2016 International Conference on Wireless Communication and Network Engineering (WCNE 2016)
ISBN: 978-1-60595-403-5

An Empirical Study of Insecure Communication in Android Apps

Yue-heng ZHANG*, Jun-liang SHU, Juan-ru LI, Qing WANG, and Da-wu GU

Computer Science and Engineering Department, Shanghai Jiao Tong University, Shanghai, China

*Corresponding author

Keywords: Android apps, Proprietary protocol, Security, Program analysis.

Abstract. Android apps rely on secure communication protocol to prove the confidentiality of sensitive data transmission. However, inexperienced developers tend to adopt insecure communication and introduce security risks. To study how prevalent the insecure communication protocols are used by real world Android apps, we conducted an in-depth analysis to examine popular apps from Google Play and MyApp Android app market. We collect 435 apps from major categories, such as gaming, shopping and social networks, and we monitored the communication of those apps and classified their used protocols into three categories: secure, insecure, and proprietary. Then we investigated those proprietary ones to find potential insecure implementation. We designed and implemented RawDroid, a protocol audit system combining network monitoring and program analysis technique to systematically inspect the security of proprietary protocol. We found that a large number of developers frequently use non-standard proprietary protocols. Among all analyzed apps, our security audit revealed that 36.7% apps adopted a proprietary protocol, and all those proprietary protocols fail to achieve confidentiality: some of them send sensitive data in the form of plaintext to servers; some misuse cryptographic algorithms and lead to the exposure of transferred privacy even if the content is encrypted. We believe this kind of protocols pose great security threats to Android ecosystem.

Introduction

The confidentiality of transmitted privacy data heavily relies on the security of protocols. In general, developers must build robust protocols, which guarantee apps to communicate with their servers securely against attackers. To protect from being attacked through untrusted network, security best practices suggest deploying HTTPS protocol. Nonetheless, many apps still adopt security-by-obscurity policy to deploy their own proprietary protocols. Therefore, security risks are introduced.

To our knowledge, although the security analysis of common application layer protocols has been widely studied in previous works. There is no systematic research on analyzing the prevalence of Android apps' insecure communication, especially on proprietary protocols. Most of previous works[1,2] rely on the fact that the format of analyzed protocol is well defined in network traffic. Unlike common protocols, however, the formats of proprietary protocols vary from one to another, which makes it difficult to be analyzed. In addition, many proprietary protocols employ encryption to hinder the analysis. Hence new analysis should be proposed to address these issues.

In this paper, we conducted a systematically study to investigate the situation of insecure network protocol usage in Android apps. We fulfilled the study in two steps. The first step of our work involves an automated testing to find the use of potential insecure communication (non-HTTPS of those apps. Then, we further analyze used proprietary protocols to audit the security. We propose RawDroid, a lightweight and semi-automatic protocol audit system to detect security flaws of proprietary protocols. RawDroid combines network traffic analysis and dynamic program analysis to locate potential insecure protocols. It hooks all key functions in system libraries related to network to record the invoking and the parameters of them. RawDroid also captures the network traffic and system status for later analysis during the execution. With the help of these run-time information, RawDroid determines whether an Android app uses proprietary protocol, and enhances manual analysis through pinpointing related functions in code to efficiently evaluate the security of the used protocols.

To validate RawDroid, we selected 435 apps involving entertaining, online shopping, and social network communicating from Google Play and MyApp Android app market. The analysis of RawDroid found that insecure communication (non HTTPS) are used by 363 apps (83.4%). Among them, 158 apps (36.7%) use proprietary protocols rather than common application layer protocols (e.g., HTTP, HTTPS) to transmit data. And we manually audited 60 apps using proprietary protocols with the help of RawDroid, finding that all proprietary protocols are insecure. We found 54 apps send sensitive data in plaintext, while six apps misuse cryptography to build insecure encryption schemes. The cases of cryptographic misuses include hard-coded key in their custom cryptographic algorithms, key transmitting in plaintext with encrypted data. The experimental results show that the potential threats are severe and can lead to serious attacks, leaking user's privacy. The results also raise the alarm to developers about the importance of applying security protocols instead of designing proprietary ones.

In this paper, we make the following contributions:

- We conduct an in-depth analysis on insecure network communications of Android apps, which provide a panoramic view of the usage of insecure communication in Android apps.
- We present RawDroid as a protocol audit system to analyze the security of data transmission through proprietary protocols in Android and use it to perform a large scale evaluation of proprietary protocols.
- Our study demonstrates that transferring data through proprietary protocols is common in modern Android apps. However, most of them fail to protect the data due to security flaws.

Analysis of Insecure Communication

How network communications guarantee the security of apps in Android is a common question to security researchers. In order to investigate this question, we study real world Android apps to address this question. Given an app, our preliminary step takes an off-the-shelf installed app as input, and launches it relying on Google Monkey [3], a popular tool for automatically executing and simulating user interaction to trigger various events of apps. After that, we collect information about the network usage of the app. The command *netstat* is used to monitor all kinds of network information, such as network usage, interface statistics, route table and so on. We use this command to record each server's IP address and port numbers during Android runtime. However, it is limited to only preform *netstat*. In order to record the app's individual network usage completely, we also monitor these four system files: */proc/net/tcp(6)* and */proc/net/udp(6)*. All of the four files are readable with any permissions. These four files are used to record all connections status for TCP or UDP. We use app's uid to filter each network trace. For the future analysis, we also capture the network traffic for each app.

In this work, we built a dataset of apps. We select 435 apps which were randomly downloaded from Google Play and MyApp Android Market, and each of them had been downloaded at least millions of times. There were three apps that had no ability to communicate with the network, and two apps crashed by some reasons. We finally successfully analyzed 430 apps. The results of our analysis are listed in Table 1.

Table 1. The Types of Protocols Used by Sample Apps.

Type	The Number of Apps
HTTP	203
HTTPS	69
Proprietary Protocol	158
Total	435

The results demonstrate that commonly used protocols in network communication fall into three categories—HTTP, HTTPS, Proprietary Protocol. According to the result, only 69 (15.9%) apps use

HTTPS as their transmission method. 158 (36.7%) of the apps use other protocols to transport sensitive data to servers. 203 apps utilize HTTP as their primary way of network communication. The fact clearly shows that users' sensitive data are insecure and easily expose to attackers.

It is easy to know that apps will use both HTTP and HTTPS. We consider that this situation is insecure. In this paper, we define that only HTTPS is secure protocol and HTTP and proprietary protocol are insecure. The definition of secure and insecure protocols is not arbitrary, and it is necessary and realistic for our study. First of all, all the sending data are plaintext which are encapsulated into HTTP. It is easy to be MITM attacked. For example, in an untrust WiFi channel, a game app only uses HTTP to transmit data, attackers can tamper with the runtime data. If the packages contain any payment information, it may fool users and make game companies profit loss. For another instance, an app GET a picture from advertisement server by HTTP, attackers can tamper with network traffic and load false advertising to cheat users on purpose.

We also summarize the port numbers used in different protocols. The connections of HTTP exploit the port numbers ranged from 8000 to 8080 and port 80. While, major apps use the 43, 443, 8020 to implement HTTPS. There are no particular rules on port numbers, since, it depends on developers' hobbies.

Analysis of Proprietary Protocol

In order to analyze the vulnerability of the usage of insecure communications and assess their prevalence in existing apps, we implement RawDroid to scalable and accurately examine apps from Google Play and MyApp Android markets. It consists of Inline Hook to record necessary runtime information, and analysis of potential weakness. In Figure 1, we show the overall workflow of the system, which involves several steps to determine potential threats in Android apps. Given a target app to analyze, the first step is to dynamic determine whether apps use insecure communications, and if so, we capture data packets and network usage status in the meantime. In the second step, we select data packets of all candidate apps, and use traffic analysis module to detect the apps whether contains sensitive data in plaintext. After this step, we classify apps which use encrypted proprietary protocols and re-run them to record method traces of network API and specified parameters. Finally, in order to determine whether these connections are indeed vulnerable, we combine program analysis technique to detect and manually verify the security of proprietary protocols.

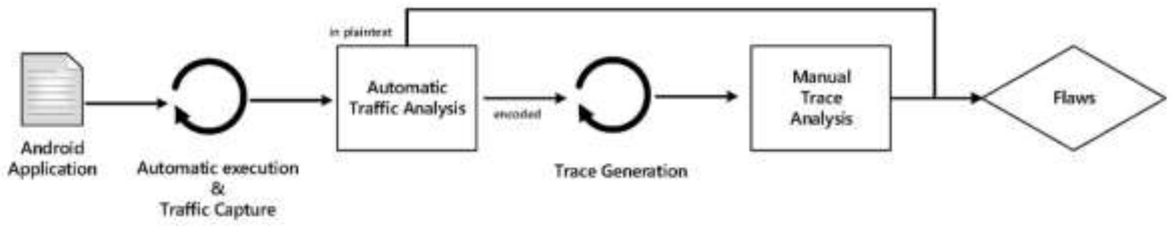

Figure 1. Analyze Apps with RawDroid.

Evaluation

We evaluate our systems ability and efficiently identify suspicious connections on a large set of apps from Google Play and MyApp Android market, including entertaining, online shopping, and social network communicating. We use LG Nexus 4 running Android 4.4.3 to evaluate RawDroid. We also root this phone, because RawDroid deploy *Xposed* and *Adbi* to record runtime information, which requires root assessment. In order to verify the results and draw a conclusion, for our study, we randomly select 60 apps from the proprietary protocol category. Table 2 presents the results of our analysis. We take the network traffic of each app as input, and find that most (54) of them only use TCP/UDP to transmit plain text with on deeper protection. More in-depth discussions are outlined in the following. We are also noted that there are only six apps that use cryptographic algorithms to protect data security, but still existing the misuse of cryptography. However, none of them use sound encryption mechanisms. The fact also reveals that a large number of developers have no security experience and user's sensitive data is insecure to expose to the attackers. In order to consider what we found, we categorize the results into 2 types such as Plaintext and Proprietary Protocol.

Table 2. Result of Proprietary Protocol Analysis.

Potential Weakness	The Number of Apps
Plaintext	54
Hard-coded Key	3
Plaintext Key Exchange	3

The Content of Sending Data through Proprietary Protocol

As we all know, TCP provides reliable, ordered, and error-checked delivery of a stream between apps running on hosts communicating over an IP network, however it does not support safe channel to transport data. In consequence, it is an untrusted channel and easily exposed to attackers. According to the content of data streams, we summarize 3 categories as following.

Sensitive Data. As our analytical results presented, this plaintext channel send users' name, password, cell-phone number, geographic coordinates, device information, session token, even the history of shopping and price of commodities. In some public network environment, attackers just sniffer router and capture packets with network analysis tools, and they can also get a view of users.

Controlling Data. Game apps often employs some bites to control game process, such as the level of VIP, virtual product trades, health point of enemy, and heartbeat packets. With analyzing the meaning of each bites and modifying necessary values, game companies suffer great losses.

Pushing Message. Pushing message is a common phenomenon in the existing apps, which is offered to end users with notification and information from remote server to apps. Some of messages are significant since developer-run severs may send update message of versions and even some sensitive data. Most pushing message, as third-party libraries, are employed into apps, such as Igexin, JPush and MiPush in China. According to our discovery, only a small part of these pushing libraries use security channel to protect their messages. Igexin [4], one of the most popular pushing company, utilizes customize algorithm to encrypt messages and the key is sent with cipher message in plaintext.

The Misuse of Cryptography in Proprietary Protocol

According to the results of RawDroid, we conduct the misuse of cryptography on two selected third-party libraries from Instant Messaging Cloud provider. With the popularity of Android platform for mobile Internet industry, apps which add communication functions, use IM to improve user experience. We find that there are at least 30 apps, each of which has been downloaded for 2 million times from game and sociality categories, that use these two libraries. And more than 30 thousand subscribers chose these libraries in Android or iOS platform. According to analysis results, most of them use proprietary protocols with custom algorithm, thence, we investigate the point that whether there are security risks.

GotyeIM. *com.open.demo* is a sample app for GotyeIM, one of the most popular IM provider in China, which can download from its official website. Its original intention is that provides developers a demo to quickly adapt, and it contains all needed IM libraries. GotyeIM first sends its authentication information to server. When it succeeded, the server sends a 32-byte pain-text key *A* to its Android client. What is more, in native, app *xor* a hard-coded key *B* with the key *A* to product a real key. After pre-process, app sends the key *A* to chatting server and encrypts user's chatting messages, login information, session tokens and history records with 3DES-ECB. It is worth mentioning that this app sends request data with the key *A* and a constant string AES" as a fake algorithm tip.

RongLian. *com.yuntongxun.ecdemo* is a sample Android app for another popular IM provider called RongLian which has cooperation with hundreds of companies. We download the demo from its website and analyze it manually. The result shows that this third-party library encrypts user's chatting history and messages to chat server with hard-coded key. The app defines a function called AES, however it implements as a simple encryption method that encrypts data based on a permutation table. The most noteworthy is that its decryption function can be searched by Google [5]. We try this Google's version to encrypt and decrypt app's data, not surprisingly, we easily obtain user's private data. Finally, the library encapsulates all data into *Protocol Buffers*, a method of serializing structured data without data encryption by Google, and sends it in TCP.

Conclusion

We conduct the first step to investigate the usage and security of network protocol in the most popular Android apps. Our result shows that 36.7% apps use insecure communications. We design and implement a tool, RawDroid, to determine whether apps contain these vulnerabilities. Most of these vulnerabilities are caused by sending plaintext through HTTP or TCP. Also, some apps contain third-party libraries which misuse cryptographic algorithms to build proprietary protocols. We envision that our study can raise the attention of these vulnerabilities for security researchers and app developers.

References

[1] Fahl S, Harbach M, Muders T, et al. Why Eve and Mallory love Android: An analysis of Android SSL (in) security[C]//Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012: 50-61.

[2] Cai F, Chen H, Wu Y, et al. Appcracker: Widespread vulnerabilities in user and session authentication in mobile apps[J]. MoST 2015, 2014.

[3] Google Monkey on http://developer.android.com/guide/developing/tools/monkey.html.

[4] Igexin on http://www.wooyun.org/bugs/wooyun-2010-0185354

[5] CodeSearch on https://searchcode.com/codesearch/view/13566752/