needed, we were able to look in roughly the same place when going through the various builds of the application in question. Once we had a log of the certs, we could start answering some of our questions of how often certs change, which apps cert pin, and how apps differ in their cert pinning protocol.

4 RESULTS

We found that while there is no standard in cert pinning, there are three primary ways that the apps we found pin their certs. Some applications like Dropbox use a trustmanager, which is a part of Android's native library. However, this method is now considered obsolete for cert pinning and is to be replaced with Android's new *network_security_configuration.xml* file, which is to be more secure. While some of the apps we found used this method, there are other applications that do not use either of Android's security implementations and instead use a third party http client (usually okhttp) to manage their cert pinning. Libraries like these allow the developer to use the hash of a cert key rather than the key itself, which prevented us from reading the original cert, but is ultimately more secure than the other two methods.

Out of the 86 applications we tested, we found that 9 applications use cert pinning. Compared to the previous 2017 study by Razaghpanah et al [2], which found that only 3.36% of applications used cert pinning, we have seen an increase in the cert pinning protocol. However, this increase could be attributed to a variety of reasons due to our research methods. We had a smaller sample size than Razaghpanah et al (86 compared to 7,258), and we also only selected applications that were considered to be popular. Razaghpanah et. al [2] performed more of a passive choosing by measuring just what users already had on their devices.

There was also a small discrepancy we noticed between our work and Razaghpanah et al. While the 2017 study found that certain apps like Facebook used cert pinning, in our research we found that facebook does not cert pin. It could be the case that Facebook no longer pins their certs, but we do not believe this to be likely. One other common discrepancy we found is that select apps like Instagram use CA bundling rather than cert pinning. CA bundling is a similar but different protocol which is not in the scope of this study, so we did not have time to look into this more.

On the other hand, out of the 9 applications we found that performed cert pinning, we found that 3 of the applications had started implementing cert pinning within the last 2 years.

While we found an increase in the number of applications that used cert pinning, there are some downsides to cert pinning. By pinning a cert, application developers are limiting how often they can update their certs. This usually means that developers must push a new app update whenever their certificates change unless they wish their applications to stop working. Usually this would be a hassle on both the developer and the user who now has to update their apps whenever a certificate is changed. However, this problem can be circumvented by how we observed apps cert pinning. Most of the apps we found that utilized cert pinning did not pin the certificate for the target site, but rather a parent certificate authority which presumably signed the certificate for the target site. While this does get around the constant updating issue, we believe this form of cert pinning is essentially meaningless. According to Razaghpanah et al [2], android apps do not check for certificate revocation, nor do they check crl lists. Hence, since these pinned certs are not being checked for validity, and not the certificate of the developer, they might as well not use cert pinning.

Some applications like Twitter pinned their certs, however their cert file was encrypted so it was not possible to read the certs themselves. However, after looking at previous builds of the twitter app and performing a diff on the cert file showed no difference on the certs through time which would imply that the certs used were CA certs, they used certs with a long lifetime, there had been no data breach in the time period of the builds we had, or that after a breach, the certs were not changed.

5 CONCLUSION

Prior studies have found that only a small percentage of apps use cert pinning. Our study concludes that there has been an increase in apps that implement cert pinning from 2017. Although, this result shows a favorable shift towards a more secure environment for mobile apps, it is not indicative of more secure apps. We found that many of the pinned certificates were for CA authorities rather than for particular services.

5.1 Future Work

In the future, we plan to expand our APK sample size. We were greatly limited by both the constrained time frame for the project and the rate at which we could download APKs. Now that we have a streamlined way to pull APKs from APKPure, we can expedite the overall process. The main bottleneck in our pipeline is the manual analysis on apps that use cert pinning. A primary goal of this study would be to develop a standard method of automatically checking changes in certs.

In addition, some apps are not hosted on APKPure, such as banking apps like Bank of America or Capital One. A future task would be to accomodate scraping other APK sites to include missing apps. Also, we only performed our study on apps that targeted x86 architecture and no dpi. Some versions of apps are developed for only ARM architecture and a specific dpi. In order to broaden our studies scope, these versions need to be incorporated.

6 REFERENCES

[1] Y. Liu, W. Tome, L. Zhang, D. Choffnes, D. Levin, B. Maggs, A. Mislove, A. Schulman, C. Wilson. An End-to-End Measurement of Certificate Revocation in the Web's PKI. *IMC*, 2015.

[2] A. Razaghpanah, A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, P. Gill. Studying TLS Usage in Android Apps. *ICIR*, 2017.

[3] Lumen Privacy Monitor. https://play.google.com/store/apps/details?id=edu.berkeley.icsi.haystack&hl=en_US, 2019.

[4] Google Play Store, https://play.google.com/store/apps/top, 2019.