GU J Sci, Part C,9 (2): 211-219 (2021)

Gazi University

Journal of Science

PART C: DESIGN AND TECHNOLOGY

http://dergipark.gov.tr/gujsc

Detection of SSL/TLS Implementation Errors in Android Applications

Kaya Emre CİBALIK[1,*] Cemal KOÇAK[2]

[1]Gazi UniversityGraduate School Of Natural And Applied Sciences, Department of Information Security Engineering, 06500, Yenimahalle/ANKARA

[2]Gazi UniversityFaculty of Technology, Department of Computer Engineering, 06500, Yenimahalle/ANKARA

Article Info

Research article
Received:10/02/2021
Revision:24/03/2021
Accepted:05/05/2021

Keywords

Android
Application Security
SSL/TLS
Mobile Security

Abstract

Security Socket Layer (SSL) / Transport Layer Security (TLS) protocols are utilized to secure network communication (e.g., transmitting user data). Failing to properly implement SSL/TLS configuration during the app development results in security risks. The weak implementations include trusting all host names, trusting all certificates, ignoring certificate verification errors, even lack of SSL public key pinning usage. These unsecured implementations may cause Man-In-The-Middle (MITM) attacks. The major aim of this research is to detect configuration errors of SSL/TLS implementation in Android apps. It consists of the common use of existing open source tools in the static analysis phase and the combination of manual method in the dynamic analysis phase. During the static analysis phase, dynamic analysis of the findings obtained by scanning four types of vulnerabilities is used to verify the abuse status of SSL/TLS by testing. The dynamic analysis is essential for eliminating false positives generated at the static analysis stage. We analyze 109 apps from Google Play Store and the experimental results show that 45 (41.28%) apps contain potential security errors in the application of SSL/TLS. We verify that 19 (17.43%) out of 109 apps are vulnerable to MITM attacks.

1. INTRODUCTION

In recent years with the wide usage of smartphones, we have used many mobile applications in our daily life. Along with these developments, people start using smartphones instead of traditional desktop computers. The number of smartphone users worldwide today surpasses three billion and is forecast to further grow by several hundred million in the next few years [1]. Today, the leading smartphone vendors are Samsung, Apple, and Huawei. As of the second quarter of 2017, Android drives the worldwide market with an 87.9 percent piece of the overall industry, while Apple's iOS is second. Android is likewise the frequently utilized working framework for tablet PCs around the world, with a 66 percent portion of the worldwide market in 2016 [2]. The number of accessible applications in the Google Play Store was most as of late set at 2.7 million applications, in the wake of outperforming 1 million applications in July 2013. Google Play was initially dispatched in October 2008 under the name Android Market. As Google's authentic application store, it offers its clients a range of applications and advanced media, including music, magazines, books, film, and TV [3]. Most apps use SSL/TLS protocol to transmit sensitive and its authentication data.

The primary motivation behind SSL is to provide end-to-end security against active attackers and third parties. Regardless of whether the system is entirely compromised, SSL is utilized to ensure confidentiality, integrity, and authenticity for communications between the client and the server. Authenticate to the server is a crucial part of SSL/TLS connection establishment. This authentication occurs during the SSL handshaking phase, while the server is sending its public-key certificate. To ensure that encrypted traffic is secure, the client-side application or operator must verify that this certificate has been issued by a valid certificate authority, has not expired (not been revoked) and the name(s) included in certificate match(es) the name of the domain that the client is trying to connect, and other connection checks for security [4].

*Corresponding author, e-mail: ecibalik@gmail.com

DOI: 10.29109/gujsc.878053

There exist related work on inappropriate implementations of SSL/TLS in Android applications. A couple of studies [5-7], analyze issues with SSL warnings in their browsers but not inside of apps. These types of errors on SSL/TLS implementations cause MITM attacks on mobile environments. Fahl et al. [8], first raise this issue and developed an analysis tool called Mallodroid that detects vulnerabilities by using static code analysis. Then, the authors choose 100 of these vulnerable apps to perform dynamic analysis. The shortcoming of the methodology is that it requires manual confirmation of related vulnerabilities. Besides, they worked on an old version of Android OS. Sounthiraraj et al. [9] developed a tool called SMV-HUNTER that represented automated analysis of Android devices on this issue. They tested applications and libraries of SSL functionality on mobile devices running Android 2.3.6 and an iPad 2 running iOS 4.2.1 in the study. Yang et al. [10] performed only SSL/TLS error-handling vulnerability in hybrid web applications but did not deal with other codes that contain vulnerability of SSL/TLS verifications. Wang et al. [11] developed a static & dynamic analysis tool called DCDroid to detect SSL/TLS certificate verification vulnerabilities in Android apps. The shortcoming of the approach is that the authors performed dynamic analysis on Android 6.0 version but the publicly available latest Android version is Android 9.0 Pie in 2019.

In this study, we aim to detect configuration errors of SSL/TLS implementation in Android apps. We also checked whether the applications have certificate pinning in SSL/TLS validation mechanisms. We combine some open source tools for the static analysis phase and propose a model with the dynamic analysis phase. These apps cover a range from productivity, entertainment to public institutions. In summary, this paper makes the following contributions:

- We performed our study by using the Android 9.0 Pie version that is the latest version of Android released for major mobile devices in 2019. Due to using the latest version of Android OS, we planned to study on latest conditions of SSL/TLS implementations.
- We systematically studied vulnerabilities of SSL/TLS misconfigurations by combining some open source tools.
- We also checked whether or not SSL certificate pinning was used in applications during the static analysis phase.
- We choose apps according to ranks and categories.

The paper is organized as follows. Section 2, gives background information on how SSL is used in Android apps and how MITM attacks it can perform. In Section 3, we introduce the research statement and some challenges. In Section 4, we shared static and dynamic analysis processes and detailed information about them. In Section 5, we describe the data sets and give our experimental results. In Section 6, we discuss the limitations of our analysis. In Section 7, we conclude this paper and discuss the results.

2. BACKGROUND

In this section, we give the information of SSL/TLS in Android environment, Android UI and details of Man-in-the-middle attacks.

2.1. SSL/TLS InAndroid

SSL presently in fact known as TLS is a typical structure obstruct for encrypted communications among endpoints and servers. It is conceivable that an application may utilize SSL inaccurately to such an extent that malicious parties might have the option to block an application's information over the system. To enable you to guarantee this does not occur to your application, android applications have some features the basic pitfalls when utilizing secure system conventions and addresses some bigger worries about utilizing Public-Key Infrastructure (PKI) [12].

In a typical SSL usage scenario, a server is arranged with an endorsement containing an open key just as a coordinating private key. As a feature of the handshake between an SSL customer and server, the server demonstrates it has the private key by marking its authentication with open key cryptography. However, anybody can produce his or her own certificate and private key, so a simple handshake does not demonstrate anything about the server other than that the server realizes the private key that matches the open key of the endorsement. One approach to tackle this issue is to have the customer have at least one testament it trusts. There are a few drawbacks to this basic methodology. Servers ought to able to move up to more grounded keys after some time (key rotation), which replaces the open key in the authentication with another one. Lamentably, presently the customer application must be refreshed because of what is a server design change. This is dangerous if the server is not under the application engineer's control, for instance on the off chance that it is an outsider web administration. This methodology likewise has issues if the application needs to converse with self-assertive servers, for example, an internet browser or email application. To address these drawbacks, servers are ordinarily designed with testaments from understood guarantors called Certificate Authorities (CAs). The host stage mostly contains a rundown of surely understood CAs it trusts. As of Android 4.2 (Jelly Bean), Android at present contains over 100 CAs that are refreshed in each discharge.

Network Security Configuration was introduced on Android 7 and lets apps customize their network security settings such as custom trust anchors and Certificate pinning. At the point when applications target API Levels 24+(Android OS 7 and above) and are running on an Android device with versions 7+, they utilize a default Network Security Configuration that does not trust client provided CA's, decreasing the plausibility of MITM attacks by tricking clients to install malicious CA's [13].

With Android 9.0, they have made some changes and improvements to TLS validation mechanisms. RFC 2818 depicts two techniques to match a domain name against a certificate-utilizing the accessible names inside the Subject Alt Name (SAN) extension, or without a SAN extension, falling back to the Common Name (CN) [4]. Nonetheless, the fallback to the CN was deplored in RFC 2818. Therefore, Android never again falls back to utilizing the CN. To check a host-name, the server must present a certificate with a matching SAN. Declarations that do not contain a SAN matching the host-name are never again trusted [15].

After briefly explaining the working logic of SSL/TLS structure, we can talk about why certificate validation errors occur on devices using the Android operating system. Android OS has a self-defined SSL/TLS authentication and routing mechanism that does not invoke any vulnerability if properly used. However, there are many vulnerabilities because application developers do not properly configure their applications on related SSL/TLS libraries and classes [15]. To mention these situations;

- Trusting all certificates with X509TrustManager
- Improper check of domain name with Hostname Verifier
- Method for accepting all domain names (Allow_All_Hostname_Verifier)
- Ignoring SSL / TLS validation error when received with onReceivedSslerror()
- Lack of SSL Certificate Pinning.

2.2. Android User Interface And Navigation

Application's UI is everything that the client can see and cooperate with. Android gives an assortment of pre-fabricated UI parts, for example, organized format articles and UI controls that enable you to construct the graphical UI for your application. Android likewise gives other UI modules to unique interfaces, for example, exchanges, notices, and menus. An application can contain one or more Activity. These Activities are located in the AndroidManifest.xml file inside the application's apk file and start with Main Activity. It manages the view and windows. Therefore, in the dynamic analysis section, it is necessary to make audits with the help of the user interface to see which situations are causing the vulnerability.

2.3. Man-In-The-Middle-Attack (MITM)

MITM is a general term for when a perpetrator positions himself in a discussion between a client and an application-either to spy or to mimic one of the gatherings, causing it to show up as though an ordinary trade of data is in progress. There are many methods of attack in the intermediary and the means of attack used in these methods. However, if we examine the situation according to our SSL / TLS application, if the client cannot properly validate the certificate of the website or application to which it is linked, or if it contains some weaknesses in this approval mechanism, then the attackers may intervene and examine the traffic.

3. PROBLEM STATEMENT AND CHALLENGES

In this section, we introduce the major challenges in this work.

With Android 7.0, they have made some changes to the TLS validation and operation mechanism. With these changes, applications no longer rely on custom CAs added by the user or admin. Then, it is no longer possible to intervene with user-added CAs and monitor traffic during the dynamic analysis phase of applications. There are two ways to overcome this situation: it will both change the contents of the network_config.xml file and re-compile the application, or you will need to access the mobile device with root rights in the emulator and then import from Proxy’s Certificate to trusted root certificates. However, many applications will not work when you compile again.

Since we will test the weak codes, we found in the static analysis section; it is not always possible to all of the weak codes in the static analysis section. For this reason, we have chosen the most widely used open-source applications in the market during static analysis and compared them. These are Androbugs [16] and, MobSF [17]. We improve that we can compare the output of at least two different open source tools and achieve better results.

In Dynamic analysis, especially Turkish Public institutions, apps have contained Turkish characters. Therefore, we could not read text box and view output properly from related applications. For this reason, we had to do the tests of Turkish applications manually.

To reproduce automated testing, we first need to comprehend the UI components on the present screen and give the activity of the essential components, for example, content boxes need to info content, radio boxes need to check. Besides, after that, we select the interface components with acute need to snap as per the aftereffects of static examination. Existing tools are not compatible with UI automation, such as Monkeyrunner [18], whose execution does not have any purpose and not suitable for exact clicks, so it is hard to trigger intended code. Some other automation tools, such as FlowDroid [19] and DroidScope [20], can find method call relations, but cannot trigger dynamically potentially vulnerable codes.

4. ANALYSIS PHASE

In the model that is presented as Figure 1 we propose, applications are subjected to static analysis first. We used two different open source tools (Androbugs and MobSF) for static analysis. With the help of these tools, we analyzed the results and examined the results. From the results of static analysis, we determined what the weak codes are. More importantly, we have identified which type of vulnerability the application contains. For example, does the application include SSL/TLS validation vulnerability or does not pause the operation when the SSL/TLS error is received, allowing the connection. We tried to detect such outputs in the static analysis stage. Then in dynamic analysis, we tested the outputs from the static analysis at runtime with the help of ADB [21] and AndroidViewClient [22]. We tried to test the weak codes that we found in the static analysis step by using AndroidViewClient in the dynamic analysis step with Culebra [23], automation scripts. AndroidViewClient is a 100% pure python library and apparatuses that disentangles test content creation and android test automation, giving higher level tasks and the capacity of acquiring the tree of Views present at any given moment on the device or emulator screen and perform activities on it.

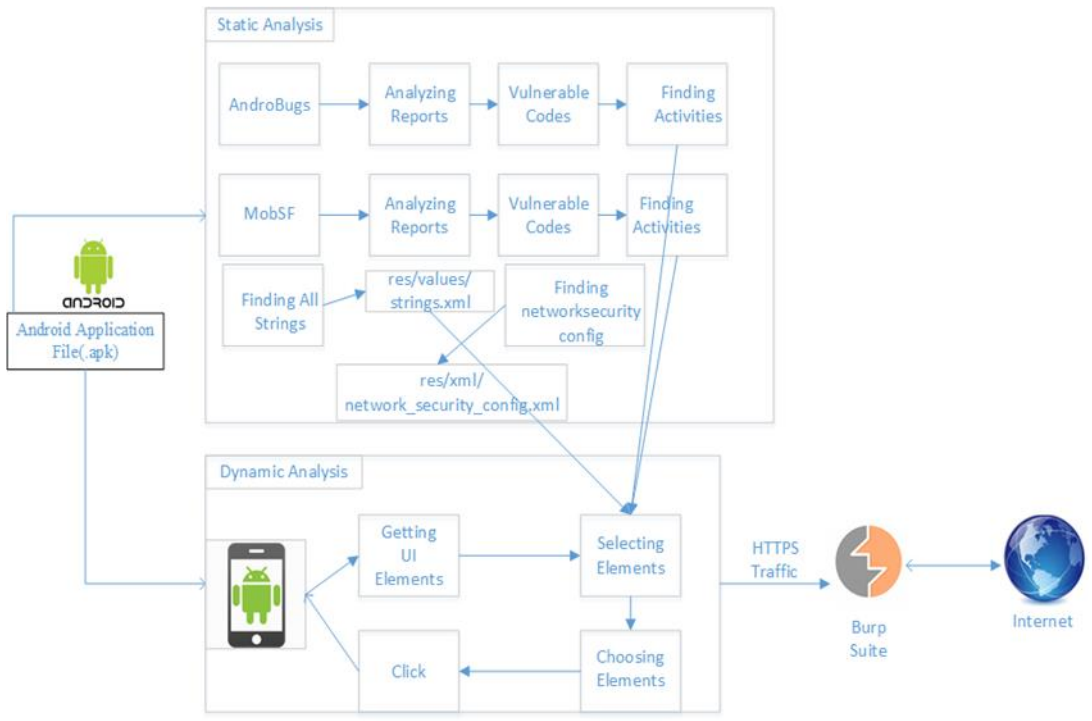

***Figure 1**. Testing model*

4.1. Static Analysis

In this section, we explained the steps of the static analysis.

4.1.1.TestingAppsBy Open-Source Tools

The process of static analysis starts with the process of decompile the application and convert it to source code. Decompiling process is mandatory whether using automated tools or using manual methods. In this work, we used two different open source tools, and these tools are used to automatically manage the decompile processes.

4.1.2. Source Code Analysis

SSL/TLS Implementation Checking: If it is not properly checked, It causes SSL/TLS verification error. In this topic, our tools checks whether the following methods exist and their results. ALLOW_ALL_HOSTNAME_VERIFIER or AllowAllHostnameVerifier, getInsecure() within SSLCertificateSocketFactory and SSLSessionCache methods.

SSL/TLS Certificate Verification Checking: If it is not properly checked, it may allow self-signed, expired or mismatch CN certificates for SSL/TLS connection. Those are vulnerable methods; "checkClientTrusted", "checkServerTrusted", "getAcceptedIssuers" methods within "X509Certificate" and functions with blank implementation.

SSL/TLS Certificate Pinning Checking: Certificate pinning is done by providing a set of certificates by hash of the public key (SubjectPublicKeyInfo of the X.509 certificate). A certificate chain is then valid only if the certificate chain contains at least one of the pinned public keys. If certificate pinning is set, the "res/xml/network_security_config.xml" file contains the hash of the public key of the certificate. When the pinning process is activated, the application will now be able to compare the hash of the public key of the server to which it is connected in SSL / TLS connections and the hash it has pin in the network_security_config.xml file to determine whether there is an attacker in the middle.

WebViewClientonreceveidSSLerror checking: First, the Webviewclient class is searched for the existence of the class, and then the onReceivedSslError method is searched from these classes. This method has two different instructions, one of which ends with proceed (), method in which the connection is resumed even if the SSL connection fails. Therefore, this method and the detection of the outputs reveal the existence of weakness.

4.2. Dynamic Analysis

In this section, we explained the steps of dynamic analysis.

4.2.1. Device Management

We used the Android Emulator virtual device from within Android Studio because it is difficult to find devices at every API level. We also thought that the emulator would make our job easier in case of problems during the tests. While the emulator environment crashed during the tests, we would quickly fix the problem with restart. We have connected emulator devices with ADB tools. Afterwards, we installed the applications to be tested and started our dynamic tests in the UI automation section.

4.2.2. User InterfaceAutomation

User Interface Automation is an important and necessary step for dynamic analysis. Because speeding up this analysis with the help of some scripts and making it more regular will help us get the results faster and help us to minimize user errors.

When an application runs, it is necessary to read all the elements on the screen and remove the corresponding attributes. These attributes are buttons, text boxes and other input areas. After the information is obtained this system should be able to run the appropriate events according to the elements collected so that Activity can jump from one to another. For example, we used AndroidViewClient to manage the components we would like to manage, elements to call from the User Interface, create appropriate events, and execute the application.

4.2.3. Setting Proxy

To simulate SSL / TLS MITM attack, it must intercept all traffic between client and server. At this stage, we used two different proxy tools. We used the Burp Suite Community Edition [24], the most widely used and most stable tool in the field, to verify certificate pinning conditions. With Android 7.0, the Android device does not trust the certificates installed by the user or admin. For this reason, it is not possible to intercept the SSL /TLS traffic without performing any certificate operation. For this reason, we must first install the self-signed certificate or another self-signed certificate of the proxy settings to the Android emulator using ADB tools. In the meantime, you need to upload the proxy's certificate to the trusted-roots certificates field. To do this, you need to have root rights on the device. With the help of ADB tools on the emulator devices, we access the root rights and import the certificate of the proxy. For other tests, we used the Mitmproxy tool [25]. The purpose of using this tool, unlike other proxy tools, Mitmproxy tool can inject fake certificates on the fly and do this after analyzing the remote server’s certificate. When we perform our tests importing no certificates, we can say that the applications where HTTPS traffic can be seen are vulnerable.

5. EXPERIMENTAL STUDY

In our experimental study, for static analysis a Kali [26], machine is installed in VMware workstation and Androbugs Framework and MobSF are installed. For dynamic analysis, we have one Windows 10 computer, Android Studio, Android Virtual Device Manager (AVD) and Android 8.0 Oreo, Android 9.0 Pie emulated device installed. In addition, Python 2.7 and ADB tool are installed to be run AndroidViewClient on this computer.

5.1. Dataset

Our dataset was selected on the original Android Play Store. When selecting the 109 applications, it was decided according to the download rates and popularity. These applications also include applications in public institutions in Turkey. The reason for this, as public institutions in Turkey to determine whether to take this type of application to take precautions against an attack and is to reveal the security features of the application. In addition, many categories of applications have been selected from productivity to business applications, entertainment to dating applications. In addition, applications over 80M are excluded from the scope due to the data size on emulator.

5.2. Static Analysis

During the static analysis process, we conducted tests using apk files of the applications with the help of two different open source tools. The results of the static analysis process are summarized in Table 1. These results show that 45 (41.28%) of 109 apps contain potentially vulnerable code.

Table 1.*Static Analysis Data*

	Google Play Store	
	Count	*Percentage*
Potential Vulnerable Apps	*45*	*41,28%*
Free from these type of vulnerabilities	*64*	*58,72%*
Total Apps	*109*	*100%*

Besides, 64 apps do not contain vulnerable codes that we have defined. These applications were removed in the emulator environment after the static analysis process and disk space was cleaned.

5.3. Dynamic Analysis

During the dynamic analysis phase, we installed our applications in the emulator environment. We run code that may contain potential weaknesses and examine the results. Dynamic analysis results were summarized in Table 2. These results show that 19 (42.22%) of 45 apps are vulnerable for SSL/TLS attacks.

Table 2*.Dynamic Analysis Data*

	Google Play Store	
	Count	*Percentage*
Vulnerability Confirmed	*19*	*42,22%*
Vulnerability Free	*26*	*57,78%*
Total Apps	*45*	*100%*

6. LIMITATION OF OUR ANALYSIS

Our proposed model performs automatic detection of SSL/TLS implementations and verifications errors and vulnerabilities, including the guidance of static and dynamic detection. However, there are still many limitations of our model.

In the static analysis phase, we used two different open source tools (MobSF, Androbugs). Using this type of tool has some advantages and some limitations. For example, in our study, the tools cannot control the Certificate Pinning condition. Therefore, we checked this condition manually from the source code. In other cases in static detection, we check source code of applications by decompiling with our open-source tools. We realized that in some conditions our open source tools classified applications as vulnerable by finding on source code that includes using the creating your X509Certificate class API instead of existing API. We also suggest existing API, otherwise, we cannot say that your apps are vulnerable.

Considering the overall study, the codes found in the static analysis were tested in dynamic analysis. There may also have been human errors because of our inability to automate this data transfer situation.

7. CONCLUSION

In this study, we propose a model to detect the configuration errors, vulnerabilities, and coding issues on SSL/TLS implementations. We analyzed 109 apps from Google Play Store. Our experimental results show that with static analysis 45(41.28%) apps contain potential vulnerabilities in the implementation and verification of SSL/TLS and with further dynamic analysis 19 (42.22%) of 45 apps are verified as vulnerable MITM attacks. We also used this model to analyze vulnerable apps, including categories, rank, and public institutions. In the future work, we are going to model to automate data transfer from static analysis codes to dynamic analysis environment and we are going to add an extension to be check the Certificate Pinning situation in the static analysis phase.

REFERENCES

[1] "Smartphone users 2020", (2020). Statista,https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[2] "Android versions market share 2019", (2020). Statista, https://www.statista.com/statistics/271774/share-of-android-platforms-on-mobile-devices-with-android-os/

[3] "Google Play Store: number of apps 2020", (2020). Statista, https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/

[4] Rescorla,E. "HTTP Over TLS". (2000). https://tools.ietf.org/html/rfc2818.

[5] Akhawe, D., & Felt, A. P., (2013). Alice in warningland: A large-scale field study of browser security warning effectiveness. In 22nd USENIX Security Symposium (USENIX Security 13), 257-272.

[6] Felt, A.P., Ainslie, A.,Reeder,R.W., Consolvo,S., Thyagaraja,S., Bettes,A., Harris,H., &Grimes,J. (2015). Improving SSL Warnings: Comprehension and Adherence. In Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems(pp. 2893–2902).

[7] Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., &Cranor, L. F. (2009). Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In USENIX security symposium (pp. 399-416).

[8] Fahl, S., Harbach, M., Muders, T., Baumgärtner, L., Freisleben, B., & Smith, M. (2012). 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 (pp. 50-61).

[9] Sounthiraraj, D., Sahs, J., Greenwood, G., Lin, Z., & Khan, L. (2014). Smv-hunter: Large scale, automated detection of ssl/tls man-in-the-middle vulnerabilities in android apps. In In Proceedings of the 21st Annual Network and Distributed System Security Symposium (NDSS'14.

[10] Liu, Y., Zuo, C., Zhang, Z., Guo, S., & Xu, X. (2018). An automatically vetting mechanism for SSL error-handling vulnerability in android hybrid Web apps. World Wide Web, 21(1), 127-150.

[11] Wang, Y., Liu, X., Mao, W., & Wang, W. (2019). Dcdroid: Automated detection of ssl/tls certificate verification vulnerabilities in android apps. In Proceedings of the ACM Turing Celebration Conference-China (pp. 1-9).

[12] Security with HTTPS and SSL. (n.d.). Android Developers, https://developer.android.com/training/articles/security-ssl

[13] Changes to Trusted Certificate Authorities in Android Nougat, Android Developers Blog, https://android-developers.googleblog.com/2016/07/changes-to-trusted-certificate.html.

[14] Behavior changes: All apps | Android Developers. (n.d.). https://developer.android.com/about/versions/pie/android-9.0-changes-all.

[15] Wei, X., & Wolf, M. (2017). A survey on HTTPS implementation by Android apps: issues and countermeasures. Applied Computing and Informatics, 13(2), 101-117.

[16] Lin, Y.-C., AndroBugs/AndroBugs_Framework. (2021). GitHub.

[17] "GitHub-MobSF/Mobile-Security-Framework-MobSF". https://github.com/MobSF/Mobile-Security-Framework-MobSF.

[18] "monkeyrunner". Android Developers. https://developer.android.com/studio/test/monkeyrunner.

[19] Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J.,& McDaniel, P. (2014). Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. AcmSigplan Notices, 49(6), 259-269.

[20] Yan, L. K., & Yin, H. (2012). Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis. In 21st USENIX Security Symposium (USENIX Security 12) (pp. 569-584).

[21] "Android Debug Bridge (adb)". Android Developers. https://developer.android.com/studio/command-line/adb.

[22] Milano, D. T., dtmilano/AndroidViewClient. (2021). https://github.com/dtmilano/AndroidViewClient

[23] Milano, D. T., (2021). https://github.com/dtmilano/AndroidViewClient/wiki/culebra.

[24] "Burp Suite - Application Security Testing Software".https://portswigger.net/burp.

[25] "mitmproxy - an interactive HTTPS proxy". https://mitmproxy.org/.

[26] "Offensive Security Introduces Kali Linux". https://www.kali.org/offensive-security-introduces-kali-linux/.