Toward Real-time Packet Classification for Preventing Malicious Traffic by Machine Learning

Toki Suga
University of Tokyo
ga_su_@hongo.wide.ad.jp

Kazuya Okada
University of Tokyo
okada@ecc.u-tokyo.ac.jp

Hiroshi Esaki
University of Tokyo
hiroshi@wide.ad.jp

***Abstract*—Both enterprise and carrier networks face various cyber threats in daily operation. Although numerous security researchers have proposed various attack detection methods to defend against cyber-attacks, attackers have quickly responded with dramatic changes to their attack campaigns. However, ML-based methods tend to take long processing time to detect attacks compared with signature-based ones. Ideally, malicious packets must be intercepted on network path before they arrive at target hosts to avoid significant damage on users and network resources by incidents and to reduce security operation costs. To achieve ML-based attack detections in real-time, it will be necessary to resolve the problem of processing time increased by ML prediction. In this paper, we firstly propose a packet forwarding architecture with ML classification modules. We implemented this architecture on a prototype system with DPDK and common ML libraries and frameworks, and conducted preliminary experiments to reveal the system bottlenecks. Based on the results, we then proposed the DNS packet processing model that realizes ML-based attack detection in real-time by completing the classification during DNS name resolution. Our evaluation results show that almost all malicious queries are filtered before the initial packets arrive at the resolved IP address.**

***Index Terms*—Machine Learning, Packet Classification**

I. Introduction

Cyber attacks become big threats to the Internet infrastructure and its users. Although numerous security researchers have proposed various attack detection methods to defend against cyber-attacks, attackers have quickly responded with dramatic changes to their attack campaigns. Additionally, the scale of such campaigns has now spread all over the world. Furthermore, attack methods have become increasingly complex and can rarely be detected by pre-shared attack signatures. Accordingly, many of the current detection mechanisms deployed against such attacks now utilize machine learning (ML) features such as neural networks to detect mixed and complicated malicious activities [1], [2], [3].

In principle, attack detections on the network side are more beneficial for preventing security incidents, because in-network detection systems can effectively intercept malicious attacks before they arrive at the target hosts and thus prevent the spread of malicious activities to internal and external networks. However, to achieve the network-side detections, security devices must inspect every packet arriving in a network in real-time before they are forwarded. Indeed, current security devices such as Next-Generation Firewall (NGFW) and IDS can inspect network traffic with signatures at wire speeds of up to 100 Gbps. In contrast, when compared with signature-based methods, ML-based methods generally take much more time (the results in this paper show actual time values). For achieving link speed inspection, acceptable packet processing time is very short. Specifically, to achieve 10 Gbps throughput, 64 bytes must be handled at a total processing time of less than 50 ns.

While real-time attack detections are required for preventing security incidents, many studies that leverage ML-based methods have not yet considered the processing time required to detect attacks. If a detection mechanism takes an excessive amount of time to detect an attack, delays will result in the responses to the incident, or incident prevention failures will occur, potentially causing serious damage on the victim side, such as information leakage. Security specialists have been improving their detection accuracy by combining of new classification algorithms and new features, but this approach requires more processing time for each packet [4], [5].

To achieve ML-based attack detections in real-time, it will be necessary to resolve the following two problems. The first is the large processing time required for ML predictions. The second is finding ways to use of multiple algorithms in parallel to detect various attacks. In this paper, we firstly propose a packet forwarding architecture that rapidly applies learning results for packet classification. In our architecture, the entire process is split into five parts to clarify the bottleneck process. We implemented this architecture on a prototype system and evaluated it during a packet processing experiment. The results, which showed the baseline performance of the architecture, also showed that the classifier part is more than 20 times more time-consuming than the other parts, making it the most laborious process in the system. Based on the results, we then focused on individual protocol behaviors and real traffic patterns, and proposed the DNS packet processing model that realizes ML-based attack detection in real-time by completing the classification during DNS name resolution. Our implementation based on the prototype system filtered almost all malicious queries before the initial TCP traffic arrives at the resolved IP address in our evaluation.

II. Related Work

On the Internet infrastructure, various protocols are abused to conduct cyber attacks. For instance, DNS has been used to detect malicious activities such as domain generation

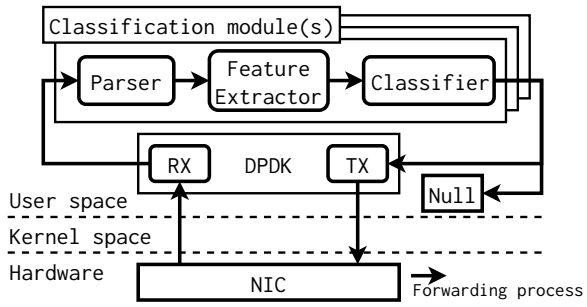

Fig. 1. System architecture of real-time packet classification by ML

algorithm (DGA) [6], DNS tunneling [7] and fast-flux [8]. In case of DGA detection, many studies leverage ML-based methods, but these methods have major problems when attempting to process packets in real-time. For example, Luo et al. [4] developed a process for detecting DGA-based malwares called DGASensor, which performs such detections effectively using a random forest algorithm, and which can process more than 7,000 queries in one second (143 μs for each packet). Meanwhile, deep neural networks (DNNs) were adopted for automatic feature extraction from real DNS traffic [5]. However, their convolutional NN (CNN) predictions take 50 - 70 ms per domain.

To achieve ML-based attack detections on the network side, it is necessary to leverage the current methods that improve packet processing performance. For instance, Lagopus switch and router [9] is a high-performance software OpenFlow 1.3 switch and router, which performs 10.1 MPPS (5.66 Gbps) with short packet of 64 bytes. The Lagopus adopts pipeline structure and CPU affinity methods, which technique can be applied to our system.

III. Design and Implementation

A. Architecture

In this section, we describe the design and system architecture of our real-time packet classification process that uses ML-based methods, which is shown in Figure 1. In our research, we split the entire packet process into five processing parts, Receiver (RX), Parser, Feature Extractor, Classifier, and Transmitter (TX). The Parser part extracts values from packet data such as src/dst IP address, src/dst port number, or application-specific values. The Feature Extractor part converts the values into feature vectors. The Classifier part classifies a packet as either legitimate or malicious using those feature vectors. If a packet is classified as legitimate, the packet will be forwarded to the TX part. However, if a packet is classified as malicious, the packet will be discarded in the system.

The classification system needs a sufficient amount of flexibility to replace and add/delete classification modules, because no single detector can prevent all malicious attacks. Hence, multiple detectors are needed as classification modules to detect and prevent various attacks. Additionally, since new detectors are constantly being produced, it must be easy to replace old detectors or to add others. For this reason, we did adopt software-based implementation which can leverage flexibility to classification modules as described above, even though hardware implementations such as those based on FPGA [10] operates faster than software. We implemented our system to reveal its basic performance characteristics and analyze the resulting bottlenecks. More specifically, We used the Intel DPDK [11] library for RX and TX in order to bypass the kernel stack overhead for forwarding. We also used LIBSVM [12] library and Tensorflow [13] framework to implement different classification modules. The Parser and the Feature Extractor parts were implemented from scratch.

B. Preliminary experiment

We conducted two types of preliminary performance measurements. These measurements were conducted to reveal the baseline performance levels of our system and to clarify the bottlenecks. The following paragraphs describe the details of the evaluation setup, including the classification modules, experimental procedures, and results.

Classification modules: To evaluate system performance levels, we implemented two classification modules based on LIBSVM and Tensorflow, which detect malicious DNS queries generated by DGAs. This is just an example and our detection targets will not be restricted to DGAs. As the detection algorithm, we use previously proposed simple algorithms [1], [2]. We chose the nu-SVC and the linear function as a type and an kernel function of SVM, respectively. The neural network model that we chose is based on a softmax regression and achieves about 91 % accuracy at the MNIST dataset. We did not retrain the both models in the experiments because training processes cannot be finished during packets processing.

For the dataset, we created a domain name list with legitimate or malicious labels assigned based on a publicly available dataset [14], [15]. More specifically, 55,000 legitimate domain names were randomly taken from Alexa Top Sites [14], and 55,000 malicious names were randomly taken from [15]. We used 100,000 domains for training and 10,000 for validation. The ratio of legitimate to malicious domains in both the training and validation datasets was 1:1.

For classification model training, we used the bag-of-words model [16] which further considers an order of domain name structure. Specifically, 2847-dimensional vectors were extracted from each domain name string. Each element of a vector was set to 0 or 1. In the domain names 39 character types could be used (a, b, c, $\cdots$, y, z, 0, 1, $\cdots$, 8, 9, ., -, _), and the maximum length of the domain names in the dataset was set at 73. When the Nth character of a domain name is X, which is a Kth character in 39 character types, 1 is set to the 39(N-1)+Kth element in the vector. For example, when a domain name *ask.com* is converted into a feature vector, 1 is set to 1 (a), 58 (s), 89 (k), 154 (.), 159 (c), 210 (o), and 247th (m) elements and 0 is set to other 2,840 elements.

Forwarding performance: In our experiments, three physical servers (sender, forwarder, and receiver) were used. The sender and the receiver were connected through the forwarder that

TABLE I
AVERAGE PROCESSING TIME IN EACH MODULE (µs)

	RX	Parser	Extractor	Classifier	TX	Total
LIBSVM	0.375	0.0972	2.173	161418.452	7.167	161428.282
Tensorflow	0.276	0.0838	1.360	69.938	0.327	72.001

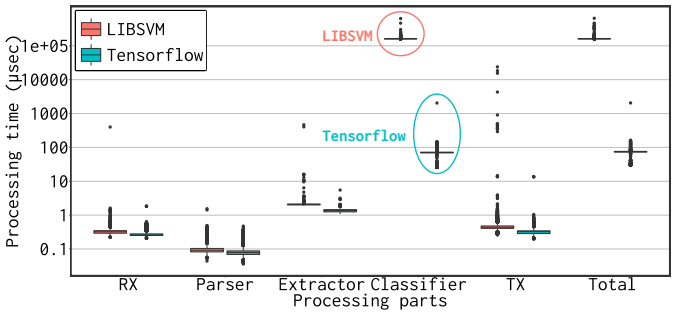

Fig. 2. Individual module processing times

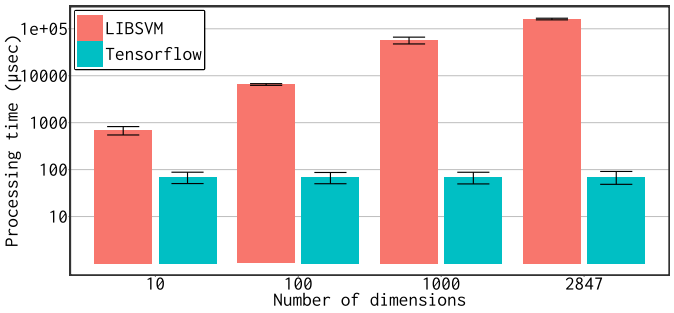

Fig. 3. Processing time on classifier parts with different dimension size

hosts our system with the two classification modules, as mentioned above. The sender generates and sends DNS queries to the receiver based on the validation dataset. Both the sender and receiver machines were equipped with Intel i7-4770K CPUs (3.50 GHz, 4 cores in total), 32 GB of RAM, and Intel 1G NIC. The forwarder machine was equipped with Intel i7-6700K CPUs (4.00 GHz, 4 cores in total), 32 GB of RAM, and Intel EXPI9402PT PRO/1000 PT Dual Port NICs. The link speed was set at 1 Gbps and we confirmed that a sample DPDK forwarding program could achieve that speed. We then measured the processing time of each part shown in Figure 1 using the *clock_gettime* function of the program.

Figure 2 shows the processing time of each module in log scale, whereas Table I indicates average processing time in each part. These results show that the Classifier is the most time-consuming part of the whole processing. In practice, 97.1 % of the total processing time was consumed by the Classifier. In the LIBSVM module, each processing part takes longer processing time than the Tensorflow results due to some outliers. When the average processing time was converted to throughput, that value was approximately 4.43 Kbps (6.19 pps) and 9.94 Mbps (13.89 Kpps) for LIBSVM and Tensorflow, respectively.

Feature size dependency: To reduce the processing time of the Classifier part, next we measured the impact of feature vector size on the processing time. To accomplish this, we changed the number of dimension from 10 to 2,847 and compared the processing time of the Classifier parts. Figure 3 shows the processing time for each dimension size of the features on a log scale. As can be seen in the figure, while the time increases linearly with the number of dimension in LIBSVM, the average time entirely falls in a range of 65 - 75 µs in Tensorflow. This result shows that the two modules need more than 65 - 75 µs, even if the input feature size is just 10. Table II shows the detection accuracy of the

TABLE II
DETECTION ACCURACY (%) IN DIFFERENT SIZE OF FEATURE VECTORS

	Number of dimensions			
	10	**100**	**1000**	**2847**
LIBSVM	42.88	79.45	91.46	91.42
Tensorflow	59.97	75.01	87.21	87.22

classification task with different sized feature vectors. These results confirm that adding more feature vectors improves the detection accuracy.

C. Additional processing time

As shown in Section III-B, we show that the measurement result of our prototype system is forwarding performance of approximately 10 Mbps in Tensorflow, which falls behind the required level of 1 to 10 Gbps. The Classifier is the most time-consuming part. This part is independent of input feature size according to the feature size dependency analysis in the preliminary measurements. However, more features and long processing time contribute to improving classification accuracy of ML-based modules. Accordingly, we need to leverage more processing time on the classification part, as well as to reduce entire processing time on the system. To achieve the goal, we focus on individual protocol behaviors and real traffic patterns.

On DNS protocol, for example, even the system cannot finish classification for a received query packet from a client, there is time to receiving the reply packet from the DNS server. The system can use the time as additional processing time. By this technique, we can save roughly 1 - 10 ms for each query. In this paper, we expand the prototype architecture in Figure 1 and propose a packet classification model which achieves ML-based attack detection in real-time. This model obtains additional processing time outside of the packet processing in order to avoid any modification on classification modules.

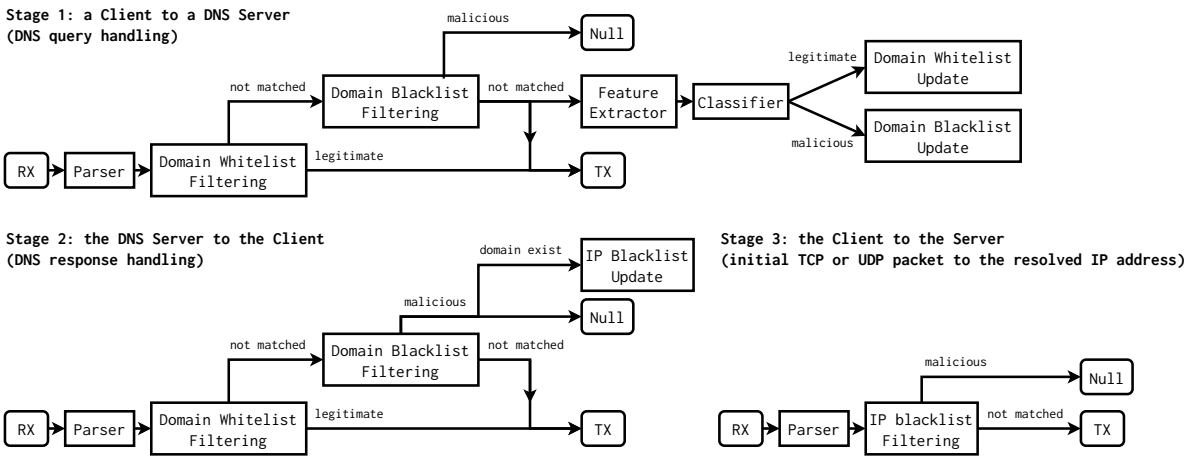

Fig. 4. The system architecture of proposed model

In addition, there was no overlapping between the traffic data in the previous experiment, but real networks have periodic patterns. Accordingly, we will not need to classify repeated traffic by classification modules. This idea is achieved by simple filtering methods such as whitelists or blacklists.

We integrate these two ideas into the system as illustrated in Figure 4. The key technique for the integration is to finish time-consuming ML-based classification during DNS name resolution, and to apply the classification result to simple filtering methods. The additional components include three filtering modules, a domain whitelist, a domain blacklist, and an IP blacklist. As five new processing parts, Domain Whitelist Filtering, Domain Blacklist Filtering, IP blacklist Filtering, Domain Blacklist Update, and IP Blacklist Update are added. In our model the packet processing is classified into the following three stages by the result of the Parser part.

Stage 1: a Client to a DNS Server (DNS query handling): A DNS query from a client firstly passes the Domain Whitelist Filtering part. When the query name (qname) is classified as legitimate in this part, the packet is forwarded to the TX part. In contrast, when the qname is not matched in the whitelist, the packet next passes the Domain Blacklist Filtering part. When the qname is classified as malicious in this part, the packet is discarded on the NULL device. In contrast, when the qname is not matched in the blacklist, the packet is forwarded to the TX part. Then, after the transmission, the Feature Extractor part converts the qname into feature vectors, and the Classifier part classifies the qname. If the classification result indicates the packet as legitimate, the qname is added to the domain whitelist. Conversely, if the result indicates malicious, the qname is added to the domain blacklist to block DNS queries which has same qname in early stage without ML processing.

Stage 2: the DNS Server to the Client (DNS response handling): A DNS response from the DNS server firstly passes the Domain Whitelist Filtering part. When the qname is classified as legitimate in this part, the packet is forwarded to the TX part. In contrast, when the qname is not matched in the whitelist, the packet moves to the Domain Blacklist Filtering part. When the qname is classified as malicious in this part, the packet is discarded in the system and the RCODE in the response message is checked. When the RCODE is NoError, IP addresses of the qname can be extracted and the IP addresses are added to the IP blacklist to discard packets in the Stage 3. Conversely, when the qname is classified as legitimate in the part, the packet is forwarded to the TX part.

Stage 3: a Client to the Server (initial TCP or UDP packet to the resolved IP address): An initial TCP or UDP packet to the resolved IP address from a client passes the IP Blacklist Filtering part. When the destination IP address is classified as malicious in the part, the packet is discarded in the system. In contrast, when the address is not matched in the blacklist, the packet is forwarded to the TX part.

In our system, all packets pass through only simple filtering modules, processing time of which is shorter than that of ML-based classifiers. Accordingly, we can reduce entire packet processing time on the system.

IV. Evaluation

In this section, we reveal the performance of our proposed system by two experiments. We implemented the architecture that is shown in Figure 4 by expanding the prototype system described in Section III. Three types of filtering methods are implemented by using hash-based data structures. The domain name lists such as Alexa Top 1 Million Sites [14] generally do not contain subdomains. Accordingly, to improve a match rate of the whitelist, an extracted domain name by the Parser part is converted to a part of the name which consists of one label and second-level domain (SLD) and top-level domain (TLD) by using List of second-level domains [17].

A. System performance

Figure 5 shows the experimental environment for the performance measurements. In this experiment, the client sends *Telnet* connections to target servers which are indicated by the FQDNs list. The list which the client queries is same

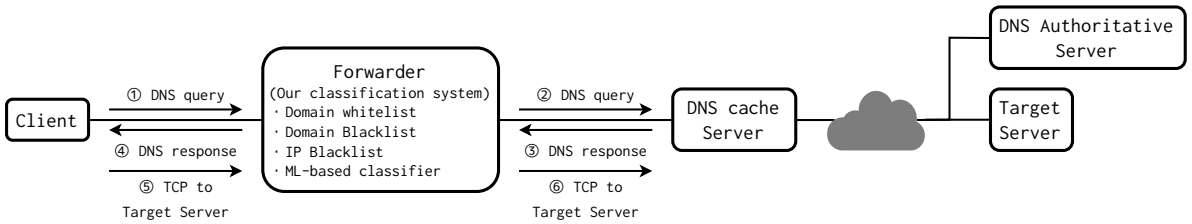

Fig. 5. Experimental environment

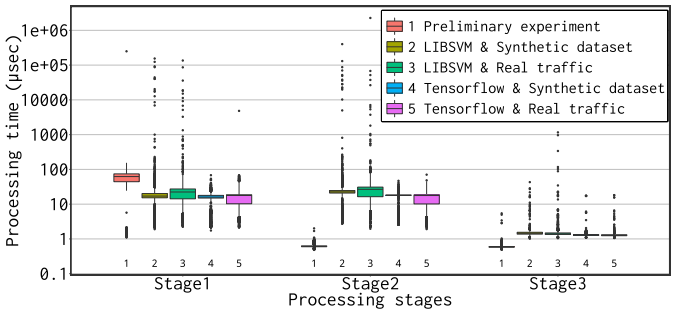

Fig. 6. Processing time in each stage

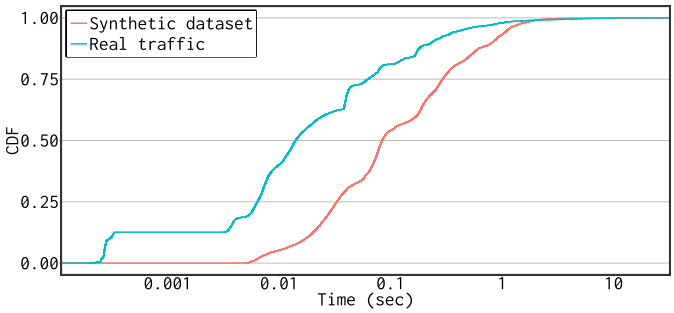

Fig. 7. Name resolution time for each query

as the validation dataset of the preliminary experiment. The DNS cache server resolves the names. The forwarder hosts our system. We assumed that malicious activities have not been observed in the environment. For this reason, in this experiment initial table size of the three lists are set to 10,000, 0, and 0 for the domain whitelist, the domain blacklist, and the IP blacklist, respectively. We use OpenDNS Random Sample List [18] as the initial domain whitelist, which has 9 overlapping domains with the FQDNs list.

The "Synthetic dataset" column of Table III shows the number of domains which are classified in each stage. The "Forward to classifier" row indicates the number of domain names which passed the Classifier part, and the prediction result. The "Passed all stage" row indicates the number of initial TCP packets classified as malicious based on the Stage 1, but the classification could not finish before the initial packets forwarded to the server. The result shows that 99.98 % of malicious queries are filtered before the initial TCP traffic arrives at the resolved IP address in both LIBSVM and Tensorflow modules.

Figure 6 shows the processing time from the RX part to the TX part in each stage. In the Stage 1, the total time is reduced by approximately 55 µs for Tensorflow as compared to the original processing time as we measured in section III. On the other hand, approximately 1 - 20 µs of additional processing time is introduced by our system in the Stage 2 and Stage 3 compared with the prototype system, because the prototype has no processing module for DNS response packets and initial TCP packets..

In figure 7, the "Synthetic dataset" line shows the CDF of resolution time for each DNS query. This result confirms that our system can save approximately 0.01 - 1.0 s additional processing for ML-based classifications.

B. *Performance with real traffic*

Next, we measured the performance with a real DNS query dataset which has periodic patterns as validation dataset which contains collected 24-hour period DNS queries in an academic research network in Japan. The details of the traffic are shown in Table IV. We created the whitelist by using Alexa Top 1 Million Sites [14]. In this experiment, 10,000 FQDNs were randomly chosen from the dataset, which has 5,891 overlapping domains with the whitelist. The initial values of the three lists are set to 1,000,000, 0, and 0 for the domain whitelist, the domain blacklist, and the IP blacklist, respectively.

The "Real traffic" column of Table III shows the number of domains in each stage. The result shows that our system can filter out 100 % of malicious queries in real traffic, too. The processing time from RX to TX in Figure 6 shows no significant difference between the synthetic dataset and real traffic. In figure 7, the "Real traffic" line shows the CDF of resolution time for each DNS query. In real traffic, periodic queries hit the cache in the DNS cache server. For this reason, the time that our system can obtain for ML-based classification is 53.7 % shorter than that of the synthetic dataset on average.

V. DISCUSSION

In this section, we discuss the performance of our system and our approaches to fill the gaps in real-time ML-based attack protections. In the proposed processing model, available processing time for classification algorithms depends on DNS resolution time between clients and DNS servers. Additionally, the time is not stable and varies with network environments. As a future work, we conduct performance measurements with

TABLE III
THE NUMBER OF DOMAINS IN EACH STAGE (10,000 DOMAINS)

Stage	Decision		Synthetic dataset		Real traffic	
			LIBSVM	Tensorflow	LIBSVM	Tensorflow
1	Legitimate		9	9	5,891	5,891
	Forward to classifier	Legitimate	9,991 5,516	9,991 5,519	4,109 3,056	4,109 2,251
		Malicious	4,475	4,472	1,053	1,858
	Malicious		0	0	0	0
2	Legitimate		5,525	5,528	8,947	8,142
	Malicious		4,474	4,471	1,053	1,858
3	Legitimate		0	0	0	0
	Malicious		0	0	0	0
	Passed all stage		1	1	0	0

TABLE IV
DETAILS OF DNS QUERY TRAFFIC DATA

Index	Value
Period	2018-1108-06:28:31 - 2018-1109-06:28:22
Total amount	28165972
Variety of qname	786611

different type of DNS resolvers such as a public DNS server to reveal general time, and need to examine available classifier algorithm except for SVM and neural network. In addition, we also explore to use other features to improve classification accuracy by the algorithms. In this paper, the classifiers use only simple features acquired from qnames, and the features are not sufficient for advanced classification algorithms in cyber security.

Our system can operate only one ML-based detector at the same time. However, to detect various malicious activities in networks, multiple detectors are needed to operate at the same time, as mentioned in Section III. We consider methods such as parallelization to operate multiple detectors simultaneously. Furthermore, to improve the software routing performance of our system, we intend to review and reuse current methods such as pipeline structure or CPU affinity methods.

VI. CONCLUSION

In this paper, we discussed the need for ML-based attack detection during real-time packet forwarding as a method of preventing security incidents to avoid significant damage on users and network resources and to reduce security operation costs. As a potential solution, we firstly proposed and implemented a prototype system, and then evaluated its performance levels by experiments in order to clarify its architectural limitations and bottlenecks. Based on the results, we then proposed the DNS packet processing model that realizes ML-based attack detection in real-time by completing the classification during DNS name resolution. Our evaluation shows that almost all malicious queries are filtered before the initial TCP traffic arrives at the resolved IP address.

ACKNOWLEDGMENT

This work was supported by JST CREST Grant Number JPMJCR1783, Japan.

REFERENCES

[1] M. Zouina and B. Outtaj, "A novel lightweight url phishing detection system using svm and similarity index," *Human-centric Computing and Information Sciences*, vol. 7, no. 1, p. 17, 2017.

[2] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016, pp. 21–26.

[3] A. Nisioti, A. Mylonas, P. D. Yoo, and V. Katos, "From intrusion detection to attacker attribution: A comprehensive survey of unsupervised methods," *IEEE Communications Surveys & Tutorials*, 2018.

[4] X. Luo, L. Wang, Z. Xu, J. Yang, M. Sun, and J. Wang, "DGASensor: Fast Detection for DGA-Based Malwares," in *Proceedings of the 5th International Conference on Communications and Broadband Networking*. ACM, 2017, pp. 47–53.

[5] B. Yu, D. L. Gray, J. Pan, M. De Cock, and A. C. Nascimento, "Inline dga detection with deep networks," in *Data Mining Workshops (ICDMW), 2017 IEEE International Conference on*. IEEE, 2017, pp. 683–692.

[6] A. K. Sood and S. Zeadally, "A taxonomy of domain-generation algorithms," *IEEE Security & Privacy*, vol. 14, no. 4, pp. 46–53, 2016.

[7] G. Farnham and A. Atlasis, "Detecting dns tunneling," *SANS Institute InfoSec Reading Room*, vol. 9, pp. 1–32, 2013.

[8] T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling, "Measuring and detecting fast-flux service networks." in *NDSS*, 2008.

[9] Y. Nakajima, T. Hibi, H. Takahashi, H. Masutani, K. Shimano, and M. Fukui, "Scalable high-performance elastic software openflow switch in userspace for wide-area network," *Proc. Open Networking Summit (ONS 2014), Santa Clara, CA*, 2014.

[10] K. Alrawashdeh and C. Purdy, "Reducing calculation requirements in fpga implementation of deep learning algorithms for online anomaly intrusion detection," in *Aerospace and Electronics Conference (NAECON), 2017 IEEE National*. IEEE, 2017, pp. 57–62.

[11] I. D. P. D. Kit, https://www.dpdk.org/, 2018.

[12] C.-C. Chang and C.-J. Lin, "Libsvm: a library for support vector machines," *ACM transactions on intelligent systems and technology (TIST)*, vol. 2, no. 3, p. 27, 2011.

[13] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[14] K. R. Competitive Analysis, & Website Ranking — Alexa. (2018) https://www.alexa.com/.

[15] D. N. O. Project, https://data.netlab.360.com/dga/, 2018.

[16] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 1245–1254.

[17] "List of second-level domains - domains index," https://domains-index.com/downloads/list-of-second-level-domains/, 2018.

[18] "opendns/public-domain-lists: Opendns public domain lists of domain names for training/testing classifiers," https://github.com/opendns/public-domain-lists, 2018.