Unknown Family Detection Based on Family-Invariant Representation

Toshiki Shibahara[*]
NTT Secure Platform Laboratories
toshiki.shibahara.de@hco.ntt.co.jp

Daiki Chiba
NTT Secure Platform Laboratories
daiki.chiba@ieee.org

Mitsuaki Akiyama
NTT Secure Platform Laboratories
akiyama@ieee.org

Kunio Hato
NTT Secure Platform Laboratories
kunio.hato.gm@hco.ntt.co.jp

Daniel Dalek
NTT Security
daniel.dalek@nttsecurity.com

Masayuki Murata
Osaka University
murata@ist.osaka-u.ac.jp

Abstract

Researchers and security vendors have proposed detection systems leveraging machine learning. However, these systems potentially have a vulnerability; they poorly detect malicious data created with newly developed attack tools (unknown families). If features of unknown families significantly differ from those of known families, detecting unknown families is extremely difficult. We focus on the tendency that some features are inherent across different families because attackers have to use such features to exploit vulnerabilities or reduce the cost of attacks. Based on this insight, we propose a method, i.e., a neural network, that classifies based on an invariant representation in similar known families for improving in the detection of unknown families. The family-invariant representation is learned with adversarial training: an optimization method used in domain adaptation methods. Specifically, we calculate the prediction probabilities of families using a conventional neural network and optimize the representation to confuse families whose predicted probabilities are similar. We applied our proposed method to a malicious communication-sequence detection system. Our method outperformed a conventional neural network in terms of the true positive rates of unknown families by at most 19%.

1 Introduction

Attackers automatically create malicious data with attack tools to efficiently accomplish their malicious objectives. For example, they create malware samples with toolkits [1] and malicious websites with exploit kits [2]. To detect malicious data and prevent damage caused by them, researchers and security vendors have proposed detection systems leveraging machine learning [3, 4, 5]. These systems effectively detect new malicious data on the basis of similarities with training data: previously collected malicious and benign data. However, these systems potentially have a vulnerability. Specifically, they poorly detect malicious data created with newly developed attack tools (unknown families) [6]. Since attackers develop new attack tools to evade detection systems, unknown families have different features from malicious data created with known attack tools (known families).

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

[*]The author is also with Osaka University, Osaka, Japan.

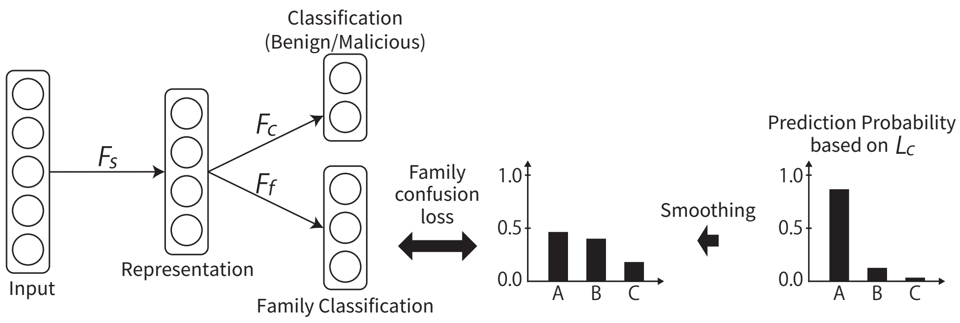

Figure 1: Neural network architecture of our proposed method.

Therefore, unknown families are difficult to detect with conventional detection systems, with which training and test data are assumed drawn from the same distribution.

If features of unknown families significantly differ from those of known families, detecting unknown families is extremely difficult. However, some features are inherent across different families because attackers have to use such features to exploit vulnerabilities or reduce the cost of attacks. For example, in drive-by download attacks, applications exploiting browsers or their plugins are limited to Flash, PDF, and Java, and abused domains are frequently acquired from loosely operated registrars [7]. Based on this insight, features inherent across known families are effective in improving the detection of unknown families. We should also be aware that effective features are not always inherent across *all* known families. For example, some families abuse Flash to exploit a vulnerability of a browser, but some families abuse PDF. Therefore, we use features inherent across similar families to improve detection performance.

We propose a method, i.e., a neural network, that classifies based on an invariant representation in similar known families for improving in the detection of unknown families. The family-invariant representation is learned with adversarial training: an optimization method used in domain adaptation methods [8, 9]. Conventional domain adaptation methods optimize representations to equally confuse source and target domains. With our proposed method, we prioritize similar known families in representation learning. Specifically, we calculate prediction probabilities of families using a conventional neural network and optimize a representation to confuse families whose predicted probabilities are similar. We applied our proposed method to a malicious communication-sequence detection system and evaluated the method's detection of unknown families.

2 Proposed Method

Our proposed method uses dataset $\mathbf{X} = \{(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i)\}_{i=1}^{N}$ for training, where $\mathbf{x}_i$ is an input datum, $\mathbf{y}_i \in \mathbb{R}^2$ is a class label, i.e., benign or malicious, and $\mathbf{z}_i \in \mathbb{R}^{N_z}$ is a family label. A benign class label is denoted as $\mathbf{y}_i = [1, 0]$, and a malicious one is denoted as $\mathbf{y}_i = [0, 1]$. The family labels are attached only to malicious data, and N_z denotes the number of families. We use a neural network having two heads, as shown in Fig. 1. The shared network $F_s(\mathbf{x}; \boldsymbol{\theta}_s)$ calculates representation $\mathbf{h}$, one head $F_c(\mathbf{h}; \boldsymbol{\theta}_c)$ predicts a class label $\mathbf{y}$, and the other head $F_f(\mathbf{h}; \boldsymbol{\theta}_f)$ predicts a family label $\mathbf{z}$, where $\boldsymbol{\theta}$ denotes the parameters of neural networks. We use $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$ for a predicted class and family label, respectively.

Optimization. In the training phase, we consider three loss functions and optimize them using adversarial training. The first loss function is used to calculate classification loss and defined using the cross entropy $\mathcal{L}_c = -\sum_{i=1}^{N}\sum_{j=1}^{2} y_{ij} \log \hat{y}_{ij}$. The second loss function is used to calculate family prediction loss and defined using the cross entropy $\mathcal{L}_f = -\sum_{i=1}^{N}(\mathbb{1}[y_{i2} = 1]\sum_{j=1}^{N_z} z_{ij} \log \hat{z}_{ij})$. The third loss function is used to calculate family confusion loss $\mathcal{L}_{conf}$. We define this in the next subsection. We iteratively optimize the parameters of neural networks on the basis of the following two objectives:

$$\min_{\boldsymbol{\theta}_s, \boldsymbol{\theta}_c} \quad \mathcal{L}_c + \alpha\mathcal{L}_{conf} \tag{1}$$

$$\min_{\boldsymbol{\theta}_f} \quad \mathcal{L}_f, \tag{2}$$

where α is a hyper parameter that controls the effect of the family confusion loss.

Family Confusion Loss. We define the family confusion loss to prioritize confusion between similar families. To identify similar families, we use the prediction probabilities of families when we minimize $\mathcal{L}_c$ and $\mathcal{L}_f$. The prediction probability p_{ij} denotes a probability of family j when $\mathbf{x}_i$ is input. Before calculating the family confusion loss, we smooth p_{ij} by adding a constant $a \in \mathbb{R}$: $p'_{ij} = \frac{p_{ij}+a}{\sum_{k=1}^{N_z}(p_{ik}+a)}$, where p'_{ij} is the smoothed prediction probability. The family confusion loss is defined as follows:

$$\mathcal{L}_{conf} = -\sum_{i=1}^{N}(\mathbb{1}[y_{i2}=1]\sum_{j=1}^{N_z} p'_{ij} \log \hat{z}_{ij}). \tag{3}$$

Since p' is a distribution between p and a uniform distribution, and similar families have similar p, the representation is optimized to confuse more similar families than dissimilar ones. Therefore, we can obtain invariant representations in similar known families by minimizing $\mathcal{L}_{conf}$.

3 Evaluation

We apply our proposed method to a malicious communication-sequence detection system and evaluate the method's detection of unknown families. We describe the experimental setup in Section 3.1 and experimental results in Section 3.2.

3.1 Experimental Setup

Conventional Methods. We compare our proposed method with two conventional methods. One is a conventional neural network (baseline). It has the same neural network architecture as our method and optimized by minimizing $\mathcal{L}_c$. The other is a method for learning invariant representations in *all* known families without taking into account the similarities between families (equal confusion). With this method, the family confusion loss is defined using the cross entropy between a predicted family label and uniform distribution: $\mathcal{L}_{conf} = -\sum_{i=1}^{N}(\mathbb{1}[y_{i1}=1]\sum_{j=1}^{N_z}\frac{1}{N_z}\log \hat{z}_{ij})$.

Datasets. We use malicious and benign proxy logs for our evaluation. From these proxy logs, we extract sequences of communications whose source IP addresses and destination fully qualified domain names (FQDNs) are the same and extract feature vectors regarding their URLs and content-types (see Appendix A for details). The total number of benign feature vectors is 110,000 and that of malicious feature vectors is 856. The malicious data include four families: Rig, Neutrino, Magnitude, and Sundown. We conduct our evaluation using four datasets, each containing test data of one of the four families and training data of the other three families. In other words, a family in the test data is not included in the corresponding training data. Using these four datasets, we conduct the evaluation assuming a family in the test data is unknown. More detailed information on these datasets is provided in Appendix B.

Hyperparameter Optimization. We optimize the hyperparameters by a cross-validation using the training data. For the cross-validation, we prepare three different datasets, each containing validation data of one of three families and training data of the other two families. The benign data is randomly split into halves for training and validation. We select the best hyperparameters in terms of a partial area under a receiver operating characteristic (ROC) curve (pAUC) in a region of false positive rates (FPRs) from 0 to a threshold. Since detection systems for cyber security commonly maintain their FPRs to less than 0.1, we selected 0.1 as the FPR threshold when we calculate a pAUC.

3.2 Experimental Results

We compare the three methods' detection of unknown families using ROC curves, as shown in Fig. 2. When Neutrino was assumed to be unknown, all methods achieved high detection performance. When the others were assumed to be unknown, our method outperformed the conventional ones. In the best-case scenario, our method achieved a 19% higher true positive rate (TPR) than the baseline at 3.2% FPR when Magnitude was assumed to be unknown. Unlike our method, the equal confusion did not outperform the baseline. The representation of the equal confusion was

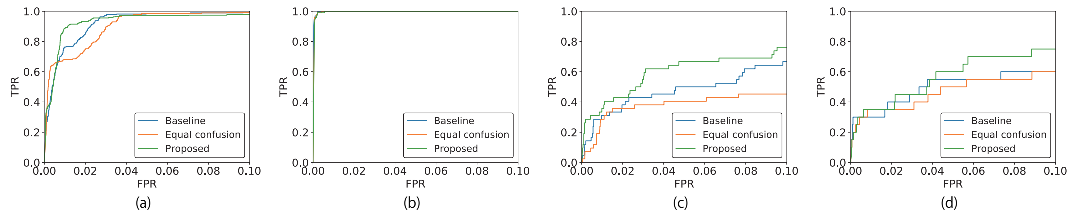

Figure 2: ROC curves. Families assumed to be unknown are (a) Rig, (b) Neutrino, (c) Magnitude, and (d) Sundown.

inferred not to include effective features because the representation was optimized to confuse not only similar families but also dissimilar families.

4 Discussion

Applicability of Proposed Method. In this evaluation, we applied our method to a malicious communication-sequence detection system, but our method can be applied to other systems. If malicious data are produced with attack tools, our method's detection of such data is expected to improve. Since most malicious data related to cyber security are produced in this manner, our method can be applied to detection systems for malicious Android applications [3] or malicious websites [5].

Validity of Evaluation. We used one of the four families as test data assuming that it is an unknown family. However, we should ideally evaluate the detection of malicious data produced by newly developed attack tools. In our evaluation, we showed that our method improved in detection performance or achieved sufficiently high detection performance if any family was assumed to be unknown. Therefore, our method is expected to improve in the detection of malicious data produced by newly developed attack tools.

5 Related Work

Detection Systems with Machine Learning. Systems for detecting malicious data, such as malicious Android applications [3], vulnerable firmware images of IoT devices [4], and malicious websites [5], have been proposed by applying machine learning. These systems are focused on detecting data similar to malicious training data but not unknown families. A method has been proposed for extracting invariant feature vectors with respect to changes in malicious data [10]. This method requires a hypothesis of change in malicious data, but we cannot predict the features of unknown families.

Adversarial Training. Adversarial training is used for learning domain-invariant representations in domain adaptation [8, 9]. In these studies, unlabeled data of a target domain were used for training, but we could not obtain unknown families in the training phase. Furthermore, domain adaptation methods optimize representations to equally confuse source and target domains, but we prioritize similar known families.

6 Conclusion

We have proposed a method, i.e., a neural network, that classifies based on an invariant representation in similar known families for improving in the detection of unknown families. The family-invariant representation is learned with adversarial training: an optimization method used in domain adaptation methods. Specifically, we calculate the prediction probabilities of families using a conventional neural network and optimize the representation to confuse families whose predicted probabilities are similar. We have applied our proposed method to a malicious communication-sequence detection system and showed that our method outperformed a conventional neural network in terms of the TPRs of unknown families by at most 19%.

References

[1] Juan Caballero, Chris Grier, Christian Kreibich, and Vern Paxson. Measuring pay-per-install: the commoditization of malware distribution. In *Proceedings of the 20th Usenix Security Symposium*, pages 187–202, 2011.

[2] Chris Grier, Lucas Ballard, Juan Caballero, Neha Chachra, Christian J Dietrich, Kirill Levchenko, Panayiotis Mavrommatis, Damon McCoy, Antonio Nappa, Andreas Pitsillidis, Niels Provos, M. Zubair Rafique, Moheeb Abu Rajab, Christian Rossow, Kurt Thomas, Vern Paxson, Stefan Savage, and Geoffrey M. Voelker. Manufacturing compromise: the emergence of exploit-as-a-service. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, pages 821–832, 2012.

[3] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *Proceedings of the 2016 Network and Distributed System Security Symposium*, 2016.

[4] Xiaojun Xu, Chang Liu, Qian Feng, Heng Yin, Le Song, and Dawn Song. Neural network-based graph embedding for cross-platform binary code similarity detection. In *Proceedings of the 24th ACM Conference on Computer and Communications Security*, pages 363–376, 2017.

[5] Teryl Taylor, Xin Hu, Ting Wang, Jiyong Jang, Marc Ph Stoecklin, Fabian Monrose, and Reiner Sailer. Detecting malicious exploit kits using tree-based similarity searches. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy*, pages 255–266, 2016.

[6] Symantec. Internet security threat report. https://www.symantec.com/security-center/threat-report.

[7] Davide Canali, Marco Cova, Giovanni Vigna, and Christopher Kruegel. Prophiler: a fast filter for the large-scale detection of malicious web pages. In *Proceedings of the 20th International Conference on World Wide Web*, pages 197–206, 2011.

[8] Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, and Mario Marchand. Domain-adversarial neural networks. *arXiv preprint arXiv:1412.4446*, 2014.

[9] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. In *Proceedings of the 29th Advances in Neural Information Processing Systems*, pages 343–351, 2016.

[10] Karel Bartos, Michal Sofka, and Vojtech Franc. Optimized invariant representation of network traffic for detecting unseen malware variants. In *Proceedings of the 25th USENIX Security Symposium*, pages 807–822, 2016.

[11] Toshiki Shibahara, Kohei Yamanishi, Yuta Takata, Daiki Chiba, Mitsuaki Akiyama, Takeshi Yagi, Yuichi Ohsita, and Masayuki Murata. Malicious url sequence detection using event de-noising convolutional neural network. In *Proceedings of the 2017 IEEE International Conference on Communications*, pages 1–7, 2017.

[12] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[13] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

A Malicious Communication-Sequence Detection System

A malicious communication-sequence detection system accepts proxy logs as input and detects sequences of communications to malicious websites, which are built with exploit kits and abused for drive-by download attacks. Proxy logs are collected by recording HTTP requests when hosts access the Internet. The logs include timestamps, source IP addresses, destination URLs, HTTP

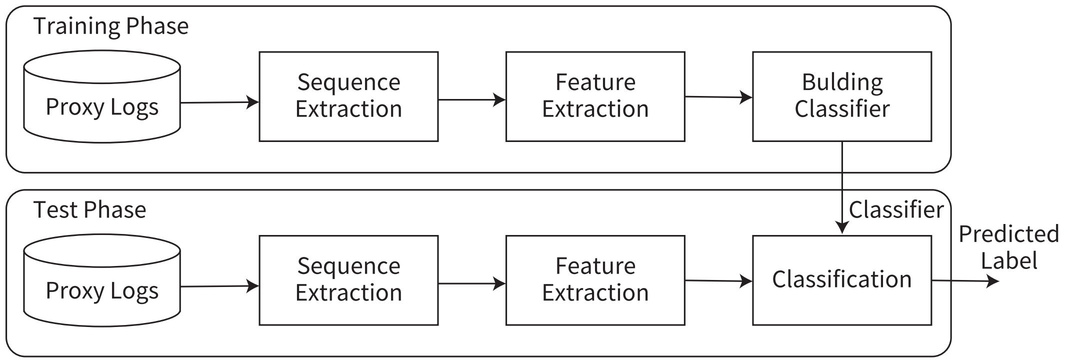

Figure 3: Malicious communication sequence detection system

methods, and HTTP headers. Since a victim's host typically sends multiple HTTP requests to the same malicious website, higher detection performance can be achieved using feature vectors representing a sequence of HTTP requests compared to using feature vectors representing a HTTP request [11]. Furthermore, since a host typically sends multiple HTTP requests to the same benign website, classification becomes easier by taking into account multiple HTTP requests to the same website. Therefore, we extract sequences of communications whose source IP addresses and destination FQDNs are the same. Note that we identify HTTP requests to the same website on the basis of their FQDNs because HTTP requests to the same domain are not necessarily related to the same websites if they are built on a hosting service.

Figure 3 shows the overview of this system. In the training phase, sequences are extracted from proxy logs whose class and family labels are known. After that, feature vectors are extracted from the sequences and used for building a classifier. In the test phase, sequences are extracted from proxy logs whose labels are not known. Feature vectors are then extracted from the sequences and classified as benign or malicious with the classifier. We describe sequence extraction, feature extraction, and classification below.

Sequence Extraction. We extract sequences of communications whose source IP addresses and destination FQDNs are the same from proxy logs. Furthermore, we split sequences if the elapsed time from the first communications becomes two minutes or the number of communications becomes 100. When a sequence is split, we start to extract a new sequence.

Feature Extraction. We design features referring to conventional systems for detecting drive-by download attacks [5, 7], as shown in Table 1. We extract feature vectors representing a communication then integrate vectors related to a sequence. Finally, we obtain feature vectors representing sequences of communications.

Features representing a communication are divided into two types: general and URL. General features are the interval, existence of identical communications, and combination of HTTP method and content-type. URL features are the presence of IP address in hostname, presence of subdomain, popularity of top level domain (TLD), length of FQDN/URL path/filename/query string/URL, depth of URL path, presence of binary file, filename extension (`.php` or `.html`), filename extension (`.js`), filename extension (`.pdf` or `.swf`), number of query parameters, and ratios of capital letters/small letters/numerical letters/symbols. HTTP methods are categorized into GET, POST, and others. Content-types are categorized into text, application, binary, image, video, audio, font, multipart, and others. The number of HTTP-method and content-type combinations is 27. We use the highest Alexa[2] rank among domains with a certain TLD as the popularity of that TLD.

We integrate the above features with different procedures depending on the data formats. For continuous features, we calculate their average and standard deviation. For binary features, we calculate their summation and average. For the category feature, we use their 1-grams and 2-grams. The dimension of an integrated feature vector is 793. The feature vectors are normalized before they are input into the classifier so that the average and standard deviation of each feature become 0 and 1, respectively.

[2]https://www.alexa.com/topsites

Table 1: Features representing a communication.

Type	No.	Feature	Format
General	1	Interval	Continuous
	2	Existence of the identical communication	Binary
	3	HTTP method and content-type	Category
URL	4	Presence of IP address in hostname	Binary
	5	Presence of subdomain	Binary
	6	Popularity of TLD	Continuous
	7	Length of FQDN	Continuous
	8	Length of URL path	Continuous
	9	Length of filename	Continuous
	10	Length of query string	Continuous
	11	Length of URL	Continuous
	12	Depth of URL path	Continuous
	13	Presence of binary file	Binary
	14	Filename extension (`.php` or `.html`)	Binary
	15	Filename extension (`.js`)	Binary
	16	Filename extension (`.pdf` or `.swf`)	Binary
	17	Number of query parameters	Continuous
	18	Ratio of capital letters in query string	Continuous
	19	Ratio of small letters in query string	Continuous
	20	Ratio of numerical letters in query string	Continuous
	21	Ratio of symbols in query string	Continuous

Table 2: Dataset.

		Training		Test	
Label	Family	Period	Number	Period	Number
Benign		Oct. 10, 2017	10,000	Jan. 16, 2018	100,000
Malicious	Rig	May 7, 2015–Nov. 5, 2016	270	Nov. 7, 2016–Oct. 25, 2017	270
	Neutrino	Jun. 19, 2013–Jul. 12, 2016	97	Jul. 13, 2016–Sep. 26 2016	97
	Magnitude	Jan. 15, 2014–May 28, 2015	41	May 28, 2015–Aug. 5, 2017	42
	Sundown	Dec. 27, 2015–Dec. 29, 2016	19	Dec. 29, 2016–May 7, 2017	20

Classification. We describe the detailed implementation of a neural network. As F_s, F_c, and F_f, we use multi-layer neural networks, each consisting of input, output, and one hidden layer. All layers are fully connected. The activation functions for the last layers of F_c and F_f are softmax, and those for all other layers are ReLU. We apply dropout [12] to F_c to prevent overfitting and select adam [13] as the optimizer.

B Datasets

We use malicious and benign proxy logs for our evaluation. Malicious proxy logs are prepared using pcaps collected from Malware Traffic analysis[3] and Broad analysis[4]. Malicious data include four families: Rig, Neutrino, Magnitude, and Sundown. Benign proxy logs are collected from company networks with users' consent. All information identifying users or companies is not recorded to protect privacy.

From these proxy logs, we extract sequences of logs whose source IP addresses and destination FQDNs are the same and extract feature vectors, as described above. Malicious data are divided into halves on the basis of the collection date. The former half is used for training and the latter for testing. Benign training data were collected on Oct. 10, 2017, and benign test data were collected on Jan. 16, 2018. The total number of feature vectors was 110,856. Table 2 shows the number and period of benign feature vectors and each family.

[3]https://www.malware-traffic-analysis.net/

[4]http://www.broadanalysis.com/

Table 3: Examples of benign and malicious communications

Label	Family	URL	Content-type
Benign		www.ntt.co.jp/news2018/1807/180718a.html	html
Malicious	Rig	[snipped].northwestfloridacannabis.org /?ct=kulture&oq=X96cpLOFRaAG[snipped]	x-shockwave-flash
	Neutrino	[snipped].morgansdecorators.com /street/[snipped]Y3B5eA.swf	x-shockwave-flash
	Magnitude	[snipped].dropsfry.gdn /d9947c8e03e9dc40167c02718275b280?[snipped]	x-shockwave-flash
	Sundown	[snipped].hse.mobi /7/?947545190441&id=2[snipped]	x-shockwave-flash

We prepare four datasets, each containing test data of one of four families and training data of the other three families. In other words, a family of the test data is not included in the corresponding training data. Using these datasets, we conduct the evaluation assuming a family of the test data is unknown. For example, if we assume that Rig is an unknown family, we use all training data except for Rig to build a classifier and use benign and Rig test data to evaluate the detection performance of that classifier.

Table 3 shows examples of benign and malicious communications. Among malicious data, the structure of URLs differs depending on their families. For example, the URLs of Rig, Magnitude, and Sundown have query strings, but the URL of Neutrino do not. The length of domains and depth of URL paths also differ. On the other hand, some features are inherent across all families. Communications of all families have the same identifier of the content-type and include random strings.