Open Source Based Privacy-Proxy to Restrain Connectivity of Mobile Apps

Alexander Mense
University of Applied Sciences
Technikum Wien
Vienna, Austria
mense@technikum-wien.at

Sabrina Steger
University of Applied Sciences
Technikum Wien
Vienna, Austria
ic14m010@technikum-wien.at

Dragan Jukic-Sunaric
University of Applied Sciences
Technikum Wien
Vienna, Austria
ic14m035@technikum-wien.at

András Mészáros
University of Applied Sciences
Technikum Wien
Vienna, Austria
ic14m042@technikum-wien.at

Matthias Sulek
University of Applied Sciences
Technikum Wien
Vienna, Austria
ic14m033@technikum-wien.at

ABSTRACT

Mobile Devices are part of our lives and we store a lot of private information on it as well as use services that handle sensitive information (e.g. mobile health apps). Whenever users install an application on their smartphones they have to decide whether to trust the applications and share private and sensitive data with at least the developer-owned services. But almost all modern apps not only transmit data to the developer owned servers but also send information to advertising-, analyzing and tracking partners. This paper presents an approach for a "privacy- proxy" which enables to filter unwanted data traffic to third party services without installing additional applications on the smartphone. It is based on a firewall using a black list of tracking- and analyzing networks which is automatically updated on a daily basis. The proof of concept has been implemented with open source components on a Raspberry Pi.

Categories and Subject Descriptors

ACM 1998: • **K.4.1 Computers and Society:** Public Policy Issues – *Privacy;* • **C.2.0 Computer-Communications Networks:** General – *Security and Protection*

ACM 2012: • **Security and privacy**: Security services – *Pseudonymity, anonymity and untraceability; Privacy-preserving protocols*.

General Terms

Security

MoMM '16, November 28-30, 2016, Singapore, Singapore

© 2016 ACM. ISBN 978-1-4503-4806-5/16/11…$15.00

DOI: http://dx.doi.org/10.1145/3007120.3007163

Keywords

Mobile Apps; Security; Privacy; Sensitive Information; Proxy.

1. INTRODUCTION

With their big variety of functions and their connectivity mobile devices have partially replaced the Personal Computer. They enable access to email, provide calendars to save all your appointments, ensure connectivity with your friends and because of the thousands of apps their functionalities are endless extendable. But they also store large amount of personal data as well as information about others like friends or business partners. There are more benefits: they are able to supply sensors and with the help of these sensors it is possible to use GPS for navigation, to use Bluetooth to collect information about your heart rate or to make pictures wherever you are. All these small parts together make mobile devices currently best opportunity available for getting private data as well as profiling, e.g. GPS data can give hints about the workplace and also the living address. Collected health information makes it furthermore possible to gain knowledge about the physical health [1]. While third-party applications open even more possibilities for smartphones, through access to user data they serve as a tool for profiling. Therefore it seems it is not possible to stay anonymous anymore [2].

It has been shown in several different evaluations [1] [2], [3], [4] that a big part of the mobile apps have security and/or privacy issues. A study from Huckvale et al [4] about the analysis of apps included in NHS England's Health Apps Library shows the lack of security and privacy of many mHealth apps and reports that most apps do not handle data according to their privacy policy, and that some apps do not even have any privacy policy.

One of the biggest problems is that almost no mobile app only connects to the developer-owned backend services, but also contacts third-party websites for advertising, analytics ad tracking. The communication with the advertising sites occurs mainly unencrypted. Some of the analyzed applications also send "usage data" in plain text to third-party advertisers after e.g. measuring fitness activities in a fitness app.

2. GOAL AND CONCEPT

There are already software products for mobile devices, which offer functionalities to limit connectivity (e.g. firewalls), limit access to data (SRT AppGuard) or directly try to enhance privacy (e.g. TaintDroid). But they have to be installed directly on the device and usually need a rooted device.

In order to hinder sniffing activities of apps we choose a different approach, which is able to prevent that advertising-, tracking- and analyzing networks are contacted by the apps.

The concept is to provide a service in the network (proxy), forward all network traffic from the mobile device to the where the traffic is analyzed. In a first step the goal for the work described in this paper was to disable connections to ad-, analytics- and tracking services. In a further step network traffic can be analyzed to identify and prevent possible leakage of sensitive data for a better protection of privacy.

To block connections to unwanted service the following components are employed:

- Use a VPN to tunnel all network traffic from the mobile device to the proxy service
- Use a firewall to filter and block traffic to ads-, analytics- and tracking services
- Automatically update firewall rules based on a list of ads-, analytics- and tracking services

Figure 1 shows the usual data flow from mobile apps, figure 2 illustrates the concept of the privacy proxy.

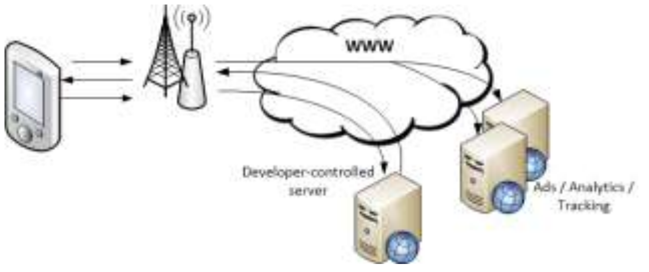

Figure 1: Typical mobile app connections

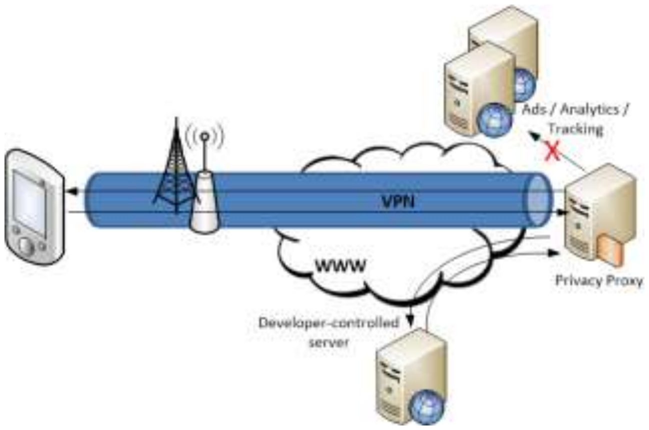

Figure 2: Connections using the privacy proxy concept

3. IMPLEMENTION

For a proof of concept the service was implemented on a Rasperry Pi using open source tools and components.

This section secure communication from the mobile device to the Raspberry Pi is described and the second part explained the filtering based on IPTables firewall blacklists and shows an automatic solution to update the unwanted domains on a daily basis.

3.1 Hardware - Raspberry Pi

The use of Raspberry Pi is a cheap, always-on, and simple way for users to create their own VPN endpoint. Raspberry Pi is based on open source software and therefore it is a transparent and secure solution for protecting privacy.

3.2 VPN - strongSwan

To be sure that every single packet is being sent to the Raspberry the VPN connection needs to be stable over long periods of time. Also the different smartphone operating systems need to offer the same IPSec possibilities. One possibility which is actually quite easy to implement with Raspberry Pi packages and is also supported by iOS and Android devices is IPSec with the Extensible Authentication Protocol, in short IPSec XAUTH.

The standard IPSec packages for the Raspberry Pi are openSwan1 (now named libreSwan) and strongSwan2 (strongSwan is available through non-standard repositories). But because of the higher stability, the more frequent update cycles, and the better documentation we decided to use strongSwan [Z] instead of openSwan. For the XAUTH compatibility the charon library and some extra packages (libcharon-extra-plugins) need to be installed. IKEv1 was used to be compatible also older versions of mobile operation systems.

3.2.1 Authentication

For the authentication the XAUTH option with pre-shared keys is used. Afterwards strongSwan should look for a username/password combination in the ipsec.conf file. As this is a very simple installation, the username and the password are stored in cleartext in the ipsec.conf file. For the routing/forwarding options the sysctl.conf has to be modified, as it needs to forward packets. To provide stronger security different RSA methods are provided by strongSwan. For future implementations we recommend using at least a server certificate authentication prohibiting man-in-the-middle attacks.

3.2.2 Stability

As the Raspberry Pi has to be reachable at any time we propose the use of a dynamic DNS solution for the private home network, in case the internet service provider (ISP) is changing IP addresses frequently. We also recommend editing the Raspberry Pi's interfaces file and assign a static IP address. Unfortunately the Raspberry Pi WiFi driver is poorly implemented, resulting in low VPN speed and/or connection losses. Thus we suggest using cable over WiFi. In order to enable VPN connection the ISP modem needs to be able to do port-forwarding on UDP port 500 and 4500 from the official IP to the Raspberry Pi private/local IP.

3.2.3 Security

In association with strongSwan [7] a large variety of IKEv1 cipher suites are being offered [8]. This is a clear advantage, as the offered cipher suites on the smartphones vary from operating system to operating system and also from version to version. The selected suite for a specific connection is visible in /var/log/daemon.log. For example, below is the IPSec stack, primarily chosen by android devices:

IKE:AES_CBC_256/HMAC_SHA2_256_128/PRF_HMAC_SHA2_256/MODP_1024

The first part describes the encryption, the second and third the integrity hashing algorithms used and the final part the Diffie Hellman groups used for the key exchange process. The example above shows that the used protocols offer a high grade of security for private VPNs. With strongSwan we ensure that the established tunnel, although open source software is being used, offer a high level of security.

3.2.4 Testing

By keeping the tunnel alive over a longer period of time (about two hours) the stability was tested. If the device reaches a Telecom dead zone, the re-establishing process occurred immediately and no further problems were detected. To minimize traffic being sent over the mobile data network, we recommend compressing the data stream. strongSwan, per default does not compress data streams. However this feature can be easily activated with the compress=yes parameter in the ipsec.conf file. Unfortunately this feature can cause incompatibility with some iOS and Android versions.

3.3 Traffic control with IPTables

Blocking of unwanted connections to ad-, analytics- or tracking-services is done with the IPTables firewall [X] which is part of almost all Unix/Linux operating system versions.

3.3.1 Automatic update of hosts to block

A static blocking list which needs to be manually updated periodically is not an optimal and permanent solution. Therefore the list of hosts to block needs to be adapted automatically on a daily basis. The privacy service has to cope with this ever changing environment. The automatic update is done on the basis of a black list containing ad-, analytics- and tracking servers.

3.3.2 Blacklist provider

There are several providers who publish lists of tracking and advertising hosts. To select a blacklist we defined two criteria: the reliability/reputation of the blacklist provider and regular updates of the list. Based on these criteria Shalla Secure Service KG [9] was selected which provides a blacklist with over 1.7 million entries. Additionally it is free of charge for personal usage.

3.3.3 Blacklist format

The used lists contain domain names and IP addresses which need to be transformed for the use with IPTables, as it is primarily designed to work with IP addresses. Furthermore periodical control of the domain names for changed IP's is needed.

3.3.4 Requirements and Tools

For the automatic update of the filter tables for IPTables we defined the following requirements:

- Automatic run, without user action
- Possible use with different list providers
- Easy deployment and use on the Raspberry Pi
- The ability to download the source list from the internet, but also read local definitions
- Fast update of the blocking rules in IPTables (to minimize time of only partial protection)

To achieve these goals and implement the functionalities a combination of the Perl scripting language and the Linux Bash Shell scripting was used. Both are available over Raspberry PI's repository as well. Among the scripting languages Perl is the best for text processing. The language design includes built-in functions for regular expressions, which makes the manipulation of text-based files easy. For file and directory management, we decided to use callout to Bash Shell scripts, which provides a more compact and clear method. To achieve an automatic start of the "HostlistLoader" tool the built-in Linux Cron is used.

3.3.5 "HostlistLoader"

The main parts of the developed application to load the list with the hosts to block and to convert them into appropriate firewall rules are:

- HostlistLoaderConfig.xml (The configuration file) defines parsers, which interpret the sources.
- LocalFileParser: The LocalFileParser module loads only the locally stored list of blacklisted hosts (defined either by IP or by domain name).
- InternetParser: The InternetParser module loads a TAR GZIP file from the Shalla Secure Services KG5 provider. This parser contains some special actions, like preparing the loaded file. In the current case the preparation is implemented in a different Shell script, the defaultInternetParser_prepare.sh file, which extracts the source compressed file in a temporary directory, selects the tracking and advertising lists, concatenates them and places the resulting file to a defined directory. Further processing of the prepared list happens subsequently in the parser module.
- StartupFileParser: The StartupFileParser module is a built-in optimization for a special case: the system startup. At system start, the resources of the system are heavily loaded with system tasks and a complete cold start would take longer time.

The whole "HostlistLoader" automation tool consists of severeal Perl modules for configuration management, maintaining the IP list in the memory, converting and applying the IP list as IPTables rule sets as well as logging.

3.3.6 Optimization

All parser modules (except the special built-in StartupFileParser) contain a domain caching method. If the source, loaded from the providers, contains domain names, the automation resolves them, but it takes more time to do so (approximately from a few hundred milliseconds to a few seconds, depending on the network). As the IP assignment to domain names changes relatively rarely it makes another optimization possible. The resolved domain names and the resulting IP addresses can be stored in a defined domain cache file for all the parser modules defining the cache. As a mandatory attribute (keep_days), the validity period should be given. The automation always checks the age of the cache, and if needed, it makes a new domain name resolution.

3.3.7 Logging

The updating module implements a comprehensive logging. All the successful steps are indicated by an "[INFO]" tag and all problems are marked by "[ERROR]" tags. This makes filtering of the occurring problems easy. The output log file is also configurable through the start parameters of the main script.

3.3.8 IPSet

During the tests we experienced slow performance if large amount of IPTables rules were used. Therefore we switched from IPTables to IPSet [10] which is a framework included in the Linux kernel. IPSet performs the same task 11-times faster than IPTables [11]. Furthermore IPSet also automatically can block the whole port range for a specific IP address. With this it is guaranteed that hidden communication over nonstandard ports is also blocked.

After all parser modules have finished their work, the IPSet module takes the list of IP addresses created in the memory and generates an IP hash set in the Linux system. If the set already exists, it erases the old set and fills in the newly constructed list. At this point an approximately two to five minutes long "non-protection" gap occurs. Future improvement might be to change the flushing mechanism to a synchronising mechanism, closing this small security gap.

4. DISCUSSION AND FURTHER WORK

The concept of implementing privacy controls in form of intermediary services is not fundamentally new. There has been work in the area of protection of user location data (e.g. [12]) as well as rule-based release of data (e.g. [13]). Many models propose to upload data to trusted stores and control access of application to this data, which is quite different to the concept of real-time filtering of data streams. Davies et al. did similar work in the area of Internet of Things (IoT) [14]. They proposed an architecture connecting IoT devices with specific data drivers to "Privacy Mediators" running in private Cloudlets [14]. To some extent our privacy proxy can be interpreted as such a mediator service.

But in difference in the presented approach there is no need for specific data connectors as all network traffic is routed through the VPN to the filtering service. Such a network proxy is definitely a straight forward way to control connections and data traffic of mobile apps.

Data traffic of mobile applications can be easily analyzed by and - as it is shown in section 4 - it is easy to implement. But such a proxy could also be an effective service to scan and analyze the network traffic to detect data leakage.

The biggest advantage of the proxy solution is the displacement of the logic from the mobile device to a server architecture. No app installation, no rooting and/or flashing of the device is needed. The user is able to use the mobile device operating system's native VPN functions to establish a secure connection to the user's own Raspberry Pi. This way of transferring the data from a privately controlled device guarantees that no third party can steal them. The Raspberry Pi uses a freely available and updated blacklist to make blocking decisions. The mobile device only has to support VPNs. Various other infrastructure solutions would require more logic on the client side and would therefore not lead to a universal solution.

The traffic from the servers can be easily controlled with a firewall and furthermore the data can be classified and analyzed with a variety of methods such as pattern matching (using regular expressions or word blacklisting), taint analysis or machine learning.

Filtering the network traffic with a firewall is quite normal in any enterprise environment and usually does not have any great performance impacts on outgoing traffic. As the current solution of the proxy uses an optimized firewall implementation there are only minimal performance implications.

The proof-of-concept implementation is based on a Rasperry Pi and enables private configuration and usage of the proxy. But the concept can be also implemented in a cloud based environment as "privacy-as-a-service".

5. REFERENCES

[1] M. B. Barcena, C. Wuesst, H. Lau, "How safe is your quantified self?", Symantec, August 2014. https://www.symantec.com/content/dam/symantec/docs/white-papers/how-safe-is-your-quantified-self-en.pdf, Last access: 20.08.2016

[2] A. Barczok, S. Porteck, "Schutz vor Schnüffel-Apps," Appgehört, Schnüffel-Apps einschränken, April 2015.

[3] A. MENSE, A. Steger, M. Sulek, D. Jukic-Sunaric, A. Mészáros: Analyzing privacy risks of mHealth applications. In: Transforming Healthcare with the Internet of Things, Studies in Health Technology and Informatics, vol 221, IOS Press, 2016, ISBN 978-1-61499-632-3 (print) | 978-1-61499-633-0 (online), doi:10.3233/978-1-61499-633-0-41

[4] K. Huckvale, J. Tomás Prieto, M. Tilney, P.-J. Benghozi, and J. Car, Unaddressed privacy risks in accredited health and wellness apps: a cross-sectional systematic assessment. BMC Medicine 2015, 13:214. doi:10.1186/s12916-015-0444-y

[5] Wu-Chen Su, A Preliminary Survey of Knowledge Discovery on Smartphone Applications (apps): Principles, Techniques and Research Directions for E-health, International Conference on Complex Medical Engineering, pp.369-374, IEEE Computer Society Press, June 2014

[6] https://www.openswan.org/

[7] https://www.strongswan.org/

[8] StrongSwan: IKEv1 Cipher Suites. https://wiki.strongswan.org/projects/1/wiki/IKEv1CipherSuites, Last access: 20.08.2016

[9] Shalla Secure Services KG: Shalla Secure Services - Shalla's Blacklists. http://www.shallalist.de

[10] http://ipset.netfilter.org/

[11] d(a)emonkeeper's purgatory: Mass-blocking IP addresses with ipset. http://daemonkeeper.net/781/mass-blocking-ip-addresses-with-ipset, Last access: 20.08.2016

[12] A. R. Beresford, F. Stajano: Location privacy in pervasive computing. IEEE Pervasive Computing, 2(1), pp46-55, 2003.

[13] M. Mun, S. Hao, N. Mishra, K. Shilton, J. Burke, D. Estrin, M. Hansen, R. Govindan: Personal data vaults: a locus of control for personal data streams. In Proc. of ACM CoNEXT, p17, 2010.

[14] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, B. Amos: Privacy Mediators: Helping IoT Cross the Chasm. In Proceeding HotMobile '16 Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications, pp 39-44, ACM New York, NY, USA 2016, ISBN: 978-1-4503-4145-5, doi: 10.1145/2873587.2873600