International Journal of Research in Engineering and Science (IJRES)
ISSN (Online): 2320-9364, ISSN (Print): 2320-9356
www.ijres.org Volume 9 Issue 5 ǁ 2021 ǁ PP. 56-60

Anti-Spyware

Mohit Kejriwal[#1], Naman Vyas[#2], Rishikesh Gorule[#3], Debarati Ghosal[#4]
[#]*Department of Information Technology*
Vidyalanakar Institute of Technology, India

***ABSTRACT**: -*
The essence of this technology paper lies in developing and using a simple VPN to prevent data leaks inside the phone. Android-based smartphones get huge benefits from their counterparts depending on the market among users. The increasing use of Android OS makes it a good target for attackers. There is an urgent need to create solutions that monitor user privacy and that can monitor, access and block these Eavesdropping applications.
***KEYWORDS**— Android, Data leaks, Privacy.*

Date of Submission: 27-04-2021 Date of acceptance: 11-05-2021

I. INTRODUCTION

A. *Definition*

As smartphones increase in prevalence and functionality, they have become responsible for a greater and greater amount of personal information. The information smartphones contain is arguably more personal than the data stored on personal computers because smartphones stay with individuals throughout the day and have access to a variety of sensor data not available on personal computers. Developers of smartphone applications have access to this growing amount of personal information; however, they may not handle it properly, or they may leak it maliciously. The Android smartphone operating system provides a permissions-based security model which restricts application's access to user's private data. Each application statically declares its requested permissions in a manifest file which is presented to the user upon application installation. However, the user does not know if the application is using the private locally or sending it to some third party. To combat this problem, we present AndroidLeaks, a static analysis framework for finding leaks of personal information in Android applications. To evaluate the efficacy of AndroidLeaks on real world Android applications, we obtained over 23,000 Android applications from several Android markets. We found 9,631 potential privacy leaks in 3,258 Android applications of private data including phone information, GPS location, WiFi data, and audio recorded with the microphone.

II. EXISTING WORK

Jiyuan Sun, Shaozhen Ye, Tao Shang, Jianwei Liu, Qi Lei in their paper raised that both benign and malicious developers are attracted to Android platform because anyone is allowed to publish applications on the Android market. Such capability leak vulnerability on the Android platform may lead to permission elevation and privacy disclosure by making malware bypass Android security mechanism. This paper presents a code scanner tool - Droidprotector which is applied to help developers search bugs and focus on the business of applications rather than the security problems. Firstly, Markov blanket is used for feature selection. Secondly, source code is analyzed by a machine-leaning method. Finally, malicious intents and capability leaks are detected. By collecting 3482 applications and 59 source files to learn Markov blanket as the feature set and testing this code scanner tool, the experimental results show that DroidProtector can detect the vulnerability of Android source code effectively by using Markov blanket to select features correctly.

Julian Schutte, Dennis Titze, J.M. de Fuentes in their paper proposed AppCaulk, an approach to harden any existing Android app by injecting a targeted dynamic taint analysis, which tracks and blocks unwanted information flows at runtime. Critical data flows are first discovered using a static taint analysis and the relevant data propagation paths are instrumented by a taint tracking code at register level. At runtime the dynamic taint analysis woven into the app detects and blocks data leaks as they are about to occur. In contrast to existing taint analysis approaches like Taint droid, AppCaulk does not require modification of the Android middleware and can thus be applied to any stock Android installation. In this paper, we explain the design of AppCaulk, describe the evaluation of its prototype, and compare its effectiveness with Taintdroid.

Wooguil Pak, Youngrok Cha, Sunki Yeo in their paper proposed a new approach to protect the private phone number data in smartphones from leaking and stealing through malicious applications. Our approach differently deals with trusted and suspicious applications and avoids the malfunctions of applications caused by

security policy. Most of outstanding feature of our approach is that it can detect the leakage of the private phone number data, trace the leaked data and finally identify which application leaked them. Furthermore, it can minimize the damage owing to the data even though it is abused for cyber-crimes.

Mustafa Hassan Saad, Ahmed Serageldin, Goda Ismaeel Salama in their paper suggested that Android-based smartphones get huge benefits from their partners in terms of market share among users. The increasing use of Android OS makes it a good target for attackers. There is an urgent need to develop solutions that monitor user privacy and that can monitor, detect and block these Eavesdropping applications. In this paper, two proposed ideas are presented. The first proposed paradigm is a spyware program to highlight the "disease" security vulnerabilities. The spy-ware app has been used to deepen our understanding of the vulnerability of the Android app, and to learn how spy-ware can be built to thwart this victim's privacy risk such as received SMS, incoming calls and outgoing calls. Spy-ware abuses an Internet service to transfer captured information from the victim's cell phone illegally to a cloud database. The Android OS's low-level console system and streaming system offer to create a spy environment by giving it full control over the victim's listening, capturing and tracking privacy. The second proposed method is a new method of obtaining a “drug” based on the fuzz testing process to reduce known risks. In this proposal, an anti-spy-ware solution "DroidSmartFuzzer" was developed. The implementation of an anti-spy-ware application has been used to reduce the risk of alleged attacks.

III. PROPOSED SYSTEM

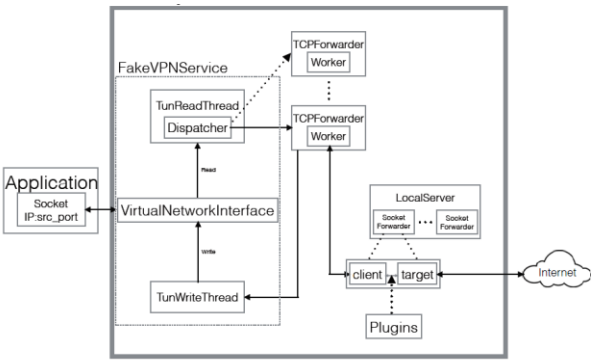

Fig 3.1

Anti-Spyware detects network traffic from a virtual network interface provided by VPNService and returns it after analysis. Figure 5.1 introduces the five main Anti-Spyware features: FakeVPNService, TCPForwarder (or UDPForwarder), LocalServer, SocketForwarder, and Plugins. Because everything except transmitters is the same between TCP and UDP and TCP transfers are very important and complex, the following definitions are based on TCP unless otherwise stated.

IV. METHODOLOGY

The following is the basic workflow of Anti-Spyware:

i. An application sends a request to a server. The request is then retrieved from the virtual network interface in the Fake VPN Service.

ii. The Fake VPN Service parses the request contained in an IP datagram and dispatches the request to the corresponding forwarder. In the TCP case, a TCP Forwarder will handle the request.

iii. The TCP Forwarder implements TCP. The TCP Forwarder communicates with the Local Server, which is running on the TCP layer. The Local Server acts like a man-in-the-middle proxy.

iv. For each request received from a TCP Forwarder, the Local Server retransmits the request to the intended server in the Internet (from now on called real server") and also retransmits the response from the real server to the TCP Forwarder. In this step, all installed Plugins are invoked to filter outgoing and incoming data.

v. The TCP Forwarder packages the response from the Local Server into an IP datagram and sends the datagram through the virtual interface to the application.

(A) The Fake VPN Service class:

The class VPNService5 has been part of the Android SDK since API level 14 (Android 4.0). It creates a virtual network interface, configures addresses and routing rules, and returns a file descriptor to the application. From the descriptor, the application can read all network traffic.

The class FakeVPNService extends the class VPNService. FakeVPNService establishes a virtual network interface with the IP address 10.8.0.1. It also defines routing rules to route network traffic sent to all IP addresses to the interface. After properly setting up FakeVPNService, the Android system will route all network traffic from all other applications to this virtual network interface in FakeVPNService. FakeVPNService sets up several threads. These threads are described next.

The TunReadThread thread reads all requests from the virtual network interface. Because many applications and services can run on the device, the amount of requests can be large. There would be heavy performance overhead due to the high network I/O cost if the TunReadThread thread handled the transmission to other components as well. To avoid this cost, this thread only adds these requests to a queue.

The Dispatcher thread is created by the TunReadThread thread. The Dispatcher thread keeps reading requests, which are IP datagrams since VPN works on the IP layer, from the queue mentioned above. The Dispatcher thread parses these IP datagrams and retrieves the protocol field from the IP header to see whether the datagram wraps a TCP packet or a UDP packet. If the datagram contains a TCP packet, a TCPForwarder thread bound to the source port number of the packet is used to handle this packet. If there has already been a forwarder for the port number, this forwarder is used. Otherwise, a new forwarder is created and bound to that port number. The reason for using the old forwarder is that a TCP connection is stateful. The forwarder maintains the state of the TCP connection.

The TunWriteThread thread retrieves responses from a queue and writes these responses to the virtual network interface. The responses are added to the queue by forwarders by calling the TunWriteThread.write() method. Because this method requires IP datagrams as arguments, forwarders need to reconstruct valid IP datagrams from TCP/UDP data these forwarders have. The main part of the reconstruction is reversing the source and destination IP addresses in the request and updating the IP checksum.

(B) TCP Forwarder Class:

To maintain the connection states required by TCP, we build a mapping relationship between one TCPForwarder and one TCP connection. Since a TCP connection can be determined by the source IP address, which is the same for all applications, and the source port number, the relationship is based on the source port number. TCPForwarder implements the TCP state machine. The following explains the implementation through a typical TCP connection life cycle:

1. When a TCPForwarder is initialized, it is in the LISTEN state and waiting for a 3-way handshake. When the TCPForwarder receives a SYN packet from an application, it will respond with a SYN_ACK packet to the application and go into the SYN_ACK state.
2. After receiving the SYN_ACK packet, the application will send an ACK packet, and the TCP connection is established. The TCPForwarder will then go into the DATA state after receiving that ACK packet. Also, the TCPForwarder will connect to the LocalServer.
3. In the DATA state, the TCPForwarder transmits each packet received from the Dispatcher to the LocalServer and responds with an ACK packet to the application. If the LocalServer sends anything back, the TCPForwarder will also transmit it to the application with an appropriate sequence number.
4. Whenever the application sends a FIN packet, the TCPForwarder goes to the HALF_CLOSE_BY_CLIENT state, does some termination work, and then goes to the CLOSED state.
5. Whenever the TCPForwarder receives a FIN packet from the LocalServer, it goes to the HALF_CLOSE_BY_SERVER state, does some termination work, and then goes to the CLOSED state.

The TCP connection states and related information are stored in TCPConnectionInfos. The TCPConnectionInfo class implements all necessary methods to read and update the states.

Because TCP is a stream delivery service, one request of a protocol using TCP (e.g., HTTP) may be divided into multiple TCP packets. In this case, using blocking transmission would be complex and introduce higher performance overhead. In our implementation, we use two threads. One thread sends requests to the LocalServer and the other reads responses from the LocalServer when it is readable.

(C) The Local Server Class:

From the point of view of applications, the LocalServer is the real server. Applications communicate only with the LocalServer. The LocalServer listens on a specific port, which is known and connected to by all TCPForwarders (or UDPForwarders). When a TCPForwarder connects to the LocalServer, the LocalServer creates two sockets: target and client. Then the LocalServer creates a SocketForwarder using these two sockets. The target socket connects to the real server; its IP address and port number are determined as follows: The

socket used by a TCPForwarder to connect to the LocalServer uses the same port number as the corresponding application (but the IP address of the socket is different). Therefore, we can use the port number to determine to which IP address and port number the application actually wants to connect. In Linux and therefore also in Android, this information can be found in /proc/net/tcp or /proc/net/tcp6.

V. RESULT

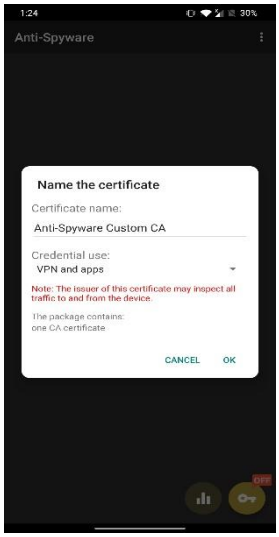

Fig 4.1
A snapshot of certificate installation.

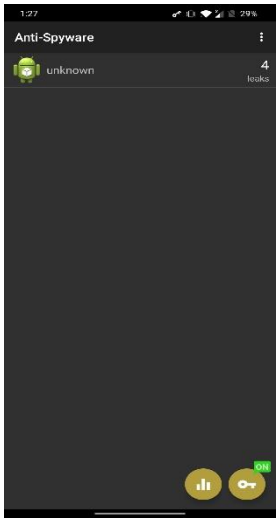

Fig 4.2

After everything is setup, the user has to turn on the VPN and run any application in background so as to catch the leaks done by running applications.

VI. FUTURE SCOPE

Future work should consider adding a better UI for better user experience. Also adding data leak done using camera, gyroscope and microphone.

VII. CONCLUSION

We propose and implement a new approach, Anti-Spyware, to filter network traffic on Android. Anti-Spyware does not require root permissions and is portable to all devices with Android versions 4.0 or later. It is easy to use without any knowledge about security or privacy. Anti-Spyware can be used to effectively detect information leakage.

REFERENCES

[1]. Jiyuan Sun, Shaozhen Ye, Tao Shang, Jianwei Liu, Qi Lei. – DroidProtector, In ICGI, 2017.
[2]. Julian Schutte, Dennis Titze, J.M. de Fuentes – AppCaulk, In IEEE, 2014.
[3]. Wooguil Pak, Youngrok Cha, Sunki Yeo – Detecting and tracking leaked private number in Android smartphones . In IEEE, 2015.
[4]. Mustafa Hassan Saad, Ahmed Serageldin, Goda Ismaeel Salama – Android Spyware Detection and Medication, In InfoSec, 2015.