FREQUENCY BASED ADVERTISEMENT BLOCKING ON ANDROID MOBILE DEVICES USING A LOCAL VPN SERVER

A THESIS

Presented to the Department of Computer Engineering and Computer Science

California State University, Long Beach

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

Option in Computer Science

Committee Members:

Mehrdad Aliasgari, Ph.D. (Chair)
Burkhard Englert, Ph.D.
Neal Terrell, M.S.

College Designee:

Antonella Sciortino, Ph.D.

By Metehan Ozsoy

M.S., 2013, University of California, Irvine

December 2016

ABSTRACT

FREQUENCY BASED ADVERTISEMENT BLOCKING ON ANDROID MOBILE DEVICES USING A LOCAL VPN SERVER

By

Metehan Ozsoy

December 2016

Ads (advertisements) are the main economic source for most free web content. Modern ads are not only a large nuisance to end users but also an often violation of their privacy via tracking methods. There has been a rise in the use of ad blocking software. The major problem with these ad blocking software is that they rely on manually generated blacklists. In other words, humans need to detect ad URLs (Uniform Resource Locator) and add them to a blacklist so that it can be used by the ad blocking software. The purpose of this project is to design and implement an automated ad blocking software for Android mobile devices that does not rely on manually generated blacklists. The hypothesis to automate the generation of the blacklist is that URLs, which are not present in a given comprehensive whitelist and are visited more than a certain threshold number, are likely to be ad URLs.

In order to test the hypothesis, an Android mobile application is developed without requiring the root access on the device. The mobile application uses a local VPN (Virtual Private Network) server to capture the entire network traffic of the mobile device. The application is installed on a number of Android devices to collect the data to test the hypothesis.

The experiments illustrate that a false positive rate of 0.1% and a false negative rate of 0.26% can be achieved by an optimal frequency threshold number. It is concluded that the URLs that are not present in a given comprehensive whitelist and that are visited with higher frequencies are more likely to be ad URLs.

ACKNOWLEDGEMENTS

I thank my professor Dr.Mehrdad Aliasgari for encouraging me to undertake this project and for guiding me in overcoming the obstacles. I also want to thank Mr. Ramin Moazeni for helping me become a better Android developer.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Ads (advertisements) have become a large annoyance for users as they often interfere with desired content. In addition, the ad content providers collect personal information about users without their consent.

Although these privacy-violating ads are the main economic source for web content, they are highly undesirable among users. As a reaction to these unwanted ads, various methods for ad blocking have been developed. According to the PageFair and Adobe 2015 Ad Blocking Report [1], there were 198 million monthly active users for the major browser extensions that block ads. Furthermore, in 2015, the number of people using ad blocking software globally grew by 41%. As a result, the estimated loss of global revenue due to the blocked ads during 2015 was $21.8 billion US dollars.

In this project, the technical shortcomings of the modern ad blocking methods are presented. According to S.Bhagavatula et al. [2], the currently deployed solutions make use of regular expressions written by contributing users to target specific ads or ad placement schemes. They make use of a manual filter list that can be extremely specific and, thus, a page element or URL (Uniform Resource Locator) that is clearly ad-related may not be classified by the list because a certain substring is not found. Additionally, these filter lists, often called blacklists, are generated by third party advertisers. The lack of an automated process to generate these large blacklists results in a substantial human overhead.

In this project, a method of automated ad blocking on Android mobile devices is proposed and tested. A technique called local VPN (Virtual Private Network) server is used to capture the entire network traffic of the device and to enable the integration of the proposed network filtering rule. In computer networking, egress filtering refers to the act of monitoring and potentially restricting outbound data by means of preset firewall rules. In this project, the practice of egress filtering is extended as network filtering by also monitoring and filtering inbound data within the mobile application. Network filtering can serve various purposes such as ad blocking, spam filtering, and malware protection. Two of the most common ways of intercepting the network traffic are proxies and VPNs.

Proxies can be used to intercept HTTPS (HyperText Transfer Protocol over Secure Socket Layer) connection between the user and the destination address. They have the advantage of being able to track (and can control) the entire internet traffic of the device. Nevertheless, on a non-rooted Android device, a proxy can be configured only for a particular internet connection. Therefore, proxy is dependent on the internet connection in use. Furthermore, a proxy cannot be used with 4G (Fourth Generation of Wireless Mobile Telecommunications Technology) connection. Therefore, the use of proxies as the network interceptor is an incomplete solution within the scope of Android mobile devices.

Unlike a Proxy, a VPN is independent of the internet or 4G connection. The network traffic is encrypted, replacing the user's ISP (Internet Service Provider) and routing the network traffic through the VPN server, including all programs and applications. This will use all the benefits of the VPN servers such as speed, geographic location, and security. However, the security aspect of VPNs is of no interest to this project because it is not desired to hide the identity of the user as the VPN server is set up on the VPN client device itself.

A mechanism called local VPN server is introduced, particularly for Android mobile devices, that will enable users to apply their own network filtering rules. A local VPN server is a VPN server configured on an Android device that accepts only the device itself as the client. The local VPN server intercepts the network traffic of the device and potentially block the unwanted URL connections.

CHAPTER 2

BACKGROUND

Today's most popular ad blocking software products are browser-based ad blockers that run on a client's web browser. X. Dong et al. demonstrate that the ads get removed from the DOM (Document Object Model) tree in this approach [3]. Currently deployed browser-based ad blockers make use of regular expressions of keywords that would appear particularly in ad URLs. These regular expressions are then compiled into a list that is used by the web browser. This list is the list of blocked host domain names, simply referred to as *the blacklist*. Hence, the browser-based ad blocking solutions rely on manual blacklists; since they are browser-based, they cannot block the ads on the entire device.

Another problem with the browser-based ad blocking software is that they can have security issues. According to S. Bandhakavi et al. [4], determining whether extensions are malicious or harbor security vulnerabilities is a hard problem. Extensions are typically complex artifacts that interact with the browser in subtle and hard to understand ways.

B. Williams states that Adblock Plus is one of the most popular ad blocking software with over 100 million downloads [5]. It has a large list of blacklisted URLs and blacklisted keywords. Although having a high volume of users yields a high rate of blacklist feedback, content providers have become faster in disguising their ad URLs. Therefore, content providers have begun to combat Adblock Plus' rapidly adaptive manual blacklist. Furthermore, according to S. Bandhakavi et al. [4], the Adblock Plus extension performs the seemingly simple task of filtering out ads based on a list of ad servers. However, the Adblock Plus implementation consists of over 11,000 lines of JavaScript code. It is, therefore, criticized of being potentially vulnerable to security attacks via JavaScript code injection.

Some of the recently developed ad blocking software protect the user's privacy by preventing content providers from tracking users. One of the privacy protecting ad blockers is Ad Muncher [6]. Ad Muncher has a dedicated group of engineers, who work on compiling the world's most comprehensive filter list to deal with the most complex advertising systems and unwanted content on the net [6]. An anonymous proxy server feature is included to help protect the user's identity. Ad Muncher became a free software in 2014. Although it has been promised to users that their personal information is not stored, sold, rented, or traded to any third party under any conditions, the application reserves the right to make changes to the policy.

Not all ad blocking software completely block ads. Since web advertising is profitable, there have been few attempts in developing new business models that aim to protect users' privacy while sharing the ad revenue. Brave is such browser-based ad blocking software, which is in the beta testing phase, that differs from other ad blockers in its business model [7]. Like Ad Muncher, Brave promises to protect user privacy. Oftentimes, ads are not removed completely, but they are replaced with new ads. The revenue of these ads is split between publishers, users, network partners, and the company itself. With Brave, websites can maximize their profit by cutting out as many middlemen as possible.

Although ad blocking software have been of great interest among users, they have some dramatic shortcomings. Some websites detect the ad block add-ons on the user's browser and deny access. In fact, the practice of *blocking the ad block add-on* have become very popular among content providers. In addition, the advertisement URLs have become much harder to detect as the content providers change the domain and URL names frequently and avoid using blacklisted keywords such as *ad* and *popup* in order to disguise their ads. Due to these shortcomings of browser-based ad blocking software, there rose a need for a more general solution that can combat with ads.

An alternative approach to the browser-based ad blocking is the use of an external VPN server to monitor and filter network traffic of the user. In this method, the ad blocking software is implemented on a VPN server and the user is the VPN client. The user allows all of his network traffic to be routed to the VPN server and gives the permission to block the connection with certain URLs.

Figure 1 demonstrates how a VPN server can be used for network filtering. In the figure, the three components of the network filtering via an external VPN server are exhibited. These components are the user, the VPN server and the internet. The request and receive relationships between the entities are demonstrated by the arrows between them. It is also illustrated that the practice of network filtering takes place within the scope of the VPN server.

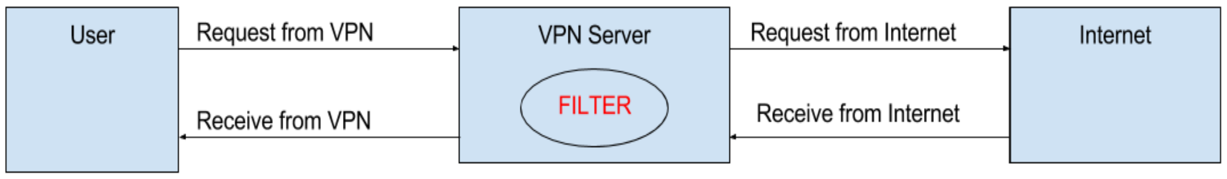

FIGURE 1. Network filtering via an external VPN server.

The external VPN method has few drawbacks. First of all, the entire network traffic of the user is routed to an external server that can be an adversary. The user has to trust that the VPN service provider will secure the user's connection, keep it encrypted, and protect it from prying eyes. Nevertheless, not all VPN service providers are trustworthy. Some of them diligently log clients' connection times, dates, and IP (Internet Protocol) addresses. A VPN server can even track the types of traffic that the user sends through their networks. Furthermore, it is likely that the connection speed will be affected adversely since the user makes an additional connection before actually connecting to the target website.

Mobile phone users are exposed to annoying ads not only on websites but also in mobile applications. Few network filtering Android applications have been developed that claim to block ads on the web browser and in the applications. Android allows the ad blocking software developer to set up a local VPN server on the mobile device that captures the network traffic of the device and controls which network packets to be sent to their destinations or which packets to be received. A local VPN server uses Android's native VpnService library to configure a VPN tunnel [8]. Since network filtering rules on Android mobile devices are desired to be not dependent to any external server, the end point of the VPN tunnel is defined as the localhost, or the device itself. In this method, users do not give away their network traffic to an external source.

One of the web filter applications for Android is Adguard [9], which uses the local VPN technique. Adguard protects the user from annoying advertising, malicious and phishing websites, and online tracking [9]. Its VPN technology allows filtering the traffic on the device without root privileges. Adguard, however, is not open source and, thus, it does not provide the automation of firewall rules on the device. Adguard uses a manual blacklist of URLs for ad blocking. Although the user can arbitrarily extend this blacklist, no other method of ad blocking could be used. This is a limiting property of Adguard. This project extends the local VPN solution used in Adguard to devise an open source framework that allows the user to implement their own method of ad blocking.

The local VPN server on the Android mobile device is used to practice network monitoring and filtering, and, in particular, ad blocking. Furthermore, websites cannot detect and block the network level ad blocking software.

CHAPTER 3

PROPOSED METHOD

A method for ad blocking on Android devices is presented in this section. The method consists of two major components that are capturing the URLs from users' Android mobile phones and classifying ad URLs. It is worth to mention that the method works on non-rooted Android devices as well as rooted devices.

The technique used to capture URLs on Android devices is similar to that of Adguard, which is explained in the previous chapter. However, in addition to the technique used in Adguard, the proposed method captures the frequency of each URL on the mobile device. The frequency of a URL is simply defined as the number of times an IP packet was sent or received from that URL. A local VPN server is set up on the mobile device in order to monitor and filter the network traffic of the device. The Android application is designed to log the list of URLs and their frequencies. Then, the URLs and their frequencies are deployed to a central server. The URLs are inserted into the central database on the server with a timestamp of the format *yyyy-mm-dd*.

In order to better understand the architecture of the database, Table 1 is provided as an example. The table has three columns, which are URL, FREQUENCY, and TIMESTAMP. It is observed from the table that a total of 52715 IP packets are sent and received from 'cms.analytics.yahoo.com' on 2016-09-08, which makes it the most frequent URL on 2016-09-08. Since the rank of a URL is determined by its frequency, the URLs are ordered by their frequencies in decreasing order. It is also worth noting that the URLs in the table are the full internet addresses. For example, in the third and fourth rows of the table, the URLs 'advertising.couchsurfing.com' and 'login.couchsurfing.com' are stored separately, although they share the same domain space ‘couchsurfing.com,’ because they have separate subdomains. Logging the full internet addresses makes it possible to distinguish between ad URLs and non-ad URLs. For instance, ‘advertising.couchsurfing.com’ is clearly an ad URL while ‘login.couchsurfing.com’ is not an ad URL.

TABLE 1. Database Table of URLs with Their Frequencies and Timestamps

URL	**FREQUENCY**	**TIMESTAMP**
cms.analytics.yahoo.com	52715	2016-09-08
api.lyft.com	51240	2016-09-08
advertising.couchsurfing.com	32332	2016-09-08
login.couchsurfing.com	20204	2016-09-08
stats.pagefair.com	2106	2016-09-08

An automated procedure is engineered on the server to classify ad URLs. This automated procedure of classifying ad URLs heavily relies on the hypothesis that the ad URLs have relatively higher frequencies compared to non-ad URLs. It is worthwhile to mention that the Android application is designed to protect user privacy as the URLs are anonymously logged into the server. The rest of this section explains the steps of the ad URL classification system.

At the core of the proposed solution, there is a comprehensive whitelist of URLs, which is ordered by the frequencies of the URLs in the list. The set of whitelisted URLs, which is the list of URLs that are known to be non-ad URLs, is denoted by *W.* By using this whitelist, it is proposed that the URLs that are not whitelisted and that have higher frequencies are more likely to be ad URLs.

There are two parameters used in the proposed method. The first parameter is the *frequency threshold* number. A URL is inserted to the blacklist if its time-aggregated frequency is greater than or equal to the frequency threshold number. The second parameter of the method is the *time window size*. The URLs and their frequencies are logged daily into the database. When calculating the time-aggregated frequency of a URL, the daily frequencies of this URL are aggregated (summed) over the time period determined by the time window size. The time window size parameter enables the ad blocking software to be adaptive of the newest ad URLs. Content providers change the ad URLs within time and, therefore, the blacklist of ad URLs needs to be able to catch these changes. Also, having a time window size parameter allows the solution to ignore URLs that are outdated or simply not used anymore.

Suppose that $U_t = \{u_1^t, u_2^t, \dots, u_n^t\}$ is the finite set of URLs in the central database logged from the Android clients' network traffic on the day *t.* The *frequency* of a URL u_i^t is denoted by f_i^t for $n \geq i \geq 1$. Let *T* be a fixed positive integer that represents the time window size of the application. Given a timestamp *t*, the set $\overline{U_t}$ is defined as the time-aggregated set of URLs over the time period $[t - T, t]$ and is given by

$$\overline{U_t} = \bigcup_{t-T \leq j \leq t} U_j \tag{1}$$

or equivalently,

$$\overline{U_t} = \left\{u \in U_j \;\middle|\; t - T \leq j \leq t\right\} \tag{2}$$

Equation (1) and Equation (2) illustrate that the time-aggregated set of URLs over a time period is the union of all the URLs within that time period.

The *time-aggregated frequency* of a URL $\overline{u_a^t} \in \overline{U_t}$ over the time period $[t - T, t]$ is denoted by $\overline{f_a^t}$ and is defined by

$$\overline{f_a^t} = \sum_{t-T \leq j \leq t} f_a^j \tag{3}$$

Equation (3) states that the time-aggregated frequency of a URL over a time period is the sum of its daily frequencies within that time period.

Let δ be a fixed frequency threshold number. The set of URLs with frequencies greater than or equal to δ is given by

$$\aleph_t = \{\overline{u_a^t} \in \overline{U_t} \mid \overline{f_a^t} \geq \delta\} \tag{4}$$

It is hypothesized that the set $B_t = \aleph_t / W$ is a blacklist, a list of ad URLs, on the day *t*.

In the Experiments and Results section, different values for the frequency threshold number are tested and the results are presented.

CHAPTER 4

IMPLEMENTATION

The implementation of the proposed method consists of three major components, which are the Android mobile application, the REST (Representational State Transfer) server, and the database. (The database is contained on the REST server.) In this section, the implementation of each component is explained.

The Android Mobile Application

The Android mobile application is the client side of the system. The name of the Android mobile application is FilterEagle. FilterEagle consists of three major components, which are the local VPN server, user authentication, and URL logging.

Local VPN Server on Android

Building a VPN server on an Android device requires access to the network layer, layer 3 of the OSI (Open Systems Interconnection) model. The network layer is responsible for packet forwarding including routing through intermediate routers, since it knows the address of neighboring network nodes, and it also manages QoS (Quality of Service), and recognizes and forwards local host domain messages to the transport layer, which is layer 4. When developing an Android application, the developer is typically on the application layer, which is the layer 7 of the OSI model. In order to have access to the network layer, Android provides a native library called VpnService [8]. VpnService is a background service in Android, therefore it does not interfere with the user experience.

The work flow of the local VPN server is as follows:

1. A VPN tunnel is configured using the Builder (file descriptor) class.

2. In order to read and write packets, an input stream and an output stream are created for the VPN tunnel.
3. Outgoing IP packets are read into the input stream of the VPN tunnel.
4. The URL from the header of the IP packet is extracted and logged.
5. If the URL is in the blocked host list of the application, this IP packet is not processed further. If the URL is not blocked, the IP packet is converted to a TCP (Transmission Control Protocol) packet or a UDP (User Datagram Protocol) packet depending on the protocol extracted from its header.
6. A *Socket* object or a *DatagramSocket* is created and protected for the TCP and UDP packets, respectively [10].
7. The packet is sent to its destination using the newly generated socket object and the response is listened.
8. The response is received via the listening socket object.
9. The incoming TCP or UDP packet is converted to an IP packet.
10. The URL from the header of the IP packet is extracted and logged.
11. If the URL is in the blocked host list of the application, this IP packet is not processed further. If IP packet is not filtered, it is written into the output stream of the VPN tunnel.

User Authentication

A secret application key is stored on the mobile device when FilterEagle is installed on a mobile device for the first time. This secret key is used for the authentication of the application. During the registration of a new user, a random username is generated. FilterEagle has a switch in the settings menu, which is called ‘Automated Adblock.’ When this switch is turned on, the user is automatically logged in to the system with the username.

URL Logging

Once the user turns on the ‘Automated Adblock’ switch, the application starts logging all the URLs from incoming and outgoing TCP and UDP packets. A hash map of URL-Frequency is used to store the URLs and their frequencies. The URL-Frequency hash map is periodically deployed to the REST server, where the period is determined by the application developer. The query to the server that forwards this hash map includes the login information, the username, and the secret application key. This ensures that unauthorized users, or simply anyone who does not have FilterEagle properly installed on their phone, do not have access to the server and, therefore, they cannot forward any data.

The REST Server

The name of the REST server is AdList. AdList is developed in Eclipse IDE (Integrated Development Environment) using Java with maven repository. AdList is deployed on to Heroku, a cloud PaaS (Platform-as-a-Service), as a dyno, a lightweight linux container that runs a single user-specified command web application. On the same server, a database called AdList DB is built. AdList DB is a Postgres database, which is an object-relational database management system with an emphasis on extensibility and standards-compliance. AdList DB has two tables, which are *users* and *urls*.

The users table holds the usernames of the registered users of FilterEagle. This table remains private as it stores the necessary information for user authentication between FilterEagle and AdList. Table 2 provides an example of an instance of the users table. In this table, three users are registered. The first column of the table represents the randomly generated usernames. The second column of the table is the timestamp of the registration dates of the users in the format *yyyy-mm-dd.* The variable type of the usernames is an array of characters with variable length. The variable type of the registration dates is the native date type in SQL (Structured Query Language).

TABLE 2. Users Table on the REST Server

Username	Registration_Date
u2343j9f3kf230fnn9224n3204n	2016-08-07
n478923ndf03njfspdof-03fmdf	2016-08-11
kie39j3223irnf2903jf923jf23ff	2016-09-01

The *urls* table holds the list of URLs forwarded from FilterEagle. This table is public, and can be viewed at a public website. The most up-to-date list of ad URLs, which are generated by AdList, can be downloaded at this public website.

The process of generation of the URLs in the *urls* table is illustrated in Figure 2. In this figure, the three components of the system are represented in boxes, which are titled 'Android Mobile Application,' 'REST Server,' and 'Database.' The arrows between the Android mobile application and the REST server represent the relationship between these two components. It is illustrated that the URLs are logged by the Android mobile application, and they are filtered in the REST server. Then the filtered URLs are inserted into the database.

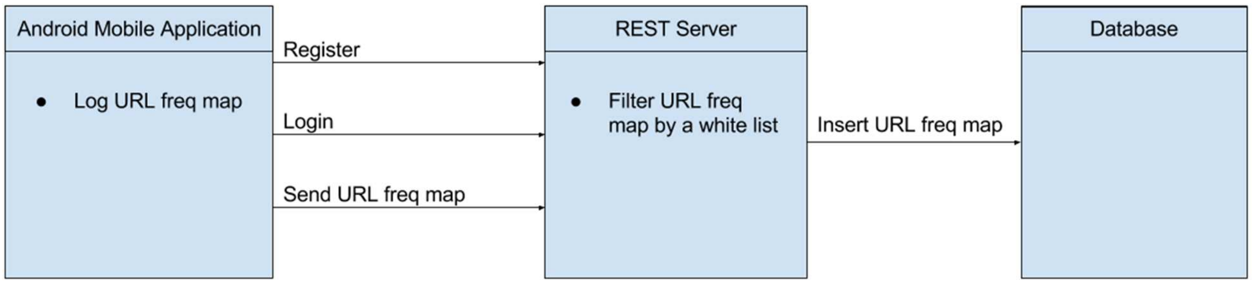

FIGURE 2. The main components of the automated ad blocking implementation.

CHAPTER 5

EXPERIMENTS AND RESULTS

It is hypothesized that URLs that are not present in a given comprehensive whitelist and that have a frequency higher than a certain threshold number are likely to be ad URLs. To help collect the data to be tested, a number of voluntary Android users contributed to this project by installing FilterEagle on their mobile phones. Note that since one of the critical ideas in the proposed method involves the frequencies of the URLs collected, it is significant that the URLs come from a diverse set of users. Having a diverse set of users helps the data to reflect the real world data and enables the experiments to provide more accurate results. Therefore, FilterEagle was run on its users' mobile phones for two months.

Once the URLs are inserted into the *urls* table in the database, they were filtered in several steps to generate the final blacklist. These steps are whitelist filtering, large domain filtering, and frequency threshold filtering.

Whitelist Filtering

As explained in the preceding section, a comprehensive whitelist is used to remove the whitelisted URLs from the URLs collected by FilterEagle. In the experiments, two sources are used as whitelists. These whitelists can be downloaded at 'Anthologeek's Top1000' and at 'Alexa's Top One Million.' The server of this project, AdList, has a Boolean flag for each whitelist that determines whether that specific whitelist is used or not.

'Anthologeek's Top1000' is updated weekly and 'Alexa's Top One Million' is updated daily. The updates are programmatically incorporated into the server. Therefore, the cumulative whitelist is defined as the union of 'Anthologeek's Top1000' and 'Alexa's Top One Million.' This cumulative whitelist is updated at least daily on the server.

Large Domain Filtering

It is observed that there are URL domains with a large number of subdomains. These large domains are mostly dynamic non-ad URLs, and they are not entirely captured in the whitelist filtering phase. The large URL domains that FilterEagle detected are Facebook, Google, Amazon, and Twitter. If a URL is in the subdomain of these four large domains and if it does not contain the keyword *ad*, it is removed from AdList's URLs list.

Frequency Threshold Filtering

As a final step in generating the blacklist, URLs that have a frequency of less than a threshold number are also removed from AdList's URLs list. The frequency threshold filtering is accomplished using two alternative methods that are the standard score and the log score. Suppose that $\{u_1, u_2, \dots, u_n\}$ is the finite set of URLs in AdList after the whitelist and large domain filtering. The *frequency* of a URL u_i is denoted by f_i for $n \geq i \geq 1$.

Method 1

The mean of the frequencies f_i where $n \geq i \geq 1$ is given by

$$\bar{f} = \frac{1}{n}\sum_{i=1}^{n} f_i \tag{5}$$

Equation (5) states that the mean of the frequencies is the sum of the frequencies divided by the number of the frequencies.

The standard deviation of the frequencies f_i where $n \geq i \geq 1$ is given by

$$\sigma_f = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f_i - \bar{f})^2} \tag{6}$$

Equation (6) is the formulation of the amount of the dispersion of the frequencies. The standard deviation of the frequencies is calculated by using Equation (6).

The standard score of a URL u_i where $n \geq i \geq 1$ is given by

$$S(u_i) = \frac{f_i - \bar{f}}{\sigma_f} \tag{7}$$

Equation (7) suggests that the method of the standard score ranks a URL by how much the frequency of this URL differs from the average frequency of all URLs. This is because the denominator in Equation (7) is the standard deviation σ_f of the URLs that is a fixed number over a time period, and, therefore, it does not affect the ranking of the frequencies.

Method 2

Note that most TCP connections will consist of an exchange of multiple TCP packets. This will cause the frequency of a URL to be increased by the number of TCP packets sent and received. In order to eliminate the effect of this duplication of URLs, the logarithm of each frequency is taken. The *log frequency* of a URL u_i is defined as $\log(f_i)$ where the logarithm is taken base 2. Then, the *log score* of a URL u_i is denoted by $LS(u_i)$ and is given by

$$LS(u_i) = \frac{\log(f_i)}{\mu} \tag{8}$$

where μ is the mean of the log frequencies and is given by

$$\mu = \frac{1}{n} \sum_{i=1}^{n} \log(f_i) \tag{9}$$

Equation (9) illustrates that the mean of the log frequencies is the sum of the log frequencies divided by the number of the frequencies.

Equation (8) states that the log score of a URL is the logarithm (base 2) of its frequency divided by the mean of the log frequencies. The equation suggests that the log score ranks the URLs by their log frequencies. This is because the denominator in Equation (8) is the mean of the log frequencies, which is a fixed number over a time period, and, therefore, it does not affect the ranking of the URLs.

It is worth noting that method 2 preserves the order of the frequencies. In other words, it is implied by Equation (8) that

$$f_1 \geq f_2 \geq \cdots \geq f_n \xRightarrow{implies} LS(u_1) \geq LS(u_2) \geq \cdots \geq LS(u_n) \tag{10}$$

Results

In order to test the performance of the application, an up-to-date true blacklist called ‘Easy List’ is used and can be downloaded. This true blacklist of URLs is denoted by B. In order to enhance the correctness of the experiment, a true blacklist keyword set that has the words ‘ad,’ ‘popup,’ ‘slider,’ ‘click,’ and ‘lytics’ is used. The set of URLs in the entire URL universe that contains a true blacklist keyword is denoted by *K*. It is assumed that any URL containing a true blacklist keyword is an ad URL. Therefore, any URL in the set K is an ad URL.

The set of URLs in AdList before the frequency threshold filtering is denoted by A_{before}.

The set of URLs in AdList after the frequency threshold filtering is denoted by A_{final}.

In the performance testing, two statistical measures are used, which are the false positive rate (FP) and the false negative rate (FN).

The false positive rate is defined as

$$FP = \frac{|A_{final} \setminus (B \cup K)|}{|A_{final}|} \tag{11}$$

Equation (11) states that the FP is simply the percentage of the URLs that are classified as ad URLs but are not ad URLs in reality.

The false negative rate is defined as

$$FN = \frac{|(A_{before} \setminus A_{final}) \cap (B \cup K)|}{|A_{before} \setminus A_{final}|} \tag{12}$$

Equation (12) states that the FN is simply the percentage of the URLs that are not classified as ad URLs but are ad URLs in reality.

The experiments are run for different frequency threshold numbers by using the two frequency scoring methods. An optimal frequency threshold number minimizes both the FP and FN rates. In other words, it is desired to catch a maximum number of ad URLs and a minimum number of non-ad URLs.

Table 3 exhibits the performance analysis of the application when the standard frequency scores are used. The first column of the table shows the different values of the frequency threshold number, which is the only parameter in this experiment, in increasing order. The second and third columns are the total number of URLs in AdList before and after the frequency threshold filtering, respectively. The last two columns, FP and FN, represent the false positive and false negative rates, respectively, which are obtained by the corresponding frequency threshold number. A total of 16 different frequency threshold numbers in the range [-0.6, 1.6] are tested.

It is observed that, when the threshold number is less than 1.0, there is a negative correlation between the frequency threshold and the FP. According to Table 3, AdList classifies ad URLs with more accuracy as the frequency threshold is increased. In addition, there is a consistent negative correlation between the frequency threshold and the FN.

An analogous experiment is run using the log frequency scores. Table 4 illustrates the results of the experiments using the log frequency scores. Table 4 has the same structure and definitions as in Table 3. They differ only in the way that the frequencies are scored. A total of 17 different frequency threshold numbers in the range [0.70, 1.50] are tested. Since the standard score and the log score rank the URLs differently, different frequency threshold numbers are tested for the two scoring methods.

TABLE 3. Experiment Results Using the Standard Frequency Scores

Standard Frequency Threshold	$\lvert A_{before} \rvert$	$\lvert A_{final} \rvert$	FP	FN
-0.6	1445	1217	0.18	0.95
-0.5	1445	924	0.19	0.41
-0.4	1445	733	0.18	0.30
-0.3	1445	639	0.17	0.27
-0.2	1445	534	0.15	0.24
-0.1	1445	459	0.12	0.22
0.0	1445	396	0.12	0.21
0.1	1445	343	0.11	0.20
0.2	1445	321	0.11	0.19
0.3	1445	305	0.11	0.19
0.5	1445	274	0.11	0.18
0.8	1445	194	0.11	0.17
1.0	1445	152	0.12	0.17
1.2	1445	119	0.12	0.16
1.4	1445	104	0.13	0.16
1.6	1445	89	0.15	0.16

TABLE 4. Experiment Results Using the Log Frequency Scores

Log Frequency Threshold	$\lvert A_{before} \rvert$	$\lvert A_{final} \rvert$	FP	FN
0.70	1445	1127	0.001	0.63
0.75	1445	1069	0.01	0.62
0.80	1445	968	0.02	0.45
0.85	1445	848	0.01	0.36
0.90	1445	781	0.02	0.33
0.95	1445	712	0.02	0.29
1.00	1445	606	0.02	0.26
1.05	1445	531	0.05	0.23
1.10	1445	460	0.07	0.21
1.15	1445	400	0.10	0.21
1.20	1445	303	0.19	0.19
1.25	1445	295	0.19	0.19
1.30	1445	281	0.20	0.19
1.35	1445	249	0.22	0.18
1.40	1445	219	0.23	0.18
1.45	1445	208	0.25	0.17
1.50	1445	203	0.25	0.17

In Table 3, it is observed that the minimum FP that can be achieved is 11%. When the minimal FP of 11% is achieved, the minimum FN possible is 17%. Hence, the optimal frequency threshold, when the standard frequency scoring scheme is used, is 0.8. The results in Table 3 show that AdList classifies ad URLs with an accuracy of 11% when the frequency threshold is 0.8. Furthermore, for this frequency threshold, 17% of the URLs left out are ad URLs.

Table 4 illustrates that, when the log frequency scoring scheme is used, the minimum FP that can be achieved is 0.001%, and the minimum FN possible is 17%. This means that the log frequency score is a more accurate ad URL ranking method than the standard frequency score in classifying ad URLs.

The experiment results in Table 4 exhibit a negative correlation between the frequency threshold and the FN. On the other hand, there is a positive correlation between the frequency threshold and the FP. Hence, the optimal frequency threshold cannot minimize both the FP and FN. Therefore, a frequency threshold is chosen so that it balances out a small FP and a small FN. Table 4 illustrates that such optimal frequency threshold is 1.00. When the frequency threshold is 1.00, the FP is 0.02% while the FN is 26%.

A small FP indicates that the most visited ad URLs are classified accurately. On the other hand, a large FN indicates that a portion of the least visited ad URLs are not captured. The results in Table 3 and Table 4 suggest that the FPs are significantly less than the FNs. Therefore, AdList manages to classify the most visited ad URLs while it places less emphasis on capturing the least visited ad URLs.

CHAPTER 6

CONCLUSION

The purpose of this project is to design and implement an automated ad (advertisement) blocking software for Android mobile devices that does not rely on manually generated blacklists. The hypothesis to automate the generation of the blacklist consists of URLs (Uniform Resource Locator) that are not present in a given comprehensive whitelist and are visited more than a certain threshold number are likely to be ad URLs. In order to test this hypothesis, a system is built that consists of a client side and a server side.

The client side is an Android mobile application. The mobile application uses a local VPN (Virtual Private Network) server implementation to intercept the network traffic of the mobile device. The URLs are stored together with their hit rates (number of times visited) and daily timestamps. These stored URLs are periodically deployed to a central database, which is implemented on the server side, to auto-generate the blacklist.

The server side is a REST (Representational State Transfer) server application implemented in Java. The server periodically receives the URLs from the Android clients, and stores them into a database. Among the URLs in the database, a number of them are ad URLs. The hypothesis is used to classify the ad URLs among all the URLs in the database. This classification is achieved by implementing a whitelist filtering, a large domain filtering, and a frequency threshold filtering.

The client side and the server side applications are run for a two-month time period. As a result, a blacklist is auto-generated on the server side. The auto-generated blacklist depends on the frequency threshold parameter, which is the minimum of number of times a URL is visited within a time period. This auto-generated blacklist is then compared to a true blacklist in order to test the accuracy of the method.

The results of the experiments are summarized as illustrated in Table 5. It is observed that, when the standard scores are used, the optimal frequency threshold is 0.8. This optimal frequency threshold results in an FN (false negative) of 17% and an FP (false positive) of 11%. When the log scores are used, the optimal frequency threshold is 1.0. This optimal frequency threshold results in an FN of 23% and an FP of 2%.

TABLE 5. Summary of Experiment Results

	Standard Score	Log Score
Optimal Frequency Threshold	0.8	1.0
False Negative Rate	17%	23%
False Positive Rate	11%	2%

An FP measures the accuracy of the classification of the ad URLs. Hence, a small FP indicates that the most frequently visited ad URLs are accurately classified. On the other hand, an FN measures the accuracy of the classification of the non-ad URLs. Hence, a large FN indicates that a portion of the least visited ad URLs are not captured. Table 5 illustrates that the FPs are significantly less than the FNs. Therefore, the most frequently visited ad URLs are classified more accurately than the least visited ad URLs.

Future Work

A future work can include an improvement of the URL data gathered in the server to better realize the real-world online advertising trends. In order to enhance the data, the client side application, which is the Android mobile application, can be installed and used by a larger set of

Android users. Furthermore, by having a more diverse set of Android users, the proposed method can result in better FPs.

It is significant that the classification of the ad URLs heavily depends on the URL frequency scoring method. In this project, two frequency scoring methods are used, which are the standard score and the log score. In the future, the proposed method can be implemented by using different frequency scoring methods. When using different frequency scoring methods, the new experiment results should be compared to the results in Table 5. In the most ideal case, the FP and the FN are both desired to be close to zero.

REFERENCES

REFERENCES

[1] PageFair and Adobe, “The 2015 Ad Blocking Report,” Aug. 2015; https://pagefair.com/blog-/2015/ad-blocking-report/.

[2] S. Bhagavatula, C. Dunn, C. Kanich, M. Gupta, and B. Ziebart, “Leveraging Machine Learning to Improve Unwanted Resource Filtering,” presented at the Grace Hopper Celebration of Women in Computing Poster Session, 2014.

[3] X. Dong, M. Tran, Z. Liang, and X. Jiang, “AdSentry: Comprehensive and Flexible Confinement of JavaScript-based Advertisements,” *Proc. Annual Computer Security Applications Conference* (ACSAC11), 2011, pp. 297–306.

[4] S. Bandhakavi, S. T. King, P. Madhusudan, and M. Winslett, “VEX: Vetting Browser Extensions for Security Vulnerabilities,” presented at the USENIX Security Symposium, 2010.

[5] B. Williams, “Adblock Plus and A Little More,” 2016; https://adblockplus.org/blog/100-million-users-100-million-thank-yous.

[6] Ad Muncher, “Ad Muncher: Blocks Ads in All Browsers,” 2016; https://admuncher.com.

[7] Brave, “Brave: Browse Faster,” n.d.; https://brave.com/#faster.

[8] Android, “Android: Android VpnService Library,” 2016; https://developer.android.com/reference/android/net/VpnService.html.

[9] Adguard, “Adguard: The World’s Most Advanced Ad Blocker,” 2016; https://adguard.com.

[10] R. Buyya, “Socket Programming,” *Object Oriented Programming with Java: Essentials and Applications.* McGraw Hill Education, 2009, pp. 367-381.