International Journal of Science and Research (IJSR)
ISSN: 2319-7064
ResearchGate Impact Factor (2019): 0.28 | SJIF (2019): 7.583

SSL Pinning in Android Applications: A Comprehensive Study

Naga Satya Praveen Kumar Yadati

DBS Bank Ltd
Email: *praveenyadati[at]gmail.com*

Abstract: ***The rapid growth in mobile device usage has sometimes led to a neglect of security in application development. While SSL/TLS has been a cornerstone for securing communications, it is not without vulnerabilities. One significant issue is SSL pinning bypassing. This paper explores security controls to mitigate SSL pinning bypassing, reviews existing bypassing techniques, and introduces two new methods. We conducted experiments on popular applications to assess the effectiveness of these controls and bypassing methods. Finally, we propose an applicability framework that links security controls to corresponding bypassing methods, offering guidance for pentesters and developers.***

Keywords: SSL pinning; security; mobile applications; Android; auditing; vulnerabilities; OWASP

1. Introduction

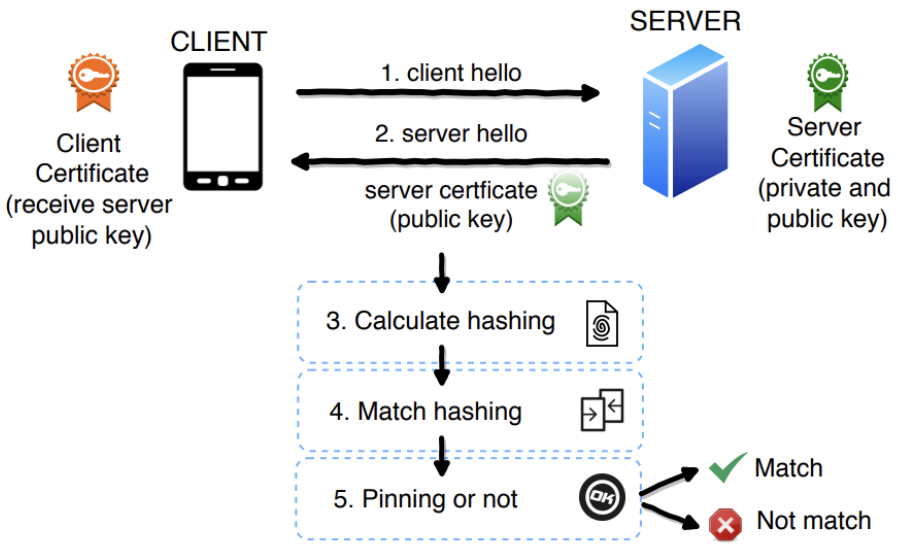
The increasing reliance on mobile devices for tasks traditionally performed on web services necessitates equivalent security measures for both environments. SSL/TLS (Secure Socket Layer/Transport Layer Security) has been widely adopted to secure internet communications, including HTTP protocols.

In 2018, over 1.5 billion smartphones were sold, contributing to a surge in security threats. Users frequently encounter scams through mobile applications, such as downloading unofficial app versions or exposing sensitive information. Many apps lack robust SSL/TLS validations, making them vulnerable to **Man-in-the-middle** (MiTM) attacks and other threats like replay attacks, eavesdropping, and session hijacking. Although SSL pinning, or certificate pinning, enhances security by ensuring the server's certificate is not compromised, it is not foolproof and can be bypassed.

The OWASP (Open Web Application Security Project) Mobile Application Security Verification Standard (MASVS) aims to standardize mobile app security requirements to protect data flows over insecure channels. This paper outlines how apps can be fortified with specific security controls to mitigate bypassing attacks.

We analyze SSL/TLS vulnerabilities and the importance of SSL pinning techniques, propose security controls to prevent bypassing, and evaluate popular Android apps to test bypassing methods. Two new bypassing methods are introduced, and a framework for applicability based on security controls is developed.

The paper is structured as follows: Section 2 provides the background, including OWASP Mobile Testing Guide and SSL/TLS protocols. Section 3 presents security controls against bypassing methods. Section 4 details our experimental approach, and Section 5 analyzes the results. The paper concludes with a summary and future work directions in Section 6.



2. Background

This section provides the foundational concepts for this study. It introduces the OWASP Mobile Testing Guide as a security model, describes the operation of SSL/TLS, and outlines SSL/TLS vulnerabilities to justify the research.

2.1 OWASP Mobile

OWASP is a global organization that creates standards for web application security. It offers various methodologies, including the well-known OWASP Top 10, which lists the most common vulnerabilities. OWASP develops Top 10 security risks for web, mobile, and IoT software. Our study uses the OWASP Top 10 Mobile, updated in December 2016, as a reference point.

Rank	OWASP Mobile Top 10 Risks
1	M1 - Improper Platform Usage
2	M2 - Insecure Data Storage
3	M3 - Insecure Communication
4	M4 - Insecure Authentication
5	M5 - Insufficient Cryptography
6	M6 - Insecure Authorization
7	M7 - Client Code Quality
8	M8 - Code Tampering
9	M9 - Reverse Engineering
10	M10 - Extraneous Functionality

2.2 SSL/TLS Protocol

SSL/TLS protocols secure communication over networks and are widely used to protect data transferred on the Internet. They employ a combination of public-key and symmetric-key cryptography to ensure data confidentiality, integrity, and authenticity.

2.3 SSL Pinning and Its Bypassing

SSL pinning involves embedding a server's SSL certificate or public key within an application to prevent man-in-the-middle attacks. The application compares the server's certificate with the embedded certificate during connection. If they match, the connection proceeds; otherwise, it is terminated.

However, SSL pinning can be bypassed through various techniques. Attackers might modify the application's code, use debugging tools to intercept communications, or exploit operating system vulnerabilities.

3. Security Controls for SSL Pinning

To defend against SSL pinning bypassing, developers can implement several security measures:

1) **Certificate Transparency**: This approach logs all issued certificates in public logs, helping detect fraudulent certificates.
2) **Network Security Configuration**: Android's network security configuration feature allows developers to specify security settings for app network communications.
3) **Code Obfuscation**: Obfuscating the application code makes it more challenging for attackers to understand and modify the code.
4) **Root Detection**: Implementing root detection mechanisms prevents attackers from using rooted devices to bypass security controls.
5) **Integrity Checks**: Ensuring application code integrity helps detect and prevent tampering.

3.1. Implementing SSL Pinning in Kotlin

Here’s how to implement SSL pinning in an Android app using Kotlin:

```
fun getPinnedHttpClient(context: Context): OkHttpClient {
    val cf = CertificateFactory.getInstance( type: "X.509")
    val caInput = context.assets.open( fileName: "server.crt").use { it: InputStream
        cf.generateCertificate(it) as X509Certificate
    }

    val keyStore = KeyStore.getInstance(KeyStore.getDefaultType()).apply { this: KeyStore!
        load( stream: null,  password: null)
        setCertificateEntry( alias: "ca", caInput)
    }

    val trustManagerFactory = TrustManagerFactory.getInstance(
        TrustManagerFactory.getDefaultAlgorithm()
    ).apply { this: TrustManagerFactory!
        init(keyStore)
    }

    val sslContext = SSLContext.getInstance( protocol: "TLS").apply { this: SSLContext!
        init( km: null, trustManagerFactory.trustManagers,  random: null)
    }

    // Create an OkHttpClient that uses our custom TrustManager
    return OkHttpClient.Builder()
        .sslSocketFactory(
            sslContext.socketFactory,
            trustManagerFactory.trustManagers[0] as X509TrustManager
        )
        .build()
}
```

4. Experimental Evaluation

We evaluated the effectiveness of these security controls by selecting popular Android applications and testing them against known and newly proposed SSL pinning bypassing methods. Tools such as Frida, Xposed, and custom scripts were used to attempt bypassing the SSL pinning implementations.

5. Results and Analysis

Our experiments showed that many popular applications still have inadequate SSL pinning implementations. While some applications employed multiple security controls, others relied solely on basic SSL/TLS validation, making them susceptible to bypassing attacks. Our proposed bypassing methods successfully circumvented SSL pinning in several applications, highlighting the need for comprehensive security measures.

6. Conclusion and Future Work

SSL pinning is crucial for mobile application security but is not infallible. Developers should implement multiple layers of security controls to protect against bypassing attacks. Our applicability framework offers guidelines for pentesters and developers to assess and enhance the security of their applications.

Future work will focus on improving the proposed security controls and developing automated tools to assist developers in implementing robust SSL pinning mechanisms.

References

[1] Li, D.; Guo, B.; Shen, Y.; Li, J.; Huang, Y. The evolution of open-source mobile applications: An empirical study. J. Softw. Evol. Process. 2017, 29, e1855.
[2] Unal, P.; Temizel, T.T.; Eren, P.E. What installed mobile applications tell about their owners and how they affect users' download behavior. Telemat. Inform. 2017, 34, 1153–1165.
[3] Kumar, R.; Perti, A. Security issues with self-signed SSL certificates. Int. J. Innov. Technol. Explor. Eng. (IJITEE) 2019, 8, 7S2.
[4] Lindgren, A.; Lindoff, B. On Estimating the Number of Worldwide LTE Cell-IDs and WiFi Aps. 2018. Available online: https://combain.com/uploads/Whitepaper_WorldWide_LTE_CellID_and_WiFi_APs_A.pdf (accessed on 3 September 2019).
[5] Anthi, E.; Theodorakopoulos, G. Sensitive data in Smartphone Applications: Where does it go? Can it be intercepted? In International Conference on Security and Privacy in Communication Systems; Springer: Berlin/Heidelberg, Germany, 2017; pp. 301–319.
[6] Khan, J.; Abbas, H.; Al-Muhtadi, J. Survey on mobile user's data privacy threats and defense mechanisms. In Proceedings of the 12th Iberian Conference on

Information Systems Technolo-Gies (CISTI), Lisbon, Portugal, 14–17 June 2017; No. 7975981.
[7] D'Orazio, C.J.; Choo, K-K.R. A technique to circumvent SSL/TLS validations on iOS devices. Future Gener. Comput. Syst. 2017, 74, 366–374.
[8] Razaghpanah, A.; Sundaresan, S.; Niaki, A.A, Amann, J.; Vallina-Rodriguez, N.; Gill, P. Studying TLS usage in Android apps. In Proceedings of the 13th International Conference on Emerging Technologies (CoNEXT 2017), Ingeon, Korea, 12–15 December 2017; pp. 350–362.
[9] Fahl, S.; Harbach, M.; Perl, H.; Koetter, M.; Smith, M. Rethinking SSL development in an appified world. In Proceedings of the ACM SIGSAG Conference on Computer & Communications Security (CCS 2013), Berlin, Germany, 4–8 November 2013; pp. 49–60.
[10] De los Santos, S.; Torres, J. Analysing HSTS and HPKP implementation in both browsers and servers. IET Inf. Secur. 2017, 12, 275–284.
[11] Mueller, B.; Schleier, S. OWASP Mobile Application Security Verification Standard v 1.1.4. Available online: https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide (accessed on 21 November 2019);
[12] Dhawale, C.A.; Misra, S.; Jambhekar, N.D.; Thakur, S.U. Mobile computing security threats and solution. Int. J. Pharm. Technol. 2016, 8, 23075–23086.
[13] OWASP Mobile Top 10. 2016. Available online: https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10(accessed on 22 February 2017).
[14] Kim, S.; Han, H.; Shin, D.; Jeun, I.; Jeong, H. A study of International Trend Analysis on Web Service Vulnerabilities in OWASP and WASC. In Proceedings of the 3rd International Conference on Information Security and Assurance (ISA 2009), Seoul, Korea, 25–27 June 2009; Springer: Heidelberg, Germany, Volume 5576, pp. 788–796.
[15] Szczepanik, M.; Jozwiak, I. Security of mobile banking applications. Adv. Intell. Syst. Comput. 2018, 635, 412–419.
[16] Hickman, K. The SSL Protocol. Netscape Communications Corp: Mountain View, CA, USA, 1995.
[17] Dierks, T.; Rescorla, E. The TLS Protocol Version 1.2; RFC 5246. Available online: https://tools.ietf.org/html/rfc5246 (accessed on 21 November 2019).
[18] Gu, X.; Gu, X. On the detection of fake certificates via attribute correlation. Entropy 2015, 17, 3806–3837.
[19] Varela-Vaca, A.J.; Gasca, R.M. Towards the automatic and optimal selection of risk treatments for business processes using a constraint programming approach. Inf. Softw. Technol. 2013, 55, 1948–1973.
[20] Oracle—Java Secure Socket Extension (JSSE) Reference Guide. 2018. Available online: https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html (accessed on 3 September 2019).
[21] OpenSSL. Available online: https://www.openssl.org/ (accessed on 3 September 2019).