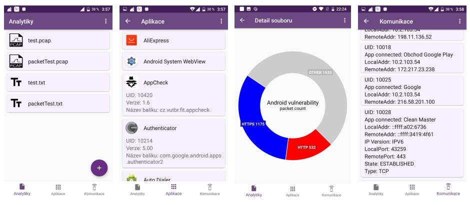
BRNO UNIVERSITY OF TECHNOLOGY FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2019
http://excel.fit.vutbr.cz

Android App for Security Monitoring of Communication

Karolína Klepáčková*



UID: 10420
Verze: 1.6
Název balíku: cz.vutbr.fit.appcheck

UID: 10214
Verze: 5.00
Název balíku: com.google.android.apps .authenticator2

RemoteAddr: 198.11.136.52

UID: 10018
App connected: Obchod Google Play
LocalAddr: 10.2.103.54
RemoteAddr: 172.217.23.238

UID: 10025
App connected: Google
LocalAddr: 10.2.103.54
RemoteAddr: 216.58.201.100

UID: 10028
App connected: Clean Master
LocalAddr: ::ffff:a02:6736
RemoteAddr: ::ffff:3419:4f61
IP Version: IPV6
LocalPort: 43259
RemotePort: 443
State: ESTABLISHED
Type: TCP

Abstract
Global problem with mobile applications whose provide some kind of internet communication is that any of them doesn't provide information about used protocols. Users connecting to public networks are at risk because attackers can eavesdrop on unsecured communications. Knowing the communication protocol and data provided to the application, a knowledgeable user can decide if it's trustworthy or not. This article aims at an Android application which enables security monitoring of other applications' communication which is called AppCheck. Describes the solution of major problems for analyzing network communication of given mobile application. A reader can find out useful information about how to circumvent the Android restrictions on getting information from other applications. Specifically, it focuses on packet processing inside a mobile device and allocating data flow to individual applications. Thanks to the solution of these problems it can find out how much is the application safe. The implemented application can catch network flow of another application and save it into pcap file which is readable eg. via Wireshark tool. This file is used for making analytic to recognize how much packets were sent in a secure way. All of these features are provided without the necessity of having higher user privileges than the common user has.

Keywords: Packet bypass — Android — Security

Supplementary Material: Youtube video of AppCheck — AppCheck on Google Play

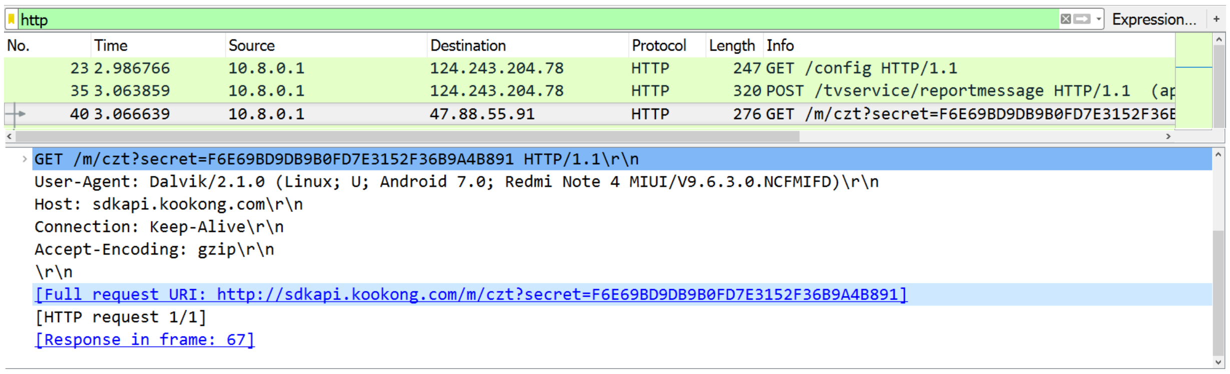
*xklepa04@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

This article describes the major pillars of my diploma thesis called *Android App for Security Monitoring of Communication*. The first idea of making an application like that came up from finding up a packet in a mobile device network communication which included plaintext string containing application's secret sent via HTTP protocol as you can see in the figure 1. The problem with this protocol is that it's not encrypted. It means that packet content is sent in a readable form. That's why I decided to design an application which can help a knowledgeable user to recognize potential security issue.

One of the main security issues comes up with connecting a mobile device to a public wireless network. Anyone can join it and listen to other's network communication. And it's the moment when a secure protocol comes in play. If you know that an application including your personal data implements only HTTP[1] instead of HTTPS[2], which is encrypted, then you might recognize that your privacy can be intruded.

[1]HTTP = Hypertext Transfer Protocol
[2]HTTPS = HTTP with Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocol



```
http                                                                    Expression...  +
No.   Time       Source      Destination       Protocol  Length  Info
   23 2.986766   10.8.0.1    124.243.204.78    HTTP         247  GET /config HTTP/1.1
   35 3.063859   10.8.0.1    124.243.204.78    HTTP         320  POST /tvservice/reportmessage HTTP/1.1  (ap
   40 3.066639   10.8.0.1    47.88.55.91       HTTP         276  GET /m/czt?secret=F6E69BD9DB9B0FD7E3152F36E

GET /m/czt?secret=F6E69BD9DB9B0FD7E3152F36B9A4B891 HTTP/1.1\r\n
User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.0; Redmi Note 4 MIUI/V9.6.3.0.NCFMIFD)\r\n
Host: sdkapi.kookong.com\r\n
Connection: Keep-Alive\r\n
Accept-Encoding: gzip\r\n
\r\n
[Full request URI: http://sdkapi.kookong.com/m/czt?secret=F6E69BD9DB9B0FD7E3152F36B9A4B891]
[HTTP request 1/1]
[Response in frame: 67]
```

Figure 1. A secret of an unknown application caught via Wireshark tool within 5 minutes of monitoring the network flow of mobile device. You can see it in a row with full request URI.

My application can provide information about the number of packets covered by secure protocol instead of insecure one. Moreover, it permits a user to choose if it will include statistics of all other applications or just one of them. Knowledge of which application sends your data unsecured might push you to uninstall it and never use it again. Or on the other hand, if you have a suspicious application which is somehow useful for you then you can analyze it and recognize that it's not dangerous. From my point of view, it's really important to keep personal data safe. That's why I searched for some applications which can provide additional information about how secure other applications are. I found some available on Google play but their biggest drawback is that they might not be understandable for a common user.

2. Existing security applications

I would like to briefly describe the most useful applications I found. They are: Netstat Plus[3], Packet Capture[4] and NetCapture[5]. The last of compared applications is AppCheck which is created by me. For a better idea of individual applications, I have created a summary table of their properties. You can see key features provided by each application in the table 1.

I will focus on clarifying the items in table 1. UDP and TCP indicate which transport protocols are tracked. Same as IPv4 and IPv6 says if are they supported and specified for each network connection. Destination and source port indicates if you can find out numbers of these ports. SSL support means that the application allows you to monitor content of packets which are using the HTTPS protocol. Application info provides you if there is specified the application name, its version, and its package name. This information is helpful in accurately identifying the application we want to monitor. Show connections provides information about the current network connection and the state info is connected to it. Track send packets tells if the applications is able to record ongoing communication. This also includes information about the size, timestamp, and content of each packet. Furthermore, it is good for the user to know what possibilities he/she has in choosing the monitored application. If it's single application, multiple applications or all of them.

	NP	PC	NC	AC
UDP	Y	N	Y	Y
TCP	Y	Y	Y	Y
IPv4	Y	Y	Y	Y
IPv6	Y	Y	Y	Y
Destination port	Y	Y	Y	Y
Source port	N	N	N	Y
SSL	N	Y	Y	N
App info	N	N	N	Y
Show connections	Y	Y	Y	Y
Connection state	Y	N	N	Y
Track send packets	N	Y	Y	Y
Track packet size	N	N	Y	N
Packet timestamp	N	Y	Y	N
Packet content info	N	Y	Y	N
Track one app	N	Y	Y	Y
Track multiple apps	N	N	N	Y
Track whole flow/app apps	N	Y	Y	Y
Save flow to pcap file	N	Y	N	Y
Analyze network security	N	N	N	Y

Table 1. Table of comparison of individual applications according to their properties. Abbreviations of compared applications: NP = Netstat Plus, PC = Packet Capture, NC = NetCapture, AC = AppCheck. Abbreviations of table values: Y = YES, N = NO.

[3]Available at: https://play.google.com/store/apps/details?id=com.rinacode.android.netstatplus&hl=en_US

[4]Available at: https://play.google.com/store/apps/details?id=app.greyshirts.sslcapture&hl=en_US

[5]Available at: https://play.google.com/store/apps/details?id=com.minhui.networkcapture&hl=en_US

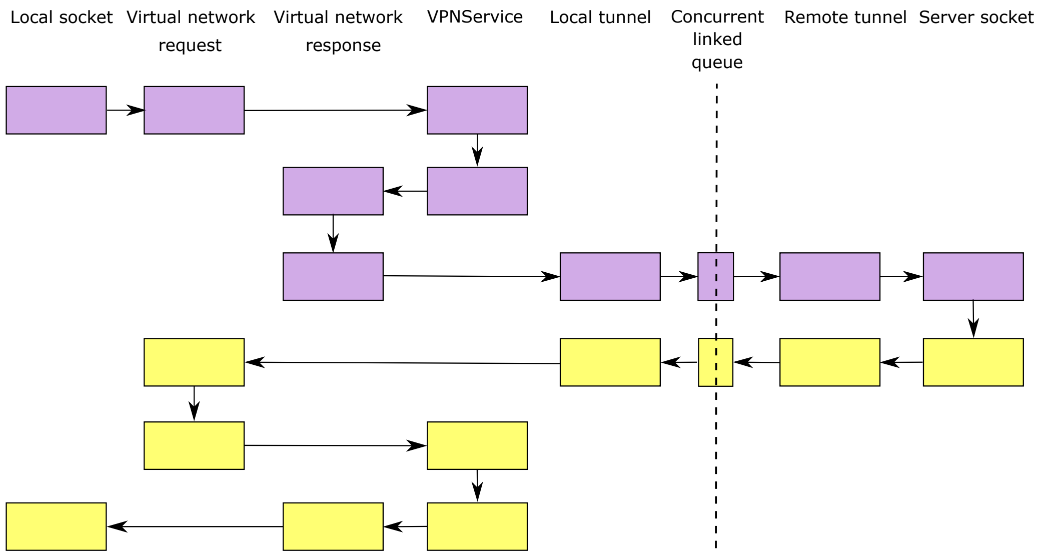


Figure 2. Packet bypass visualization through interfaces designed by author of article [1]. Blocks with violet color are used for sending packets from a mobile device. The yellow ones are used for receiving them. Individual blocks used for packet handling inside the mobile device can be seen in the figure separated by a hatched line on the left. Blocks used for data sending to the desired destination are located on the right.

For later processing, it is also necessary to save the pcap file for the recorded network flow. The last criterion is whether the application supports analyzing the recorded network flow.

To make it clear how are contents of packets using HTTPS protocol revealed. It's possible by installing a certificate offered by the application. As a result, a man-in-the-middle attack on a given communication can be made, and both communicating parties are forced to send their messages through an attacker. In this case, through the installed application. The installation of user certificates has undergone a change with Android N operating system. However, even with Android N and later ones there are ways to install such a certificate. If we want do do such operation in an application we have to customize trusted certificate authorities using Network Security Configuration.

To sum up there are few other solutions providing info about sent packets content. But they have a big disadvantage in the way they show up info for the user. It means that the added value of my application over others available on the market is the final analytic of the recorded network traffic. It also provides the ability to record multiple applications at the same time, while others provide the ability to record only one application or all of them. In contrast, two of the three tested applications allow you to look into HTTPS-secured communications, which AppCheck cannot. The last huge different is that AppCheck doesn't provide information about packet content. It's because the main aim of the application is providing analytic of transferred packets instead of let user search in them.

3. Packet capture on Android

The core of my application is to work with packets and their contents. The first prerequisite to work with them is to capture them as they pass through the network elements of the mobile device. VPN is useful for this purpose as it can be implemented by *VpnService* class in *android.net* package. It creates a point in the application through which all the data from and to the mobile device passes. Within this point it is necessary to take over individual packets and forward them to their original destination. As long as individual packets are forwarded, it is possible to save them for later processing. Subsequently, the analysis of the entire network flow can be processed from the individual stored packets.

The implementation of the VPN itself can be solved in two ways. The first is to create a VPN connection to a remote server, where it forwards the individual packets and where it is routed back to the application. The second option is by bypassing packets. In this case, the created VPN connects only to the virtual interfaces created within the mobile device. Thus, packets are never sent to remote servers. The thing which is done here is both they are stored in a pcap file and sent to its original destination via the mobile device's output network interface. So the second option will be referred to in the article as a packet bypass.

As the theoretical basis of packet capture, I've used mainly two articles. The first called *Android VPN Service Explained with Packet Bypass Example Program* [2] was from Mr. Terrence Sun, who focused on packet capture theory. It includes a code example

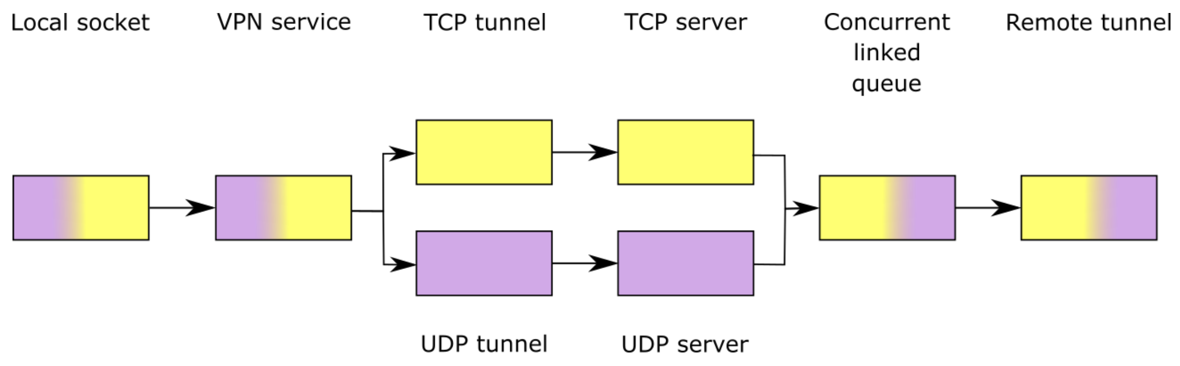


Figure 3. Schema of solution used for packet handling in AppCheck application. The yellow color indicates the flow of TCP packet. The violet color indicates flow of UDP packet. If any of these blocks has both colors it means that it's used for bypassing any type of packet. Capturing and saving packets into pcap file is handled in the *Vpn service* block.

of how to create a VPN connected to remote server and a verbal comment saying in which part of the implementation should be packets captured and how to handle them. It inspired me for the first version of my application for which I used VPN connected to remote server. The approximate guidance given in this article can be used for both VPN usage and packet bypass usage.

The first article can lead to the idea that it would be possible to capture packets only within a mobile device, i.e. without a remote server. However, it does not specifically mention such a variant. Contrary to this, the second article [1] focuses more on bypass and capture of packets only within a mobile device. It provides more accurate information on how to implement the mentioned solution which has become crucial for packet capture in my application.

Figure 2 illustrates packet tunneling through interfaces designed by the author of the article [1]. The implementation blocks that are used for sending packets from a mobile device has violet color. The yellow ones are used for receiving them. Individual blocks used for packet handling inside the mobile device can be seen in the figure separated by a hatched line on the left. Conversely, blocks used for data sending to the desired destination are located on the right.

My application uses almost the same packet bypass as it is mentioned in article [1]. Thanks to this approach it uses secure way to catch packets and extract information from them. It's done with the help of VPN and virtual tunnels through which the packets pass. So my solution provides a secure way to pass data. Packets aren't sent to any server, so they cannot be caught on the way out of the mobile device. Another advantage is that I'm not using any suspicious server to pass packets through.

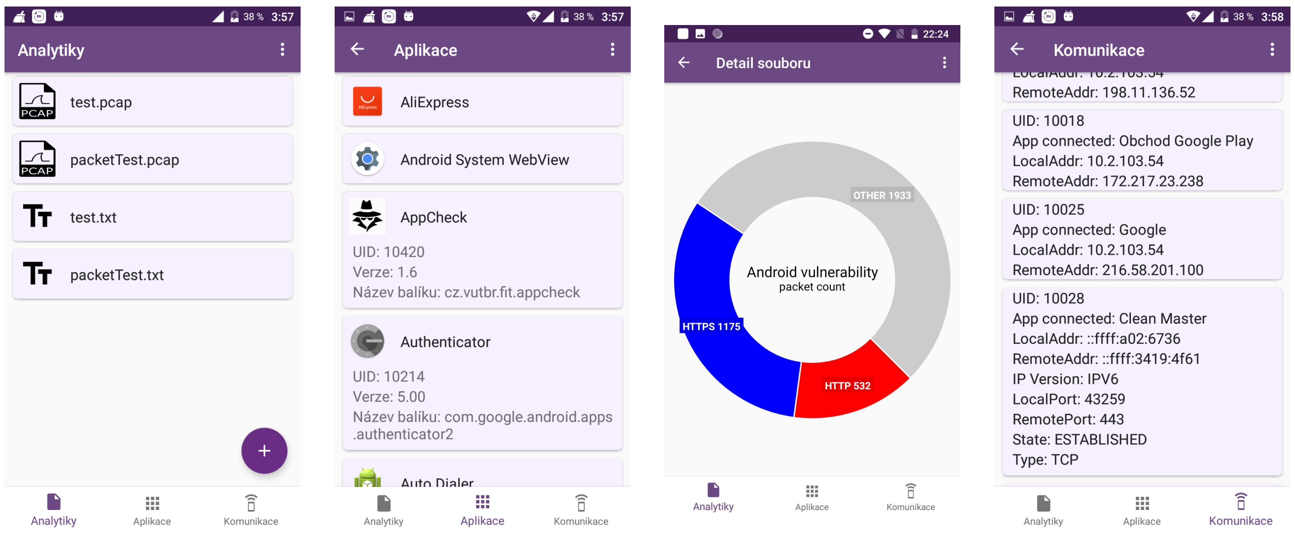
4. Implementation

In chapter 3 I mentioned two main articles that influenced my work. I tried to solve my first implementation with the help of VPN, as it is mentioned in the article by Mr. Sun [2]. The VPN was connected to a remote server running on Raspberry PI. This solution would be sufficient if I had my own server with enough capacity and secure connection. However, my implementation was unsuitable and unstable. The server did not have sufficient capacity for more than 3 users and the connection to it was insecure. For these reasons, it was necessary to find a different solution. Article [2] has brought me the idea of dealing with packet capture only within a mobile device. Thanks to it I found the second article, where the author is practically focusing on this kind of implementation. It doesn't provide any code example but it has properly subscribed packet flow diagrams.

As I already mentioned, my application uses a packet bypass. For this purpose, I have created a *Bypass Packet Capture* library that takes care of all the bypass logic. There are several packet handling blocks implemented in the library. The first is a VPN that provides communication between a locally created socket and a local tunnel. It is divided into two types depending on the protocol used - TCP and UDP tunnel. Servers, which communicate between the tunnel and the request queue (concurrent linked queue), are resolved in a similar way as tunnels. It means there are also 2 types. The last part is a remote tunnel that is connected to the concurrent linked queue. The entire data stream is send from/to the destination IP address through this remote tunnel.

The second library which I created for this application a *Pcap library*[6] hedges the work with pcap files.

[6]The *Pcap* library is publicly available at GitHub link: https://github.com/klepackovakarolina/pcaplibrary


Analytiky

test.pcap

packetTest.pcap

test.txt

packetTest.txt

Aplikace

AliExpress

Android System WebView

AppCheck
UID: 10420
Verze: 1.6
Název balíku: cz.vutbr.fit.appcheck

Authenticator
UID: 10214
Verze: 5.00
Název balíku: com.google.android.apps.authenticator2

Auto Dialer

Detail souboru

Android vulnerability
packet count

OTHER 1933
HTTPS 1175
HTTP 532

Komunikace

RemoteAddr: 198.11.136.52

UID: 10018
App connected: Obchod Google Play
LocalAddr: 10.2.103.54
RemoteAddr: 172.217.23.238

UID: 10025
App connected: Google
LocalAddr: 10.2.103.54
RemoteAddr: 216.58.201.100

UID: 10028
App connected: Clean Master
LocalAddr: ::ffff:a02:6736
RemoteAddr: ::ffff:3419:4f61
IP Version: IPV6
LocalPort: 43259
RemotePort: 443
State: ESTABLISHED
Type: TCP

Figure 4. Screenshots from AppCheck application.

It implements the necessary methods for creating pcap files from a byte stream that is read from a VPN in an endless cycle. Generally, the pcap file structure consists of a global 24-byte header, a pcap header of 16 bytes for each packet and their bodies. Before the packet forwarding is started, a pcap file only with global header is created in a local storage. Then in each iteration, a dummy L2 packet header is created via this library for each packet passing through the VPN. It does not carry any essential information but it's necessary for creating correct file structure. Then, the pcap header is appended to the particular packet and sent data are finally added.

The last step is creating the analysis of the recorded network flow from the saved file. I use the *Okio*[7] library to process individual packets written in the pcap file. It provides functionality such as: get packet's protocol, destination port, source port and many other. The implemented application currently provides information about the number of packets sent by HTTP or HTTPS protocol, or by completely different protocols such as UDP. Specifically, packets are counted based on the port to which they communicate. If it was sent to port 80 then it is recognized as a packet with HTTP protocol. If a particular packet was sent to port 443 then it is an HTTPS protocol. All others are marked as "other" in the resulting analysis. It is showed in the figure 4 in the 3rd picture from left. You can seen that during the recording of the communication, 532 packets were sent by HTTP, 1175 by HTTPS and 1933 by other protocols. This analysis was created by monitoring the flow of Chrome[8] application within 3 minutes.

Figure 3 shows the outbound packet flow (i.e. from a mobile device to the Internet) by blocks implemented in AppCheck. The individual sent packet flows in the direction shown by the arrows. The first block where the created local socket gets is VPN. Based on the transport protocol recorded in the header, it decides which local tunnel to use. However, before the packet is sent to the tunnel, a dummy L2 header and a pcap packet header are created for it. They are written into the resulting pcap file together with the contained data. It means that the packet capturing and saving into file is included in the *Vpn service* block. After writing to the file, the particular tunnel forwards the packet to the corresponding server. It remembers source and destination IP address information, ports on which it is communicated, and local tunnel mapping to a remote tunnel. Then it sends the request to the concurrent linked queue from which it is subsequently processed by the remote tunnel. Then it sends them to the original destination.

I also solve some other non-trivial issues in AppCheck, such as retrieving information from system files containing information about established network connections or mapping individual network connections to specific applications. Information about connections can be found in files stored in */proc/net/* directory which are divided by protocols they use , e.g */proc/net/tcp*, */proc/net/tcp6* and others. I map the individual connections to specific applications using it's uid, which can be obtained from *PackageManager* class in Android operating system.

[7]Available at: https://github.com/square/okio

[8]Available at: https://play.google.com/store/apps/details?id=com.android.chrome&hl=en

5. Conclusions

Currently, AppCheck[9] gives a comprehensive overview of installed applications. It provides information about their name, version, package name and icon. For network connections, it provides source and destination IP address, source and destination ports, IP version, transport protocol type, status, and especially the application to which the connection belongs. And last but not least it provides analyze of the number of packets sent by HTTP or HTTPS protocol, or by completely different protocols such as UDP.

In order to realize all the functionalities, I chose such ways that it is not necessary to require a higher level of authorization than the common user has. It means that it is not necessary to have root privileges. This approach is also supported by both the *Bypass Packet Capture* and *Pcap library*.

I tested the functionality of AppCheck on applications: 9gag[10], Chrome[11], Slack[12], AR measure[13], Remote desktop[14], IDOS[15], World Newspapers[16] and Network Scanner[17]. I have made a table 2 for a specific idea of what the outputs from the application look like. Each analyzes was made on a different application where the recording took about 1 minute.

This article should introduce readers to packet bypassing in a mobile device without using a remote server. It also provides partial information about packet capture and processing. The reader should get a comprehensive idea of the issue and its possible solution including an overview of available applications.

	HTTP	HTTPS	OTHER
9gag	129	1773	4128
Chrome (Excel@fit page)	18	240	375
Slack	0	850	874
AR measure	0	137	160
Remote desktop	31	37	931
IDOS	2	616	670
World Newspapers	807	482	1467
Network Scanner	7	66	73

Table 2. Table of analyze of individual applications. Each application was monitored for approximately 1 minute.

Improving my application could include providing packet content information. Or even disclosure of packets that have been transferred using the HTTPS protocol.

Acknowledgements

I would like to thank my supervisor prof. Ing. Adam Herout Ph.D for valuable advice.

References

[1] . Android capture tool development. blogpost (chinese), May 2018. https://www.jianshu.com/p/ae4d433597ce.

[2] Terrence Sun. Android VPN service explained with packet bypass example program. blogpost (english), Jun 2014. https://www.thegeekstuff.com/2014/06/android-vpn-service/.

[9] AppCheck is available at Google play: https://play.google.com/store/apps/details?id=cz.vutbr.fit.appcheck.

[10] Available at: https://play.google.com/store/apps/details?id=com.ninegag.android.app&hl=en

[11] Available at: https://play.google.com/store/apps/details?id=com.android.chrome&hl=en

[12] Available at: https://play.google.com/store/apps/details?id=com.Slack&hl=en

[13] Available at: https://play.google.com/store/apps/details?id=ml.kari.armeasure&hl=en

[14] Available at: https://play.google.com/store/apps/details?id=com.microsoft.rdc.android&hl=en

[15] Available at: https://play.google.com/store/apps/details?id=cz.mafra.jizdnirady&hl=en

[16] Available at: https://play.google.com/store/apps/details?id=com.KayaApps.newspapers&hl=en

[17] Available at: https://play.google.com/store/apps/details?id=com.easymobile.lan.scanner&hl=en