TrafficAV: An Effective and Explainable Detection of Mobile Malware Behavior Using Network Traffic

Shanshan Wang[†‡] , Zhenxiang Chen[*†‡], Lei Zhang[†‡], Qiben Yan[§], Bo Yang[‡], Lizhi Peng[†‡], Zhongtian Jia[†‡]
[†]School of information science and engineering, University of Jinan, Jinan, China, 250022
[‡]Shandong Provincial Key Laboratory of Network Based Intelligent Computing, Jinan,China 250022
[§]University of Nebraska Lincoln, Lincoln, NE, USA, 68588
[*]Corresponding author, Email: czx@ujn.edu.cn

***Abstract*—Android has become the most popular mobile platform due to its openness and flexibility. Meanwhile, it has also become the main target of massive mobile malware. This phenomenon drives a pressing need for malware detection. In this paper, we propose TrafficAV, which is an effective and explainable detection of mobile malware behavior using network traffic. Network traffic generated by mobile app is mirrored from the wireless access point to the server for data analysis. All data analysis and malware detection are performed on the server side, which consumes minimum resources on mobile devices without affecting the user experience. Due to the difficulty in identifying disparate malicious behaviors of malware from the network traffic, TrafficAV performs a multi-level network traffic analysis, gathering as many features of network traffic as necessary. The proposed method combines network traffic analysis with machine learning algorithm (C4.5 decision tree) that is capable of identifying Android malware with high accuracy. In an evaluation with 8,312 benign apps and 5,560 malware samples, TCP flow detection model and HTTP detection model all perform well and achieve detection rates of 98.16% and 99.65%, respectively. In addition, for the benefit of user, TrafficAV not only displays the final detection results, but also analyzes the behind-the-curtain reason of malicious results. This allows users to further investigate each feature's contribution in the final result, and to grasp the insights behind the final decision.**

I. Introduction

By the end of 2015, the number of smart phones is likely to be exceeding the number of human beings, and by 2016 there could be 10 billion smart phones around the world [1]. As for the mobile operation system, Android operation system occupies a top market share. However, the uprising of the Android system is greatly impaired by the prevalent Android malware. It is reported that 90% of the 126 apps tested faced at least two vital security vulnerabilities [2]. This frightening statistic reveals the urgency on enforcing mobile app security.

Malicious apps utilize multiple methods to evade the existing detection mechanisms provided by Android operating system or existing anti-virus software. These evasion methods include dynamic execution, code obfuscation, repackaging or encryption [3]. Sophisticated malware developers implement powerful encryption or obfuscation techniques to make their malicious activities concealed within the huge amount of network traffic. However, the network behaviors of malware can still present non-trivial anomalies that can be identified by advanced detectors, which provides us with a keen insight in malware detection.

We propose TrafficAV, an effective and explainable malware identification and classification method. TrafficAV exploits network traffic to detect mobile malware, since almost all malicious behaviors of malware are accomplished through the network interface. TrafficAV employs a traffic mirroring technology to collect network traffic generated by mobile apps, and the generated network traffic is transmitted to a server for data analysis. On the server side, TrafficAV extracts traffic features, and then uses detection models based on machine learning to detect whether the app is malicious or not. For the benefit of user, TrafficAV adds another meaningful functionality that not only displays the final detection results, but also analyzes the reason behind these malicious observations.

There are two detection models in TrafficAV, namely HTTP detection model and TCP flow detection model. We handle HTTP traffic because HTTP protocol is the most preferred protocol for the majority of mobile apps. HTTP packets carry plenty of important information to classify network traffic. However, it becomes difficult to get valuable information from HTTP traffic when the HTTP traffic generated by mobile apps is encrypted. So we have designed TCP flow detection model to compensate for the lack of HTTP model. We choose to analyze TCP, because TCP is one of the most popular transport layer protocols.

In this paper, we make the following contributions to detect Android malware:

- **Network traffic based effective mobile malware detection.** Our experimental results show that combining network traffic features with machine learning algorithms can effectively identify malicious behavior.
- **Multi-level detection.** Multiple levels of network traffic features are analyzed and the final results prove that both HTTP packet (detection rate of 99.65%)and TCP flow (detection rate of 98.16%) can effectively identify malware.
- **User-friendly result explanation.** For the benefit of user, TrafficAV shows detection results and simultaneously creates a scoring mechanism to give the reasons for making such decision.

The rest of this paper is organized as follows: Related work is introduced in Section II. Section III introduces the methodology of TrafficAV in detail. The evaluation of TrafficAV is

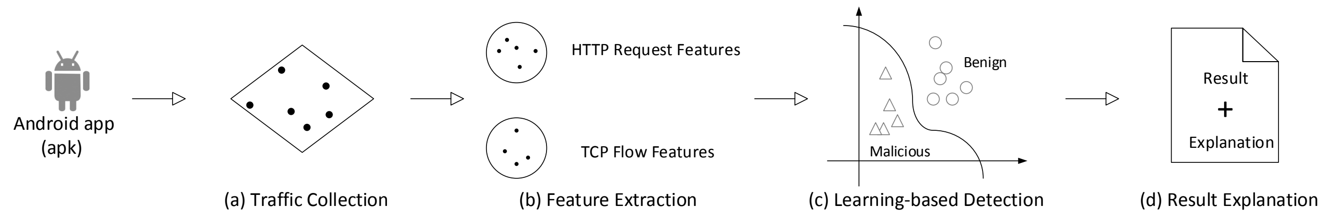


Fig. 1: Schematic depiction of the analysis steps performed by TrafficAV

introduced in section IV and section VI concludes the paper.

II. Related Work

Many security mechanisms have been proposed to detect malicious Android apps and to protect targets from being attacked. Most of the proposed mechanisms are to analyze the app's key security elements such as permission [4], sensitive API calling or critical code segments in the source code [5]. However, static analysis falls short in detecting transformation attacks. [6] describes the development trend of malware and some current detection methods.

In fact, by analyzing network traffic we can uncover the disclosure of sensitive information, the classification of network behavior, or automatic malware detection. R. Perdisci et al. classify malware according to the similarities in URLs extracted from malware's HTTP requests [7]. They propose a structural similarity of URL to cluster malware examples. Anomaly-based detection of malicious HTTP-traffic has been studied in [8]. Nizar Kheir presents a new technique to classify malware user-agent anomalies in [9] because user-agent anomalies are prevalent in malware HTTP traffic. Another technique presented in [10] generates state signatures by a long period of traffic observation.

These studies we mentioned above involve many aspects of malicious traffic analysis. Our approach is different from the above approaches in the following aspects. Firstly, Network traffic generated by mobile app is mirrored from the wireless access point to the server for data analysis. All data analysis and malware detection are performed on the server side, which does not consume any resources on mobile devices. Secondly, The proposed method combines multi-level network traffic analysis with machine learning that is capable of identifying Android malware with high accuracy. Moreover, TrafficAV not only displays the final detection results, but also analyzes the behind-the-curtain reason of malicious result.

III. Methodology

In order to detect Android malware, we propose TrafficAV method. Through the analysis of network traffic, TrafficAV can discover malicious network behavior. This process is illustrated in Figure 1.

A. Traffic Collection

In order to collect malware traffic traces in a real network environment, an active traffic generation and collection platform is utilized [11]. The platform is composed of four parts, including foundation platform, traffic generator, traffic collector and network proxy/firewall respectively. Foundation platform is built based on Android Virtual Device (AVD), while the function of traffic generator is to install and activate malware samples to generate network traffic automatically. A traffic collector is designed to collect inbound and outbound network traffic with tcpdump tool, and traffic mirror technology is utilized to mirror traffic to a server. The attack behavior is carefully monitored and controlled by proxy/firewall. By this traffic collection platform, collected traffic will be mirrored to the server to be analyzed.

B. Feature Extraction

During the feature extraction, all traffic files are processed to automatically generate feature sets. Two programs written by Python language are used to extract features of HTTP request packets and TCP flows.

1) Flow feature set: Six selected features on TCP flow are showed in Table I. The reason for selecting uploading bytes and downloading bytes is that for benign apps, generally speaking, the size of a request packet is usually small. While the returning data is usually a picture or web page or video, so the packets are typically large. But for some Trojans' traffic, an opposite behavior can be observed.

TABLE I: Features extracted from TCP flow

Id	Feature	Description
1	upBytes	Uploading bytes(client->server)
2	downBytes	Downloading bytes(server->client)
3	upPckNum	Total uploading packet number in a session(client->server)
4	downPckNum	Total downloading packet number in a session(server->client)
5	averageUpPckBytes	Average bytes of uploading packets(client->server)
6	averageDownPckBytes	Average bytes of downloading packets(server->client)

We select uploading packet number, downloading packet number, average bytes on one uploading packet and one downloading packet. Benign apps will try to make every packet carry more data, so the packet number is small, but the amount of data in each packet is large. As for Trojans apps, the data they interact every time is less, but there will be a lot of interactions.

In addition, there are a lot of similarities of TCP flow in every malware family. For example, we carefully observe three features (uploading bytes, downloading bytes, packets number in a session) of every TCP flow in FakeDoc family, FakeRun family, Iconosys family and Imlog family. The samples of each family are clustered into a group which means that these three features of the samples in each family share a lot of similarities.

2) HTTP request feature set: TrafficAV focus on HTTP packet because HTTP is the predominant protocol adopted by most mobile apps. TrafficAV leverages metadata information in HTTP request header to create HTTP request feature set. Four features extracted from HTTP request header are shown in Table II.

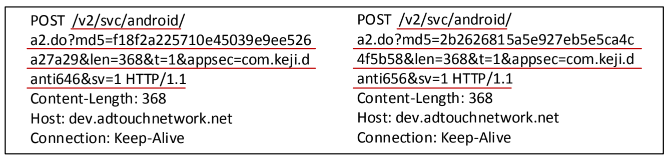
TABLE II: Features extracted from HTTP request header

Id	Feature	Description
1	Host	This field specifics to the Internet host and port number of the requested resource.
2	Request-Uri	The URI is from the request source.
3	Request-Method	The method from HTTP indicates the action to be performed on the identified resource.
4	User-Agent	This field contains information about the user agent originating the request.

Host: Using Host field in the HTTP request message to identify malicious app is a very effective method. Because if there is a communication between app and malicious Host, we can determine that the application is malware or suspicious app. However, classifying this malware and its family by solely relying on Host field is not sufficient. Because a malicious Host can possibly associate with a collection of different malwares. These malwares may have different malicious behaviors and should be assigned to different families.

Request-Uri: The Request-Uri is a long string. This brings great inconvenience to our experiment when we combine traffic features and machine learning algorithm to train detection model. Furthermore, we find a great regularity of the Request-Uri string in each family. In the case that Request-Uri does contain key in its string, the keys are the same while the values paired with the keys are always different. So we can extract the keys to represent the HTTP Request-Uri and ignore the values. Different keys are separated with the character "&". For example, in BaseBridge family, two different HTTP Request-Uri generated by two malicious apps contain the same keys, as Figure 2 shows. The underlined segment in Figure 2 is the Request-Uri. In this case we will treat this Request-Uri string as "md5&len&t&appsec&sv". When the string of Request-Uri does not contain key-value pairs, we find that in each family the Request-Uri has the invariant part. We can use the invariant part to represent the Request-Uri feature.

Request-Method: In most cases app developers only use the GET and POST two request methods. GET method is regarded as the most common request method. It is essential to send a request to obtain a resource from the server. Resources through a set of HTTP headers and rending data (such as html text, pictures or videos) are returned to the client. GET request never contains presentation data. POST method is to submit the data to the server. This method is also widely used. Almost all of the current submission operations are relying on the POST method. Analysis of a single request method seems meaningless, but when a large number of malware samples get together, we will find that only one request method is generally used in a malicious family, such as HTTP requests of all samples in Adrd family use GET method and in SMSreg family all HTTP requests apply POST method. From this point of view, the feature is effective for malware classification.



```
POST /v2/svc/android/                         POST /v2/svc/android/
a2.do?md5=f18f2a225710e45039e9ee526           a2.do?md5=2b2626815a5e927eb5e5ca4c
a27a29&len=368&t=1&appsec=com.keji.d          4f5b58&len=368&t=1&appsec=com.keji.d
anti646&sv=1 HTTP/1.1                         anti656&sv=1 HTTP/1.1
Content-Length: 368                           Content-Length: 368
Host: dev.adtouchnetwork.net                  Host: dev.adtouchnetwork.net
Connection: Keep-Alive                        Connection: Keep-Alive
```

Fig. 2: Examples of HTTP request in BaseBridge family

User-Agent: User-Agent holds the version information of request browser. Some malwares using the User-Agent leaks infected node information. For example, "Apache-HttpClient/ UNAVAILABLE (java 1.4)" is a malicious User-Agent. This User-Agent is a known bot [12]. User-Agent anomalies include typos, information leakage, outdated versions, and attack vectors such as XSS and SQL injection. So User-Agent is also a feature we should focus on.

C. Learning-based Detection

Machine learning can be used to automatically discover the rules by analyzing the data, and then the rules can be used to predict unknown data. We choose C4.5 decision tree algorithm to train detection models. C4.5 decision tree algorithms creates a model based on the tree structure. This method uses information gain ratio of training data set to build a classification model. The classification model is completed automatically using labeled data set with C4.5 algorithm. By applying the classification model, the classification of unknown network traffic can be conducted. The process of detecting one unknown app with HTTP detection model is described in Figure 3. The tree nodes represent traffic feature's name (such as Host, Request-Uri, Request-Method, User-Agent), and the branch represents the feature's value (such as host1, method1, uri2, ua1) associated with the above feature. Leaf nodes represent the class. A tuple of traffic features from one HTTP request message is sent to the detection module as an input. The classification starts from the root node and then it walks down along the corresponding branch (such as the direction of red arrow in Figure 3) according to the feature tuple (such as the input in Figure 3) until it reaches a leaf node (such as Family4 in Figure 3). Finally, the leaf node can be regarded as the detection result of the app. In other words, decision tree classification model creates a mapping relationship between a series of features and categories. According to a feature tuple, it can infer which class the app belongs to.

In the previous step, we also acquire the feature tuple from the TCP flow. Similarly, the flow detection model is obtained by training flow feature set using C4.5 decision tree algorithm. For the same reason, we can detect unknown apps with a trained TCP flow model. The detection process is similar to the process that is shown in Figure 3.

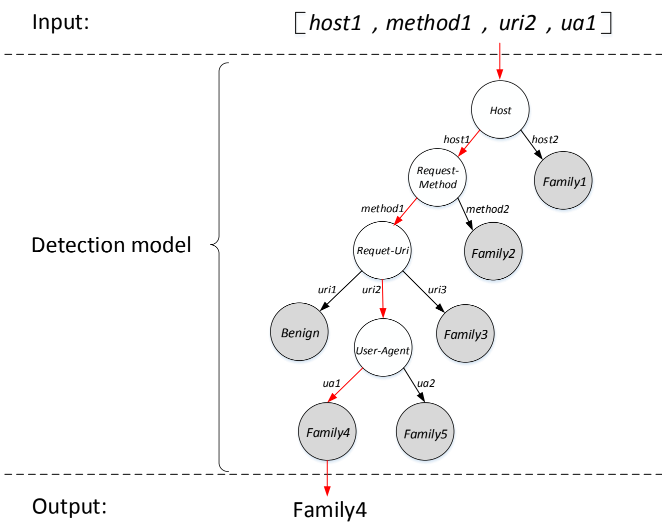


Fig. 3: The detection process with a trained C4.5 decision tree model

D. Result Explanation

In reality, a good detection system must not only show the final detection results, but also should make a reasonable explanation for the detection results. This will enhance the user's goodwill and confidence to detection system. Specifically, we detect apps with machine learning methods. The most common shortcomings of machine learning methods is that they are black-box method [13]. In TrafficAV we basically resolve the problem and extend the detection method based on machine learning. It can identify each network feature's contribution to the result. General detection methods just show the final results and user does not know how the decision is made. For the benefit of user, we design an explanation module for displaying the malicious results from HTTP request model. In addition, a good interpretive result can help researchers monitor malware's running mode and make researches have a deeper knowledge about malware.

To achieve this goal, we define four sets for each feature extracted from malicious HTTP request. S_1 set saves the values of all Host appearing in malicious HTTP request; S_2 represents the keys or unchanged part in Request-Uri; S_3 is Request-Method set; all User-Agent strings are stored in S_4. We define a feature vector where each feature is either 0 or 1. Every specific HTTP request feature tuple will be mapped to feature vector. The mapping relationship is as follows.

$$U_i = \begin{cases} 1 & \text{If the } ith \text{ feature exists in } S_i \\ 0 & \text{Otherwise} \end{cases}$$

For a better understanding of calculation process, we provide the following example. If a malware connects with a benign Host and contains an suspicious string in Request-Uri, in addition, it uses an malicious User-Agent. A corresponding vector U for this HTTP request may looks like this

$$U = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{matrix} Malicious \quad Host & S_1 \\ Suspicious \quad Request-Uri & S_2 \\ Common \quad Request-Method & S_3 \\ Suspicious \quad User-Agent & S_4 \end{matrix}$$

The importance of every feature in the detection model is different, so each feature should have different weight. We take the gain information as the corresponding weight. The weight vector $W =< W_1, W_2, W_3, W_4 >$ can be derived during the course of HTTP model training. The corresponding indication score function $F(x)$ is given by:

$$F(x) =< W, U >$$

With this function $F(x)$, we can calculate the score for each HTTP request, where x represents an HTTP request. The higher the score, TrafficAV has the more confidence in believing that this HTTP request is malicious. After getting malware's malicious score, we display each feature which has a contribution to the malicious result. To accomplish this goal, we have designed a sentence template to automatically generate description for each feature set. Table III shows the sentence template.

TABLE III: Sentence templates for result explanation

Set	Feature	Explanation
S1	Host	APP connects with the malicious Host:%s
S2	Request-Uri	APP contains suspicious Request-Uri:%s
S3	Request-Method	Most common Request-Method with the Host: %s
S4	User-Agent	APP applies suspicious User-Agent:%s

IV. Evaluation

In this section, we begin to evaluate the performance of TrafficAV. In practice, we evaluate it from the following four aspects.

A. Data Sets

1) APP data sets: Our malicious apps are from Drebin project [14], 5560 malicious apps are real malware samples. Our benign apps are downloaded from multiple popular app markets by app crawler. The initial benign samples are more than 10000. Each app downloaded from app market is sent to VirusTotal [15] to be tested. The app is added to our benign app set when the test result shows benign. Eventually, we get a benign app set of 8312 samples.

2) Traffic data sets: Android traffic data is collected with an automatic mobile traffic collection platform [11]. We get 500.4MB network traffic data generated by malware samples, and then extract 18.1MB malicious behavior traffic from it according to the malicious destination IP or domain name [15]. In the same way, we obtain 2.15GB mobile traffic data generated by benign apps. It takes more than two months for us to complete network traffic data collection. In order to improve the accuracy of the trained models, we only extract features from 18.1MB pure malicious traffic and 2.15GB benign traffic to produce training set and test set.

3) Training set and test set: Collected traffic is processed to extract valuable features on the server side. Ultimately 10225 feature tuples of HTTP request (benign HTTP requests included) and 24437 feature tuples of TCP flow (benign TCP flows included) are extracted. These feature tuples are used to train the detection model and test the detection performance of TrafficAV. Table IV shows the number of HTTP feature tuples and TCP flow feature tuples in every malware family. In this table, $N1$ represents the number of flow feature tuples and $N2$ indicates the number of HTTP feature tuples in each family.

TABLE IV: The number of traffic feature tuples in each malware family

Id	Family	N1	N2	Id	Family	N1	N2
A	plankton	1753	2224	J	Hamob	42	75
B	DroidKungFu	585	373	K	Iconosys	39	40
C	BaseBridge	368	862	L	SMSreg	21	25
D	FakeDoc	305	469	M	Geinimi	18	20
E	Gappusin	180	199	N	Adrd	17	16
F	FakeRun	139	147	O	Glodream	16	16
G	MobileTx	126	126	P	Ginmaster	13	15
H	Opfake	124	124	Q	Imlog	12	12
I	FakeInstall	102	142				

B. Detection Performance on Malicious Traffic

We split feature set into a training set (67 percent of traffic feature data) and a test set (33 percent of traffic feature data) randomly. Detection models and related parameters are determined by the training set. Test set is only used to evaluate the detection performance of TrafficAV. In order to eliminate the influence of contingency factors, we let this process repeat 10 times and the average result as the final detection result. We evaluate the detection performance of TrafficAV on every specific malware family. The detection performance of TrafficAV for each family's traffic is illustrated in Figure 4.

TCP flow model has the best performance on the FakeDoc family (D) with the accuracy rate up to 99.3% and the worst performance is on Ginmaster family (P) with accurate rate is only 25.0%. In Gappusin family (E), Ginmaster family (P) and Hamob family (J) flow detection model perform badly. In general, the performance of HTTP model on every family is quite well. On FakeDoc family (D), FakeRun family (F) and other families the accuracy rates reach 100%. But we should also note a special case that on SMSreg family (L), the detection rate of HTTP model is 0 and detection rate of flow model is 60%. From this point, we can conclude that flow detection model can be used as a supplement of HTTP detection model in some cases.

C. Detection Performance on Malicious App

1) Detection rates of two detection models : Because one app may generate more than one TCP flow and multiple HTTP

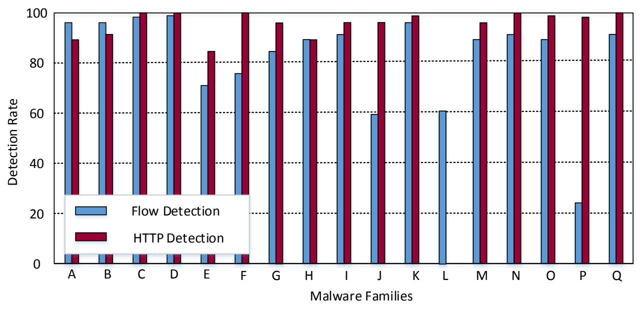


Fig. 4: Detection performance of each malware family

request packets, the detection rate of malicious traffic will not be equal to the detection rate on malicious apps. Furthermore, the traffic generated by each malware is the mixture of benign traffic and malicious traffic. Therefore, as long as we find some malicious traffic in one app, we will regard it as a malware. In this way, we calculate the true positive rate (TPR) and false positive rate (FPR) of two detection models. Figure 5 shows the final result. The detect rate of the flow model reaches 98.16% while the FPR is 5.14%. Although it can be used to detect malicious apps, it suffers from a high false positive rate. HTTP detection model on the other hand performs better than flow detection model. The TPR and FPR reach 99.65% and 1.84% of HTTP model, respectively. These statistics indicate that the accuracy of flow model is slightly lower than that of the HTTP model and the FPR was significantly higher than that of the HTTP model.

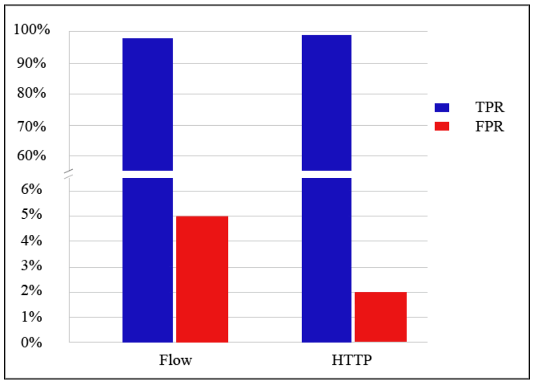


Fig. 5: The TPR and FPR on malware detection

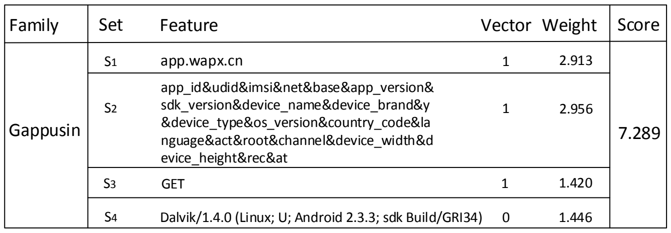
2) Comparison with other AV scanner: In our second experiment, we compare TrafficAV with other AV scanners. Since our malware samples are all from Drebin project [14], the comparing results with other scanners are cited from Drebin's experiment result. The comparing results of TrafficAV, Drebin and other anti-virus scanners are showed in Table V. The eight popular anti-virus scanners are respectively AntiVir, AVG, BitDefender, ClamAV, ESET, F-Secure, Kaspersky and McAfee. The detection performance of each scanner is taken from a public service VirusTotal [15] which is a website using many anti-virus softwares to analyze the security of uploading files.

TABLE V: Detection rates of TrafficAV, Drebin and other anti-virus scanners

	TrafficAV-Flow	TrafficAV-HTTP	DREBIN	AV1	AV2	AV3	AV4	AV5	AV6	AV7	AV8
Full dataset	98.16%	99.65%	93.90%	96.41%	93.71%	84.66%	84.54%	78.38%	64.16%	48.50%	48.34%

D. Result Explanation

Another advantage of TrafficAV is that it can produce a detailed explanation for HTTP model's detection result. We



Family	Set	Feature	Vector	Weight	Score
Gappusin	S1	app.wapx.cn	1	2.913	7.289
	S2	app_id&udid&imsi&net&base&app_version&sdk_version&device_name&device_brand&y&device_type&os_version&country_code&language&act&root&channel&device_width&device_height&rec&at	1	2.956	
	S3	GET	1	1.420	
	S4	Dalvik/1.4.0 (Linux; U; Android 2.3.3; sdk Build/GRI34)	0	1.446	

Fig. 6: Features and weights of one malware from Gappusin family

take one known malware from Gappusin family as an example to illustrate TrafficAV's result explanation. The HTTP request features extracted from one malware belonging to Gappusin family are displayed in Figure 6. Gappusin family is a popular malware family and its main functionality is to induce malicious charges. Malware belonging to this family automatically connect with network, order business or download adware without user's knowledge. In Figure 6, we can see that this malware connects with the malicious website "app.wapx.cn". The website is a known infection source which proves that the application is malware. Request-Uri contains suspicious keys. These characteristics may be signals to complete an interaction with a remote server. Its request method is GET and most malicious apps in Gappusin family use GET method for making a request to the server. Moreover, its User-Agent is an benign User-Agent, so the vector is 0. The dot product (7.289) of weight vector with feature vector is the malicious evaluation score of this malware. The higher the score is, the more likely this app is a malware. In addition, according to this score mechanism, the calculated malicious evaluation scores will be ranging from 1.420 to 8.735.

V. Conclusion

The increasing number of Android malware brings mobile users a elevating security risk, and makes the detection of mobile malware a greater challenge. In order to identify Android malware, we propose TrafficAV which proves that: by combining machine learning algorithm and traffic analysis, we can effectively detect malicious traffic and further detect malware. The detection rates of TCP flow and HTTP models reach 98.16% and 99.65% while the false positive rates are 5.14% and 1.84%. TrafficAV not only identifies the malicious apps, but also provides details about the detection results. In addition, TrafficAV completes traffic analysis and detection on the server side, which does not consume resources of mobile devices, and does not affect user's surfing behaviors. Although TrafficAV reaches a good performance in detecting mobile malware, it is largely limited by the existing malicious samples. The number of malware family and malware samples are important factors that affect the wide applicability of TrafficAV. In addition, due to the limited malicious traffic generated by the malware samples is not always malicious, we only get 18.1MB malicious traffic data. Small training samples has a negative impact on our detection results. So our future task is to collect and analyze more malware samples, and to continuously improve the detection models of TrafficAV.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grants No.60903176 and No.61472164, the Natural Science Foundation of Shandong Province under Grants No.ZR2014JL042, No.ZR2013FL003 and No.ZR2012FM010, the Doctoral Foundation of University of Jinan under Grant No.XBS1301.

References

[1] "2015 mobile threat report: The rise of mobile malware," https://securityintelligence.com/events/the-current-state-of-mobile-threats, Tech. Rep., 2015.

[2] "Sophos: Security threat report, 2014," http://www.sophos.com/en-us/medialibrary/PDFs/other/sophossecurity-threat-report-2014.pdf, Tech. Rep., 2014.

[3] "Android.bgserv," http://www.symantec.com/security response/writeup.jsp? docid=2011-031005-2918-99, Tech. Rep.

[4] Y. Zhang, M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X. S. Wang, and B. Zang, "Vetting undesirable behaviors in android apps with permission use analysis," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 611–622.

[5] X. Wang, K. Sun, Y. Wang, and J. Jing, "Deepdroid: Dynamically enforcing enterprise policy on android devices." in *NDSS*, 2015.

[6] M. Chandramohan and H. B. K. Tan, "Detection of mobile malware in the wild," *Computer*, vol. 45, no. 9, pp. 0065–71, 2012.

[7] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces." in *NSDI*, 2010, pp. 391–404.

[8] C. Krügel, T. Toth, and E. Kirda, "Service specific anomaly detection for network intrusion detection," in *Proceedings of the 2002 ACM symposium on Applied computing*. ACM, 2002, pp. 201–208.

[9] N. Kheir, "Analyzing http user agent anomalies for malware detection," in *Data Privacy Management and Autonomous Spontaneous Security*. Springer, 2013, pp. 187–200.

[10] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, "Automatically generating models for botnet detection," in *Computer Security–ESORICS 2009*. Springer, 2009, pp. 232–249.

[11] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, and J. Li, "A first look at android malware traffic in first few minutes," in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, vol. 1. IEEE, 2015, pp. 206–213.

[12] "Bots vs browsers - public bot / user agent database & commentary." [Online]. Available: http://www.botsvsbrowsers.com/details/431923/index.html

[13] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 305–316.

[14] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket." in *NDSS*, 2014.

[15] "Virustotal." [Online]. Available: https://www.virustotal.com/