Network Security Challenges in Android Applications

***Abstract*—The digital world is in constant battle for improvement - especially in the security field. Taking in considerations the revelations from Edward Snowden about the mass surveillance programs conducted by the governmental authorities, the number of users that raised awareness is constantly increasing. More and more users agree that additional steps must be taken to ensure the fact that communications will remain private as intended in the first place. Taking in consideration the ongoing transition in the digital world, there are already more mobile phones than people on this planet. According to [19] there are around 7 billion active cell phones by 2014 out of which nearly 2 billion are smartphones. Simply, the use of smartphones could open a great security hole. The most common problem when it comes to Android applications is the common misuse of the HTTPS protocol. Having this in mind, this paper addresses the current issues when it comes to misuse of the HTTPS protocol and proposes possible solutions to overcome this common problem. In this paper we evaluate the SSL implementation in a recent set of Android applications and present some of the most common missuses. The goal of this paper is to raise awareness to current and new developers to actually consider the security as one of their main goals during the development life cycle of applications.**

***Keywords*-Android; Android security; SSL; smartphones; network security;**

I. Introduction

Nowadays, the more frequent use of smartphones raises a discussion about the actual security level that is offered to the users. The use of smartphones becomes a part of every ones daily routines with all those services offered. Accordingly, the network utilization notices drastic changes. A vast majority of users are accessing the Internet via smartphones and tablets. Application markets such as the official Google Play Store[1] offer the users different applications with a broad spectrum of functionalities. A large part of the applications available in the Google Play Store require access to the Internet. The most common way for achieving this is by making use of the HTTP and HTTPS protocols. In this paper we analyze a subset of 3K applications chosen from the pool of the most recent Android applications from 2014 regarding the correct implementation of the HTTPS protocol. Although the misuse of the HTTPS is known issue and there are already some publicly available solutions for this particular problem [16], [20], developers tend to trade the security for the design and usability of applications. Such security holes render the user an easy target for attackers, which could easily lead to stealing of sensitive information or act as an entry point for more sophisticated attacks. We discovered that a large number of the application present in the Android market have a broken implementation of the HTTPS protocol. Moreover it was staggering to discover that some of these applications actually provide banking services. Furthermore we found applications that are not transferring the data over HTTPS, instead they use HTTP for data transfer. This indicated that user credentials such as usernames and passwords are sent in plain text and the consequences from this are more than obvious. Therefore we value the results from this paper as a base for our future work aimed at dynamic on-device analysis for Android applications. This work could significantly enhance the overall security of the applications presented by its capability to dynamically detect and replace insecure libraries with their secure equivalent. Our analysis confirmed that wrong use of SSL is still a problem that is present in Android applications. A summary of the obtained results can be found below:

- `22.85%` of the tested applications contain broken Trust Managers
- `8.9%` contain partially broken Trust Manager
- `14.16%` contain broken Hostname Verifier
- `3.3%` contain partially broken Hostname Verifier
- `0.07%` contain broken SSL Errors
- `29.75%` contain partially broken SSL Errors

The paper is organized as following: Section 2 provides the Background information on Android security concepts along with the related work. In Section 3 an overview of the Android SSL implementation is presented followed by the discussion of the currently available analysis methods in Section 4. Section 5 presents the middleware frameworks that can be used for dynamic interception and replacement of code. Finally, the results are presented in Section 6 followed by conclusion and planned future work.

II. Background and Related Work

In this section we provide a brief overview of the security concepts used in Android. The goal of this section is to provide the reader with the theoretical fundamentals regarding the security concepts used in Android applications. These concepts aim to provide:

- Assurance that personal user data will remain private
- Keeping particular system resources protected
- Limited environment for applications to execute

In order to achieve the previously stated goals, the Android operating system provides different levels of security, which can be classified as:

- Kernel security
- Using sandboxing techniques to enforce separate execution environments for different applications
- Providing secure communication between processes
- Mandatory signing requirement for every application

[1]Google Play Store, play.google.com

- The permission model

Just like every other widespread commercial product, Android itself has been attracting a lot attention from researchers in the field of security. To this day, different security aspects of the Android security model have been thoroughly researched, contributing to the discovery of critical vulnerabilities [36]. Most of the research is aimed at the coarse permission model, the general aspects of Android security, over-privileged applications and detection of malware. We refer the interested reader to Enck et al. [24], Vidas et al. [23] and Tabassum et al. [25] for further information and a good overview of the overall security model as well as the attack vectors in Android. Shabtai et al. [26] propose a number of security solutions for threats to the Android platform that were previously identified.

A. Application Sandboxing

This approach to system hardening provides every application with its own identification number (ID) and limits the environment in which certain code can be executed. The goal behind this idea is to improve the security by isolating the application in order to prevent outside malware, intruders, system resources and other applications from interfering with the protected application. However, Davi et al. [27] presents a privilege escalation attack performed during runtime that shows the ineffectiveness of the sandboxing feature.

B. Secure IPC

The secure interprocess communication is achieved via the Binder, which is a remote procedure call mechanism responsible for transferring the in-process and cross-process calls from i.e. Intents and Content Providers. Being the lowest level of communication that transfers information to the kernel, Tam et al. [31] propose CopperDroid[2], a novel analysis framework that leverages these low level calls for reconstruction of the application behavior in order to detect certain vulnerabilities.

C. Application-Defined and User Granted Permissions

Android uses a mandatory permission model. When an application wants to use certain services, this must be clearly stated in the manifest file. This means that upon installation the user will be notified which requirements are necessary for that particular application. Regarding HTTPS, Android does not have a separate permission that clearly specifies the use of this protocol. Instead everything is grouped into one global permission that allows access to the Internet[3]. Dhama et al. [34] give a good overview of the security challenges and general use of the permissions used in Android Applications. Furthermore there has been much effort in researching the permission model and over-privileged applications that could lead to significant privacy issues and data theft (cmp. [28], [29], [30]). We will not argue whether this permission approach could be improved because we have to take in consideration the mental model of the people, who in most of the cases do not pay attention to the permission warnings. Even if the users pay attention to these warnings it is arguable whether non-technophile users are sufficiently familiar with the presented terms, or the resulting consequences.

[2]CopperDroid Online Sandbox - http://copperdroid.isg.rhul.ac.uk/copperdroid/

[3]Internet Permission - `<uses-permission Android:name="Android.permission.INTERNET"/>`

III. Overview of Android SSL

Regarding the fact that HTTPS is the only meaningful protection mechanism for Internet communication in Android and taking into consideration the fact that the number of applications that require access to the Internet is constantly rising, in this paper we will evaluate the current state of HTTPS implementations in Android applications.

A. HTTPS and SSL/TLS

HTTP over SSL/TLS, or more commonly known as HTTPS, is a data transmission protocol which transfers normal HTTP traffic over SSL[4] or TLS[5]. In this paper we will not discuss the weaknesses of SSL/TLS, but focus on the implementation of this protocol in Android applications instead. The goal of this protocol is to provide protection against eavesdropping on the connections. The most common and widely known attack scheme against this is the man-in-the-middle attack. This attack is supposed to intercept, modify, block and/or redirect the traffic. There are several known approaches that eliminate the possibility of this attack. The most common approach is by using X.509 Certificates. This means that the host, which in our case is the application and the server that the application is communicating with, are mutually authenticated with the use of certificates. In most of the client server setups, the server obtains a X.509 certificate containing its public key and it is signed by certain known and trusted Certificate Authority (CA). In order for a communication to start, the server's certificate is then sent to the client when the client is trying to establish a communication. During the time of this exchange of the certificate, there is still an opportunity for an attacker to perform a man-in-the-middle attack. However, there are certain techniques explained in the following sections that are intended to prevent this from happening. Furthermore, the most common use of certificates can be divided as:

- Form of identification
- Public key used for encryption of data

Basically the overall goal of HTTPS is to bind the communication between the legitimate server and host. An HTTPS client checks the validity of the parameters presented in the certificate, like the common name. If some of the parameters do not match a warning is displayed. In order for this check to succeed, the Android operating system comes with preloaded root certificates from trusted vendors. According to [35], the most common trusted certificate authorities to be found are:

- Comodo SSL with `33.6%` market share

[4]Secure Sockets Layer

[5]Transport Layer Security

- Symantec (who owns VeriSign, Thawte and GeoTrust) with 33.2% market share.
- Go Daddy with 13.2% market share
- GlobalSign with 11.3% market share
- DigiCert with 2.9% market share

B. SSL implementation in Android

The open approach that Google has towards Android developers enables flexibility when it comes to implementation of certain functionalities. This enables implementation of advanced custom security concepts but also results in significant security challenges. The Android SDK provides the developers with several opportunities for implementation of the networking part of the application. This includes use of javax.net, java.net, org.apache.http and Android.net packages. However, the actual implementation is left to the developer. This means that developers should ensure proper implementation of these packages in order to achieve secure transport over the network. Fahl et al. [21] identify and classify the common missuses of SSL as:

- Trusting all Certificates
- Allowing all Hostnames
- Trusting many CAs
- Mixed mode or No SSL implementation.

All of the specified misuses are usually located in the checkServerTrusted [6] function that is actually responsible for implementation and validation of the certificates. Trusting all Certificates is the most common mistake that is implemented. This means that the TrustManager interface is set to accept all of the certificates without any check. This is achieved by overriding the interface to return null, which leads to the fact that the certificates are completely ignored. Furthermore, the hostname verification is the second most common mistake to be found. This means that there has to be a check that will determine whether the certificate is issued for the particular address that the application is trying to connect to. In other words, if an application is trying to establish communication to url: www.Android.com a certificate issued for any other domain must not be accepted and the communication has to be terminated. Although this issue is commonly found under the first category also, still there are cases where just the hostname verification is misused beside the fact that there are some certificate checks implemented. We argue that the mixed mode implementation is directly an SSL problem since there are many developers that tend to mix secure with insecure communication. Although not directly affected, the lack of indicators for secure communication such as the small lock found in the browsers renders the SSL implementation in Android with limited visibility and makes it a far more easy target to SSL stripping attacks as presented in [33]. In general, the wrong use of HTTPS is still a big issue. The next chapter will give an overview of the analysis methods used to detect these problems in applications.

IV. Analysis methods

To this day there are different techniques that are used for analysis of Android applications. The most common way to achieve this is through code analysis also known as static analysis [18] and dynamic [12] or behavioral analysis. Regarding the fact that all applications are packaged, to perform static analysis the use of additional tools such as apktool, dex2jar and jd-gui [11] is required. On the other hand dynamic analysis is performed in a manner that the application is executed in its own environment while its behavior is tracked. A good comparison of the currently available online sandboxes for dynamic instrumentation is presented by Neuner et al. [32]. However, the two approaches stated above have certain drawbacks. First in order to perform these analyses we have to obtain an actual apk file for the application, which is not a problem for a small set of applications but for a larger set of applications it can be difficult. Therefore we aim for a concept called on-device analysis, which eliminates the need of retrieving the actual apk file from the device in the first place. Regarding the fact that modern analysis tools are separately installed on machines where the analyses are performed, we focus on analysis tools that could be installed and performed on the device. Upon extensive research we identified four frameworks that can be used as base for our on-device analysis concept. These frameworks serve as a base for development of certain modules that can be used for different purposes. These frameworks are usually used to make custom enhancements to the Android operating system, like changes to the graphical user interface (GUI). Furthermore we identified the use of Cydia Substrate to bypass security features such as certificate pinning. In the following section we describe the frameworks and their functionality.

V. Frameworks for Dynamic Instrumentation

Having all this in mind, our goal is to find a way to detect these anomalies and repair them automatically. This means that everything has to be done on the phone and in the background, eliminating the need for user interaction since a vast majority of the Android users does not actually have any technical background. We have discovered that the only way to achieve this is to find a way to intercept certain functions and/or libraries and check their result. This implies that everything has to be done during runtime. Therefore we need a framework that can be used for dynamic instrumentation of Android applications. For our use case, the framework needs to provide functionalities for interception and injection of code during execution. We have identified and examined the following 4 frameworks:

- Cydia Substrate [7]
- Xposed Framework [8]
- PIN for Android [9]
- DDI Dynamic Dalvik Instrumentation for Android [10]

All of the aforementioned frameworks have much in common. The essential requirement is root access to the phone since all of the frameworks need access to the

app_process executable, which is the heart of the Android system. The technical details of the frameworks include modification, or more specific, extending the app_process executable to load a JAR file on startup. The classes of the loaded file are implemented in every process including the system services and according to this are able to act with their powers. Their power is demonstrated via the hooking functionality that allows the developer to hook, intercept and even modify code during execution. Therefore we render these frameworks as promising candidates that could be used in on-device analysis. Furthermore, we see potential use of these frameworks for manipulation of libraries. The fact that they need root access in order to be able to operate is an promising indication that certain unstable or insecure system libraries could be detected and replaced with more stable and secure versions. To this day we have not noticed the use of these frameworks for security purposes. Therefore, we plan to evaluate their functionality in deeper detail and decide to what extent these frameworks can be used to enhance system security. Table 1 classifies the features of each of the frameworks.

VI. Case Study

We conducted static analysis on a set of 3K applications from Google Play Store[14] regarding the SSL implementation. We used the malloDroid[15] script to detect the number of applications with a broken SSL implementation. We found out that there is still a large number of applications that require the Internet permission and should have used SSL for data transmission, but instead the overridden trust manager was set to return null, eliminating the use of SSL in the first place and thereby accepting all of the certificates. Furthermore, just observing the names of the classes that override the default trust manager leads us to the fact that SSL is sometimes absolutely misused. We have encountered classes such as *FakeTrustManager*, *FakeSSLTrustManager*, *AcceptAllTrustManager* etc. The implementation of these classes was following the pattern given in Listing 1 and Listing 2.

The results from the 3842 tested applications are listed in the Table II. A broken TrustManager was found in 848 application which is around 22% of the whole set of applications. Furthermore we discovered that 544 applications have a bad implementation of the HostnameVerifier. This brings us to the fact that use of these applications could easily lead to data theft. Taking the number of tested applications into consideration, in accordance with the total number of applications present in the Google Play Store, we believe that the real number of application that have SSL misuse problems is much higher.

It is more than obvious that there is still a major problem present in Android regarding secure network communication. Apart from the fact that many techniques such as certificate pinning and forced use of HTTPS exist, developers still tend to ignore the important role that SSL has in Android. Fahl et al. [21] showed the

```
public class TrustManager implements
    javax.net.ssl.X509TrustManager{
  public TrustManager(){
    return;
    }
  public void
     checkClientTrusted(java.security.cert.
  X509Certificate[]p1, String p2){
    return;
    }
  public void
     checkServerTrusted(java.security.cert.
  X509Certificate[]p1, String p2){
    return;
    }
  public
     java.security.cert.X509Certificate[]
     getAcceptedIssuers(){
    return 0;
    }
}
```

Listing 1. Template of a FakeTrustManager

```
class NetworkManager$1 implements
    javax.net.ssl.HostnameVerifier {

   NetworkManager$1()
   {
      return;
   }

   public boolean verify(String p2,
       javax.net.ssl.SSLSession p3)
   {
      return 1;
   }
}
```

Listing 2. Template of a broken Hostname Verification

state of 13500 applications chosen back in 2012th and compared to those results we can conclude that there is not much improvement regarding the use of SSL in Android applications.

VII. Conclusion

The results presented in this paper brought us the real picture regarding the state of HTTPS usage for network communication in the analysed subset of the most downloaded 3K Android applications from 2014. We think that this problem is a result of multiple drawbacks from different aspects among which we render the lack of knowledge, both from developers and users, as one of the main reasons for this problem. Taking the digitalization of the world around us into consideration, like the use of services that provide online banking etc., it is essential to fill the gap in security produced by the misuse of the SSL protocol. Keeping sensitive data private should be the main goal of anyone developing applications intended for smartphone use.

	Java Code	Native Code	Class Loading	Method Calls Interception	Security
Xposed	✔	✔	✔	✔	
Cydia Substrate	✔	✔	✔	✔	✔
PIN	✔	✔			
DDI	✔	✔	✔		

Table I
CHARACTERISTICS OF DYNAMIC INSTRUMENTATION FRAMEWORKS FOR ANDROID

	Broken	Possibly Broken
Trust Manager	848	342
Hostname Verifier	544	127
SSL Error	3	1143

Table II
RESULTS

VIII. FUTURE WORK

Our future work is aimed at evaluation of the aforementioned frameworks in order to distinguish to what extent these middleware interfaces can be used for improving the overall security in Android. It is clear that this approach has potential use in improving the security of Android applications on a global scale regarding the fact that it could be later on embedded into the official version of the Android operating system. We think that this approach, if adapted adequately to specific needs, can lead to elimination of certain problems such as introduction of separate permissions for services identified as a potential point for information leaks.

ACKNOWLEDGMENT

The authors would like to thank... more thanks here

REFERENCES

[1] Statistical information for Google Play Store, http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, February 2015

[2] Android Permission Model, http://developer.Android.com/guide/topics/security/permissions.html

[3] Stack Overflow, http://stackoverflow.com/

[4] Telegram Messenger, https://www.telegram.org/

[5] WhatsApp, https://www.whatsapp.com/

[6] X.509 TrustManager, http://developer.Android.com/reference/javax/net/ssl/X509TrustManager.html

[7] Cydia Substrate, http://www.cydiasubstrate.com/

[8] Xposed Framework, http://repo.xposed.info/module/de.robv.Android.xposed.installer

[9] PIN for Android, https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool

[10] Dynamic Dalvik Instrumentation toolkit, https://github.com/crmulliner/ddi

[11] Tools for decompiling Android applications, http://forum.xda-developers.com/showthread.php?t=1755243

[12] Andrubis: A tool for Analyzing Unknown Android Applications, http://blog.iseclab.org/2012/06/04/andrubis-a-tool-for-analyzing-unknown-Android-applications-2/

[13] Cydia Substrate SDK - http://www.cydiasubstrate.com/id/73e45fe5-4525-4de7-ac14-6016652cc1b8/

[14] Google Play Store, https://play.google.com/store

[15] MalloDroid, https://github.com/sfahl/mallodroid

[16] OWASP, https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

[17] William Klieber, Lori Flynn, Amar Bhosale, Limin Jia, and Lujo Bauer. 2014. Android taint flow analysis for app sets. In Proceedings of the 3rd ACM SIGPLAN International Workshop on the State of the Art in Java Program Analysis (SOAP '14). ACM, New York, NY, USA, 1-6. DOI=10.1145/2614628.2614633 http://doi.acm.org/10.1145/2614628.2614633

[18] tienne Payet and Fausto Spoto. 2011. Static analysis of Android programs. In Proceedings of the 23rd international conference on Automated deduction (CADE'11), Nikolaj Bjrner and Viorica Sofronie-Stokkermans (Eds.). Springer-Verlag, Berlin, Heidelberg, 439-445.

[19] Mobile Phone Users, http://www.digitaltrends.com/mobile/mobile-phone-world-population-2014/

[20] Your app shouldnt suffer SSLs problems, Moxie Marlinspike, http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/

[21] Sascha Fahl, Marian Harbach, Thomas Muders, Lars Baumgrtner, Bernd Freisleben, and Matthew Smith. 2012. Why eve and mallory love Android: an analysis of Android SSL (in)security. InProceedings of the 2012 ACM conference on Computer and communications security (CCS '12). ACM, New York, NY, USA, 50-61. DOI=10.1145/2382196.2382205 http://doi.acm.org/10.1145/2382196.2382205

[22] Hubbard, J.; Weimer, K.; Yu Chen, "A study of SSL Proxy attacks on Android and iOS mobile applications," Consumer Communications and Networking Conference (CCNC), 2014 IEEE 11th , vol., no., pp.86,91, 10-13 Jan. 2014,doi: 10.1109/CCNC.2014.6866553, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6866553&isnumber=6866537

[23] Timothy Vidas, Daniel Votipka, and Nicolas Christin. 2011. All your droid are belong to us: a survey of current Android attacks. In Proceedings of the 5th USENIX conference on Offensive technologies (WOOT'11). USENIX Association, Berkeley, CA, USA, 10-10.

[24] William Enck, Damien Octeau, Patrick McDaniel, and Swarat Chaudhuri. 2011. A study of Android application security. In Proceedings of the 20th USENIX conference on Security (SEC'11). USENIX Association, Berkeley, CA, USA, 21-21.

[25] Tabassum, G., Pandit, S., Ghosh, N. (2014, December). Android Application Security. In Journal of Emerging Technologies and Innovative Research (Vol. 1, No. 7 (December-2014)). JETIR.

[26] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, Shlomi Dolev, and Chanan Glezer. 2010. Google Android: A Comprehensive Security Assessment. IEEE Security and Privacy 8, 2 (March 2010), 35-44. DOI=10.1109/MSP.2010.2 http://dx.doi.org/10.1109/MSP.2010.2

[27] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. 2010. Privilege escalation attacks on Android. In Proceedings of the 13th international conference on Information security (ISC'10), Mike Burmester, Gene Tsudik, Spyros Magliveras, and Ivana Ili (Eds.). Springer-Verlag, Berlin, Heidelberg, 346-360.

[28] Bugiel, S.; Davi, L.; Dmitrienko, A.; Fischer, T.; Sadeghi, A.-R. Shastry, B. (2012), Towards Taming Privilege-Escalation Attacks on Android., in 'NDSS' , The Internet Society

[29] Georgios Portokalidis, Philip Homburg, Kostas Anagnostakis, and Herbert Bos. 2010. Paranoid Android: versatile protection for smartphones. In Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC '10). ACM, New York, NY, USA, 347-356. DOI=10.1145/1920261.1920313 http://doi.acm.org/10.1145/1920261.1920313

[30] William Enck, Machigar Ongtang and Patrick McDaniel. 2009. On lightweight mobile phone application certification. In Proceedings of the 16th ACM conference on Computer and communications security (CCS '09). ACM, New York, NY, USA, 235-245. DOI=10.1145/1653662.1653691 http://doi.acm.org/10.1145/1653662.1653691

[31] CopperDroid: Automatic Reconstruction of Android Malware Behaviors, Kimberly Tam, Salahuddin J. Khan, Aristide Fattori, and Lorenzo Cavallaro 22nd Annual Network and Distributed System Security Symposium, NDSS 2015 San Diego, California, USA, February 8-11, 2015

[32] Sebastian Neuner and Victor Van der Veen and Martina Lindorfer and Markus Huber and Georg Merzdovnik and Martin Mulazzani and Edgar R. Weippl, "Enter Sandbox: Android Sandbox Comparison," in Proceedings of the IEEE Mobile Security Technologies Workshop (MoST), 2014

[33] Marlinspike M. More Tricks For Defeating SSL In Practice. In Black Hat USA, 2009.

[34] S. Dhama, An Overview of Security Challenges of Android Apps Permissions, International Journal of Information and Computation Technology. ISSN 0974-2239 Volume 4, Number 4 (2014), pp. 373-380 International Research Publications House http://www.irphouse.com/ijict.htm

[35] Certificate authorities, Wikipedia, http://en.wikipedia.org/wiki/Certificate_authority

[36] Tarun Mall, Samarth Gupta, Critical Evaluation of Security Framework in Android Applications: Android-level security and Application-level security, International Research Journal of Computers and Electronics Engineering (IRJCEE) Vol. 2, Iss. 1, DEC 2014 IRJCEE-120914-TM-2014-I-1