Notary-assisted Certificate Pinning for Improved Security of Android Apps

Georg Merzdovnik, Damjan Buhov, Artemios G. Voyiatzis, and Edgar R. Weippl
SBA Research
Vienna, Austria
Email: {gmerzdovnik,dbuhov,avoyiatzis,eweippl}@sba-research.org

***Abstract*—The security provided to Internet applications by the TLS protocol relies on the trust we put on Certificate Authorities (CAs) issuing valid identity certificates. TLS certificate pinning is a proposed approach to defend against man-in-the-middle (MitM) attacks that are realized using valid albeit fraudulent certificates. Yet, the implementation of certificate pinning for mobile applications, and especially for Google Android apps, is cumbersome and error-prone, resulting in inappropriate connection handling and privacy leaks of user information.**

We propose the use of TLS notary-assisted certificate pinning at the Android Runtime level. Our approach defends against a wide range of MitM attacks without needing to update the application using TLS. Furthermore, by relying on the collective knowledge of the trusted TLS notaries, we increase both the security and the usability, while at the same time we remove the burden for the user making trust decisions about system security issues. We describe a proof-of-concept implementation demonstrating its capabilities and discuss the next steps necessary towards general availability of our solution.

***Keywords*-Security; TLS; certificate pinning; Google Android; mobile apps; man-in-the-middle attacks**

I. Introduction

The use of mobile devices steadily increases, as they are becoming a central hub for information collection and consumption as well as a remote control for the Internet of Things. An increasing number of developers implement applications for mobile platforms and thousands of new applications are published each month. As of May 2016, the Google Play store offers a total of 2,178,494 apps[1]. However, with this huge amount of apps, it is inevitable that several problems arise. More and more private information are collected and transferred via mobile devices. Therefore, special care has to be taken when this data transmissions occur over untrusted or potentially monitored networks. This is often the case for smartphone applications (apps) accessing web content over open WiFi networks installed in public places [1].

It is, thus, necessary to verify the authenticity of the remote entity that the application is interacting with, so as to ensure the protection of the sensitive information. Many applications and libraries reuse already-deployed web infrastructure. A large volume of information is transmitted using the HTTP protocol. The Transport Layer Security (TLS) protocol is often used to secure the underlying network connection, combined into an HTTPS protocol.

[1]http://www.appbrain.com/stats/number-of-android-apps

TLS certificate pinning is a proposed approach to defend against man-in-the-middle (MitM) attacks. The implementation of certificate pinning for mobile applications, and especially for Google Android apps, is cumbersome and error-prone. This results in inappropriate connection handling and privacy leaks of user information.

In this paper, we propose a design to realize TLS notary-assisted certificate pinning as a means to transparently defend against MitM attacks on behalf of all installed applications in a device. The collective knowledge provided by trusted notary services can increase both the security and the usability of the Android devices.

The contributions of this paper are the following:

- We describe a new design for implementing certificate pinning at the Android Runtime layer defending against a revised threat model with stronger adversaries.
- We enrich the certificate pinning decision with TLS notary-assisted information.
- We evaluate the proposed design and show that *both* security and usability are increased *without* introducing noticeable overhead.
- We describe a proof-of-concept implementation of the design for the Google Android platform that demonstrates its applicability and feasibility.

The rest of the paper is organized in five sections. Section II provides the necessary background information. In Section III, we describe the overall design of our approach and we perform an evaluation of the design in Section IV. Section V describes a proof-of-concept implementation of our design, while Section VI provides a discussion and the conclusions of our work.

II. Background

A. TLS and X.509 certificate validation

The goal of the Transport Layer Security (TLS) protocol is to provide data confidentiality and integrity between two communicating computer applications. An often-used example use of TLS is for securing communication between a web browser (client) and a web site (server). The protocol is defined in various proposed standards by the IETF, including among others RFCs 2246, 3546, 4346, 4366, 4680, 4492, 5246, 5288, 5746, 6176, and 6655. The origins of the TLS protocol date back to 1993, when SSL v1.0 was defined. The current version is 1.2 and the next major protocol revision, TLS v1.3, is expected soon [2].

TLS can, optionally, authenticate the identity of the two communicating parties using public-key cryptography.

This is widely used for at least authenticating the server side, i.e., for proving that a (web) client indeed connects with the intended (web) server. The server authenticity is based on the Internet X.509 Public Key Infrastructure Certificate, as defined in RFCs 5280 and 6818.

When a client connects to a server over TLS, the server presents its certificate for proving its identity. The server certificate should be signed by a certificate authority (CA) that the client trusts, either explicitly (e.g., by having the user click on a warning message) or implicitly (e.g., by consulting its "trust store", i.e., a set of pre-distributed "root" certificates through which a chain of trust is built).

All root certificates are considered equally-trusted. Hence, any of the CAs can issue an equally-valid certificate for a given server. If any of the CAs is compromised, then it can be tricked to issue a fake but valid certificate for a server.

To cope with this inefficiency, various approaches have been proposed. A Certificate Revocation List (CRL) can be periodically distributed stating which of the issued certificates by a CA are not valid anymore. This can still leave a window of opportunity for an attacker, until a client updates its CRL. Online Certificate Status Protocol (OCSP) was proposed to solve this problem by allowing a client to contact online (at the time of a TLS connection setup) the CA and verify the validity of the presented certificate. Yet, this extra connection with a third-party server can introduce significant latency in page loads, especially in environments with mobile clients (e.g., smartphones) connecting over wireless or cellular links. OCSP stapling (formally, the TLS Certificate Status Request extension, standardized in RFC 6066) removes some of this burden by allowing the server to append (staple) a time-stamped OCSP response signed by the CA during the initial TLS handshake.

HTTP Public Key Pinning (HPKP) is standardized in RFC 7469. It allows a server to "pin" the hashes (fingerprints) of the valid certificates during a connection. On subsequent connections the client can check the hashes of the presented certificate. If they do not match the known ones, then the client can assume that a man-in-the-middle (MitM) attack is taking place.

HPKP cannot defend against impersonation attacks mounted when a client visits a previously unknown server. In this case, even if HPKP is employed by the server side, the client has to inherently trust the unknown hashes presented by the MitM attacker.

B. TLS Notary Services

TLS Notary Services (or simply, "notaries") are a defense against impersonation attacks utilizing crowd knowledge, as collected by notary servers. The key observation is that an attacker is not able to intercept all possible communication links with a server and mount MitM attacks. Thus, notaries can collect certificates from different points of observation (i.e., perform multi-path probing), which cannot be intercepted concurrently or altogether. As illustrated in Figure 1, when a client is presented with a (possibly impersonated) server certificate, it consults the publicly-available notaries and compares the received results in order to detect the attack.

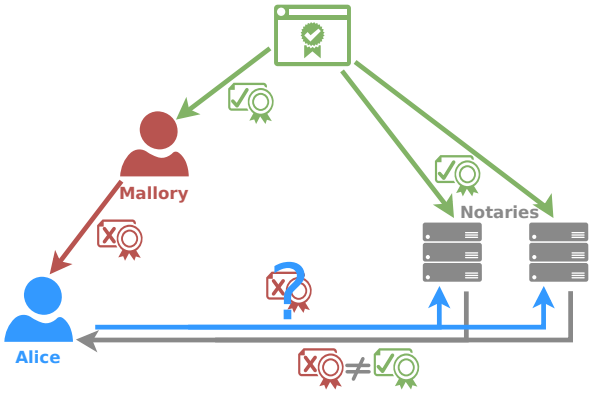

Figure 1. Detecting impersonation attacks using TLS Notary Services

Perspectives[2] and Convergence[3] are two example implementations of *active* notaries. The ICSI Certificate Notary[4] is an example implementation of a *passive* notary. It builds its certificate database by passively monitoring traffic at multiple independent Internet sites. The database can be queried through a public DNS interface.

The scalability issues of Convergence are studied in [3]. Laribus [4] is an attempt to build a P2P notary service exploiting social-connectivity graphs so as to remove the need for centralized notaries. A longitudinal study on the availability and functionality of publicly-available TLS Notary Services is presented in [5].

C. Certificate validation in Google Android

The Google Android operating system supports inherently the TLS protocol. Supported devices come with a bundled trust store containing more than 150 certificates of root certificate authorities. This list is initially populated by Google but may be further customized by third parties, such as the manufacturer of the device and the cellular network operators [6]. The size of the list steadily increases over the years, raising the concern of the research community regarding the trust model of Internet-deployed TLS [7]. elopers, developers, developers! Android developers can integrate TLS functionality and certificate validation in their apps for secure communication with servers. This functionality is offered through an API of the operating system. There are currently three alternatives for realizing this:

- TLS and certificate handling by the operating system (default handling).
- Custom validation by the application developer.
- Utilization of the "Network Security Configuration" functionality in the forthcoming version of the Android version system (Android N).

[2]http://www.perspectives-project.org/
[3]http://www.convergence.io/
[4]http://notary.icsi.berkeley.edu/

1) Default handling: The default certificate handling is done automatically by the operating system and frees the developer from all housekeeping operations. At the same time, it provides the least of the control on how exactly the validation is performed. The developer must procure and install a server certificate signed by one of the (many) trusted CAs that come with the operating system and take appropriate action in case the certificate is expired or the root CA is removed, e.g., due to a root CA compromise. The latter is an unusual event but yet one that has indeed occurred in the past[5].

Certificate pinning is currently not supported in this mode. Thus, the security-cautious developers must implement by themselves custom validation to realize this helpful functionality of certificate handling.

2) Custom validation: A second option for the developers is to handle the certificate validation by themselves. This offers the greatest flexibility and allows support for self-signed certificates as well. At the same time, the application developer is solely responsible for the flawless implementation of the admittedly complex procedure of chained certificate validation. Until now, custom validation is the only means to implement certificate pinning functionality.

3) Android N: Android N is the codename for the upcoming release of the Android operating system (probably version 7.0), announced in March 2016. In Android N, apps can customize the behavior of their secure (HTTPS, TLS) connections safely, without any code modification, by using the declarative *Network Security Config*[6] instead of using the conventional error-prone programmatic APIs (e.g., X509TrustManager).

Android N will be the first version supporting certificate pinning at the application level as a means to defend against MitM attacks[7]. The pinning information will be bundled within the `network_security_config.xml` resource file of each application.

D. Issues with certificate pinning

Earlier research findings suggest that the developers cannot cope sufficiently with certificate validation in general and certificate pinning in particular. A study performed in 2012 revealed that more than 1,000 applications out of a sample of 13,500 included a completely wrong implementation of the validation procedures [8]. A year-long study between 2013 and 2014 revealed that the situation actually does not improve but rather it is getting worse over the time [9].

Interviews with developers of applications with broken validation indicate that the developers are not fully aware of the security implications of such erroneous implementations [10]. Quite often, the developers do not consider MitM attacks as a threat altogether. Rather, their aim is solely to implement self-signed certificates because it is more convenient to them, despite the wide availability of free server certificates.

"Pin It!" is a novel approach to offer certificate pinning functionality transparently to the applications [9]. This is achieved by intercepting system calls related to certificate handling and enforcing the pinning with the assistance of the user. If a new certificate is detected, the user is asked to confirm the new certificate hash for future reference. This approach assumes that the first connection with the (web) server is not tampered (the so-called "trusted-on-first-use" or "TOFU" principle) and that the user can make informed decisions about the presented certificates. User surveys suggest that the latter is a strong assumption as the users fail en masse even in the simpler task to judge whether their browser session is protected by TLS or not [10]. It should be noted however that the "Pin It!" approach is the only feasible one at the moment and does not require an application update by the developer. It offers adequate protection for the security-conscious users against a careless developer who implements incorrectly certificate validation and exposes the private data of the users.

The Android N approach, when it will become available, is a step towards the right direction as it simplifies the integration of certification pinning. It is less invasive in nature and does not require a rooted device, since the *Network Security Config* is an integral art of the forthcoming operating system. Still, it requires the prompt action of the developer, in order to take advantage of the new functionality.

It remains to be seen if this approach will gain popularity among the developers. Also, if it will be possible for all Android devices to upgrade to the new operating system and how fast; if not and in the meantime, the developers will have to opt for a dual implementation of their application: one that supports certificate pinning through Android N and one through other means (or, even worse, not at all, creating an illusion of security and confusion to the users regarding the offered security level).

We also note that in the case of Android N, certificate pinning will occur on a per-application basis. This will be realized using a bundled resource file. Hence, if the pinning information must be updated (e.g., due to a security incident), the developers must go through the whole, time-consuming process of delivering an application update to their users, introducing further delays and extending the window of exposure to MitM attacks.

III. Design and System Architecture

In this Section, we describe the design and the architecture of an enhanced system supporting certificate pinning for Google Android mobile applications.

Our design aims to address the shortcomings of the approaches mentioned in Section II. More specifically, we aim to offer: (i) increased security, by relying on the collective knowledge of TLS notaries; (ii) transparent

[5]https://en.wikipedia.org/wiki/DigiNotar#Issuance_of_fraudulent_certificates

[6]https://developer.android.com/preview/api-overview.html#network_security_config

[7]https://developer.android.com/preview/features/security-config.html#CertificatePinning

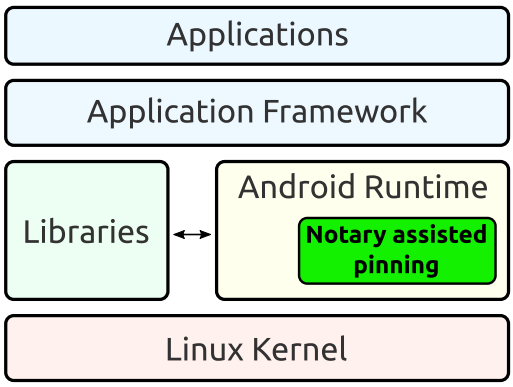

Figure 2. Notary-assisted Pinning in Android Architecture

protection for all installed applications; and (iii) increased usability, by reducing the user burden and involvement in the bare minimum.

A. Threat model

We assume a threat model where an attacker is able to launch MitM attacks on the connection between an application installed on a Google Android device and its respective web server. The attacker can intercept the TLS connection phase and inject a fraudulent certificate towards establishing a fake connection. Further, we assume that the attacker is located closely to its victim (network-wise), e.g., in a fake wireless access point but they cannot intercept communications in other parts of the Internet (e.g., between the web server and the TLS notary service nodes).

B. System design

The design of our system follows closely the system architecture of Google Android. The latter comprises four layers, as depicted in Figure 2. Following a bottom-up approach, the first layer is the *Linux Kernel*, containing all the necessary drivers that power all of the functionalities presented by the Android applications. The next layer contains the essential *Libraries* and the *Android Runtime*. The latter consists of the *Core Libraries* and the *Dalvik Virtual Machine* (DVM). Every Android application executes in its own DVM. The *Application Framework* layer provides functionalities such as views, activity manager, window manager, telephony, and location services.

Our design introduces a "Notary-assisted Pinning" component in the Android Runtime layer. The component can interface directly with the low-level functionality offered by the Linux kernel, without application intervention. Furthermore, at this level, it can hook and protect transparently all installed applications, without requiring special application logic implemented in the latter.

C. Component functionality

When an application initiates a TLS connection, the related Android library call is intercepted and the control

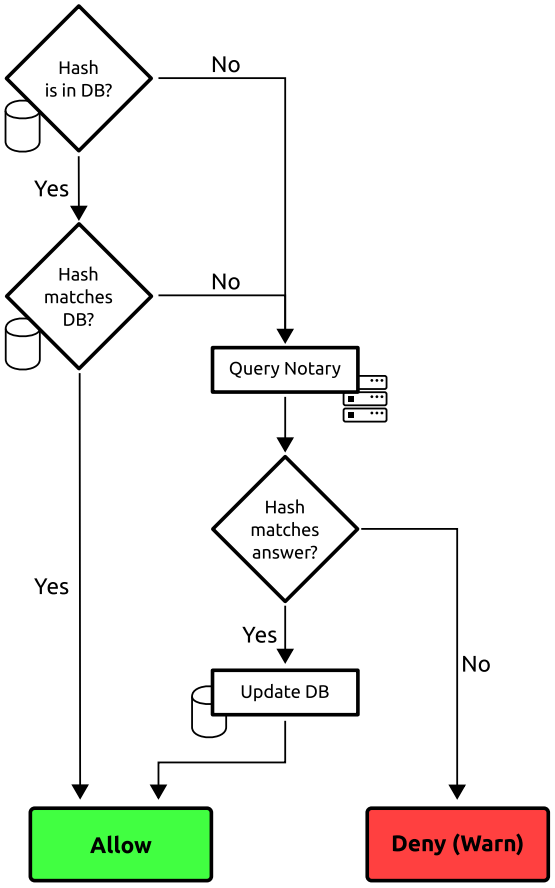

Figure 3. Certificate Validation Workflow

is passed to the component. The component workflow is depicted in Figure 3.

The component will first check if the presented certificate refers to an already pinned web server certificate. If yes, then the certificate hash is compared with the pinned one. In case a match is found, the connection is allowed and the control passes back to the calling application. In case a match is not found, the component initiates a transaction with one or more TLS notaries and checks if the hash is known to them. Once a positive conclusion is made, the control passes back to the calling application. If a negative conclusion is made, the connection is terminated and the control passes back to the calling application. If a conclusion cannot be reached (e.g., the notaries are not accessible or there is no consensus in their replies), the component can either (i) terminate the attempted and possibly untrusted connection or (ii) as a last resort, generate a warning message and ask for action confirmation by the user.

If an entry for the specific web server is not already present in the pinning database, i.e., the web site was not visited until now, then the component will again resort to the collective knowledge of the notaries. If a positive decision can be made, the pinning database is updated accordingly.

IV. Evaluation

The evaluation of our design is based on three dimensions, namely functionality, security, and usability. We analyze each of them in the next paragraphs.

A. Functionality

The proposed design does not require modifications to the functionality of the installed applications or some additional effort for the developers, as it operates transparently at a lower layer of abstraction (namely, Android Runtime). Furthermore, the system interactions involve only actions that are common to TLS processing and certificate validation (e.g., cryptographic computations and network exchanges). Some response latency can be expected and considered acceptable, as it is similar to periodically consulting a CA using OCSP.

Our design does not interfere with the recently-announced Android N certificate pinning functionality, as the latter is implemented in the application layer. Thus, we consider that our design will actually *enhance* the provided functionality and also provide an additional layer of defense, in the meantime of distributing a new version of the application through the app stores. Finally, the proposed design neither depends on a specific Android device model nor a specific application.

B. Security

We argue that our proposed design offers increased security for the device owners. We are among the first to offer certificate pinning functionality for all Android applications. This provides a first layer of defense against MitM attacks that present a different albeit valid certificate for a known web server. Even if such a certificate is presented, our design relies on the collective knowledge of trusted TLS notary services for evaluating the new information. This is a significant improvement over the "Pin It!" alternative, which relies on the user comprehending the TLS warning messages and making an informed decision [9].

The utilization of TLS notaries provides an additional layer of defense against TOFU-based attacks. While it might be the first time for a device to be presented with a specific certificate, it is highly improbable that the notaries have not seen it already. Hence, the local lack of knowledge is accommodated through the collective knowledge of the notaries.

We expect that the developers themselves are among the first to install their application, once it becomes available in the app stores. Hence, they will feed the notaries with trusted information at a very early stage, even before the general availability of the application. This can happen, for example, during the period that they bring online their web server and perform the necessary pre-deployment quality assurance tests.

TLS Notary Services operate under the multi-path probing principle [5]. Hence, for launching a successful attack, one must be able to interfere with all paths to the notaries and inject fake certificates in their databases. This attack is outside the threat model that is described in Section III-A. Yet, we note that even if this attack is successful in first place, the window of opportunity for an attacker would be rather small, as we expect that the notary operators will sooner than later detect the poisoning and remove the fraudulent certificates.

The above analysis leaves one path for an adversary to exploit. This is the communication path between the Android device and the notaries. At this stage, the adversary must present an unseen certificate to the device, so as to force the communication with the notaries. If the device cannot communicate with the notaries, then our component will (preferably) drop the TLS connection as well. We consider this as a better alternative to issuing a warning to the user for further action but still it is a configurable option. If the connection is dropped, then no information will be transmitted and leaked to the intercepting web server that is under the control of the adversary. Hence, the security is maintained. It should be noted that this behavior leads to a denial-of-service (DoS) attack. Protection against DoS attacks is outside the scope of our design and of certificate pinning in general; the aim is to secure the transmitted information from eavesdroppers. In this case, some other countermeasures should be employed (e.g., switching to an alternative network, for example a cellular connection, or delaying transmission until the device moves away the network reach of the adversary).

An adversary could also try to impersonate as a notary or alter the responses of the legitimate notaries, instead of denying the connection to the notaries altogether. By and large, this depends on the implementation of the notary query interface. If a secure connection is realized for this (e.g., over TLS), then the adversary will not be able to intercept or impersonate the notary. This leads to a TOFU-based attack scenario again. However, the certificates of the notaries can be embedded in our component and thus, defend against this threat. We expect that the notaries will be well-defended, using state-of-the-art technology. Thus, we consider as minimal, if existent at all, the risk of revocing their certificates due to a security incident. Even in that unfortunate case, it will take just an application update to restore the correct functionality. Furthermore, since the notaries do not operate on a per-application basis, this would not be a targeted attack against a specific application, device, or user but rather a generalized attack against the notary infrastructure itself.

If a clear text connection is realized for the communication with the notaries (e.g., the DNS-based query interface of ICSI Certificate Notary), then there is always the possibility for an adversary to manipulate the responses. Hence, it is necessary to realize an underlying secure channel, so as at least be able to detect fake responses (e.g., require that all responses are signed with a trusted key).

C. Usability

We consider that our design improves the usability of certificate pinning for the device user, on top of the

increased security. The "Pin It!" approach involves the device user in the trust decision for each and every certificate that does not match the stored one and for each and every newly-visited web site, where no information could have been stored already. This is far from optimal, especially if this involvement results in breaking their mental model for their primary task at hand, so as to cope with a secondary one [11].

Our approach avoids the involvement of the user as much as possible and relies on the collective knowledge of the TLS notaries instead. As depicted in Figure 3, the design relies on user involvement as an optional step (bottom right). This happens *only* when the certificate hash presented by the visited web site does not match (i) the already-pinned one and (ii) the one that notaries are aware of or (iii) have not ever seen a certificate for this web site. Even in this very rare case, it is a configurable option either to deny the connection automatically (preferable) or ask the user to confirm and continue their visit at their own risk.

It should be also noted that our approach works automatically for all installed applications, without user intervention or action, further reducing the burden for dealing with secondary security tasks. At the same time, the user is relieved from the risk of accepting a forged certificate and the certificate pinning functionality is performed automatically for them.

V. Proof-of-Concept Implementation

We implemented an Android component as a proof-of-concept (PoC) of our design to study its behavior in a realistic environment. There are many frameworks available that allow the on-device dynamic instrumentation and ease the development [12]. These frameworks allow to target, intercept, and modify specific library calls.

We opted for the Cydia Substrate framework[8], based on the analysis of [12]. Cydia Substrate is a dynamic instrumentation framework that enables interception and/or modification of system and application calls. Just by itself, Substrate does not provide any specific functionality. It acts as a platform (base) for developing particular modules, known as "extensions". The framework itself modifies the core of the Android system by injecting specific `jar` files. This gives the opportunity for the developers to intercept and manipulate the application and system calls. This is the only reason for its essential requirement which is the root privilege. Non-rooted devices keep this part of the system protected from performing any changes. Currently, the Google Android security model does not allow modification of the Android Runtime, hence, our component must be developed for a rooted device.

We base our PoC on the codebase of "Pin It!" that is readily available as open-source software[9] and realizes the basic certificate functionality already [9]. We enhanced the implementation to include the application logic to assist the certificate pinning decision based on information

[8]http://www.cydiasubstrate.com/

[9]https://github.com/dbuhov/pinningTrustManager

Figure 4. Notary-assisted certificate pinning

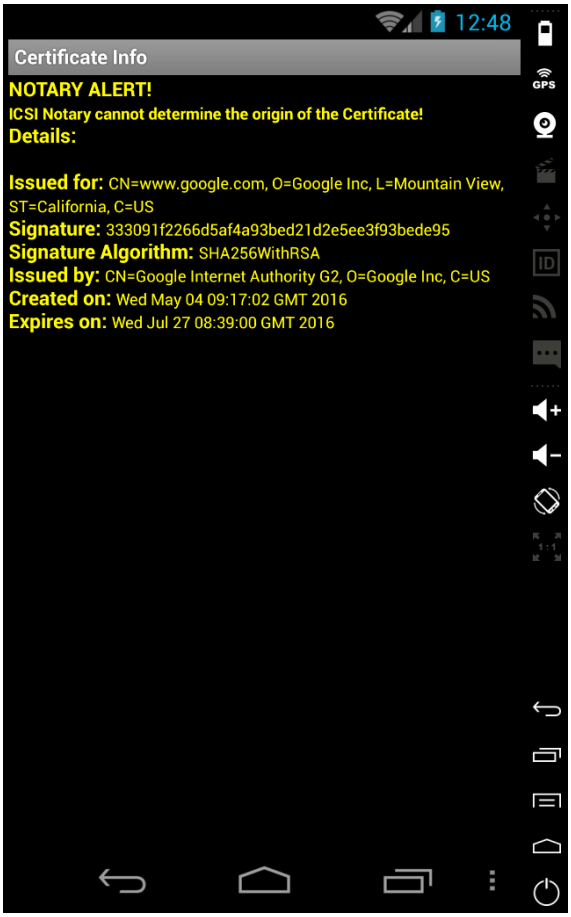

Figure 5. User notification from inside the PoC implementation

provided by notaries, as depicted in Figure 4. For the PoC, we integrate the query interface provided by the ICSI Certificate Notary service, which is publicly-available. The ICSI Certificate Notary service is consulted on the first encounter of a certificate and in case a certificate change is detected for an already pinned certificate. In all other cases, the received certificates are checked against the pinned ones, as previously. If a mismatch is detected, the connection is terminated and an appropriate notification is issued for the user (cf. Figure 5).

We experimented with our PoC implementation using valid and expired certificates. We did not notice any application problems (e.g., crashes or freezes) or noticeable delays. The query interface of the ICSI Notary Service was quite stable and the latency introduced by the additional DNS query was indistinguishable from normal network operations and the heavy cryptographic operations involved in the TLS connection setup. We note

the communication with the notary is very sporadic in nature anyway: only on first encounter and when the certificate changes. Empirical evidence from a recent study on TLS notaries suggests that certificate changes occur every few months at the most frequent [5]. Overall, the PoC implementation confirms that our design is sound and feasible to implement in the Google Android environment.

VI. Discussion and Conclusions

We presented a design and system architecture for notary-assisted certificate pinning in Google Android devices. While per-application certificate pinning will be introduced in the next version of the operating system, there is still a need for a user-centric protection against MitM attacks on all TLS connections, irrespective of the application readiness and the developer's capabilities to defend against such attacks. Furthermore, notary-assisted certificate pinning can be a layer of defense in the meantime that an application is exposed due to a certificate change and the inherent delay to update through the app store procedure.

The PoC implementation of our design confirmed that there is no noticeable performance penalty in those cases that a notary service must be consulted; yet the security improvement is very significant. A larger-scale validation is deemed necessary, possibly using the available dataset of [9], in order to study scalability issues. A production-ready implementation is envisioned for the future. For this implementation, a coordination with the notary operators is necessary, so as to implement a secure communication interface. Towards this direction, it would be helpful to see system vendors integrate such functionality in the operating system itself and thus, make the solution available for all Android devices and not only rooted ones.

Overall, our proposal for a notary-assisted TLS certificate pinning increases *both* the security and the usability of mobile devices, while reducing the burden of the users being involved in system security and trust decisions.

Acknowledgment

This work has been carried out within the scope of the Josef Ressel Center for User-Friendly Secure Mobile Environments (u'smile), funded by the Christian Doppler Gesellschaft (CDG), A1 Telekom Austria AG, Drei-Banken-EDV GmbH, LG Nexera Business Solutions AG, NXP Semiconductors Austria GmbH, and Österreichische Staatsdruckerei GmbH. Additionaly, the research was supported by the Austrian Research Promotion Agency (FFG) through the COMET K1 program.

References

[1] A. Dabrowski, G. Merzdovnik, N. Kommenda, and E. Weippl, "Browser history stealing with captive Wi-Fi portals," in *Proceedings of Workshops at IEEE Security & Privacy 2016, Mobile Security Technologies (MoST)*, 2016.

[2] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Internet Engineering Task Force, Internet-Draft, 2014, work in Progress.

[3] A. Bates, J. Pletcher, T. Nichols, B. Hollembaek, and K. R. Butler, "Forced Perspectives: Evaluating an SSL trust enhancement at scale," in *Proceedings of the 2014 Conference on Internet Measurement Conference (IMC 2014)*. ACM, 2014, pp. 503–510.

[4] K.-P. Fuchs, D. Herrmann, A. Micheloni, and H. Federrath, "Laribus: privacy-preserving detection of fake SSL certificates with a social P2P notary network," *EURASIP Journal on Information Security*, vol. 2015, no. 1, pp. 1–17, 2015.

[5] G. Merzdovnik, K. Falb, M. Schmiedecker, A. Voyiatzis, and E. Weippl, "Whom you gonna trust? a longitudinal study on TLS notary services," in *Data and Applications Security and Privacy XXX - 30th Annual IFIP WG 11.3 Conference (DBSec 2016)*. Springer, 2016.

[6] N. Vallina-Rodriguez, J. Amann, C. Kreibich, N. Weaver, and V. Paxson, "A tangled mass: The Android root certificate stores," in *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*. ACM, 2014, pp. 141–148.

[7] T. Fadai, S. Schrittwieser, P. Kieseberg, and M. Schmiedecker, "Trust Me, I am a Root CA! Analyzing SSL Root CAs in Modern Browsers and Operating Systems," in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES 2015)*. IEEE Computer Society, 2015, pp. 174–179.

[8] S. Fahl, M. Harbach, T. Muders, L. Baumgärtner, B. Freisleben, and M. Smith, "Why Eve and Mallory love Android: An analysis of Android SSL (in)security," in *Proceedings of the 2012 ACM conference on Computer and Communications security (CCS '12)*. ACM, 2012, pp. 50–61.

[9] D. Buhov, M. Huber, G. Merzdovnik, and E. Weippl, "Pin It! Improving Android network security at runtime," in *Proceedings of the 15th IFIP Networking Conference, Networking 2016*. IEEE, 2016.

[10] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking SSL development in an appified world," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (ACM CCS)*. ACM, 2013, pp. 49–60.

[11] C. Fidas, A. Voyiatzis, and N. Avouris, "When security meets usability: A user-centric approach on a crossroads priority problem," in *14th Panhellenic Conference on Informatics (PCI 2010)*. IEEE, 2010, pp. 112–117.

[12] D. Buhov, M. Huber, G. Merzdovnik, E. Weippl, and V. Dimitrova, "Network security challenges in Android applications," in *Proceedings of the 2015 10th International Conference on Availability, Reliability and Security (ARES 2015)*. IEEE Computer Society, 2015, pp. 327–332.