Regular Paper

Compression Algorithm Contribution for Infected-host Detection

Yasushi Okano†, Kazunori Kamiya†, Atsutoshi Kumagai†, Taishi Nishiyama†,
Bo Hu†, Masaki Tanikawa†, and Kazuhiko Ohkubo‡

†NTT Secure Platform Laboratories, Japan
‡Kyowa Exeo Corporation, Japan
†{yasushi.okano.ye, kazunori.kamiya.ew, atsutoshi.kumagai.ht, taishi.nishiyama.pt,
bo.hu.cf, masaki.tanikawa.ym}@hco.ntt.co.jp
‡k.ookubo@en2.exeo.co.jp

Abstract - Machine learning is being actively used to detect malware-infested hosts and their malicious communications. When applying machine learning, designing the right feature is the key for accurate detection. BoW (Bag of Words)-based feature extraction is widely applied in natural language processing and also utilized for malicious communication detection. However, BoW-based feature extraction does not always scale for handling network logs that often have new data sequences. By focusing on the fact that new data sequences in network logs are in many cases mostly similar but partly different, we propose a new detection method based on a data compression algorithm. Since the compression algorithm has a characteristic that data size after compressing is related to similarity of data, a compression algorithm based feature can be utilized for classification. According to our evaluation results with real-field proxy logs in an enterprise network, the proposed method has better at detection than a BoW-based detection method. In particular, its true positive rate (TPR) in a low false positive rate (FPR) area (0.5%) is over 30% higher than that for the BoW-based method. In addition, the results show that the proposed method effectively detects an infected host communicating with malicious URL that includes partially modified string from original malicious logs.

Keywords: malware, log analysis, data compression, machine learning

1 INTRODUCTION

Malware is becoming more sophisticated and has so many variants that anti-virus software does not detect all of them. In fact, it is reported that over 127.5 million pieces of malware were registered in 2016 [1]. To compliment the fact that detection at the endpoint is not always successful, network log analysis is one solution that monitors logs taken from network devices such as proxy and firewall and finds malicious communications derived from infected hosts.

In current log analysis, many monitoring rules have been deployed. Examples include network scans being detected once the number of different destination IP addresses from one source IP address exceeds a predefined threshold and a specific malware infection being determined if one host accesses a blacklisted URL. Many monitoring rules are based on operators' elaborations on creating rules, deciding thresholds, and maintaining blacklists. However, as malware has evolved to become able to change communication patterns easily, heuristic rule creation adds cost to operations and has difficulty catching up with malware modification. As a result, machine learning is gaining attention for automatically detecting evolving malware and for helping operations.

In applying machine learning for network log analysis, first, machine learning calculates feature values from network logs. For instance, feature values range from the length of a string, frequency of terms in device logs, and so on. Second, machine learning will classify the data into legitimate or malicious on the basis of feature values. In this process, infected hosts and malicious communications are detected.

There are many detection algorithms from LR (Logistic Regression), SVM (Support Vector Machine), RandomForest, and DNN (Deep Neural Network). However, the most critical factor for accurate detection is designing the right feature for the problem.

BoW (Bag of Words)-based features are widely applied in natural language processing and also utilized to detect malicious communications. However, BoW-based feature extraction does not always scale for handling network logs, which often have new data sequences.

By focusing on the fact that new data sequences in network logs are in many cases mostly similar but partly different, we propose a compression algorithm based feature and apply it to supervised learning for detecting malicious communications and infected hosts.

Simply put, a compression algorithm based feature is one form of the compression rate of data, which means how small the data becomes after being compressed. When the data sequence is similar to that in existing malicious data, the compression rate should be small because this data sequence is effectively compressed. On the other hand, if another data sequence is totally different from that in existing malicious data, the compression rate should be large. In this sense, the compression rate can be useful for finding if one data sequence is similar to that in malicious data. Consequently, the compression

ISSN1883-4566 © 2020 - Informatics Society and the authors. All rights reserved.

rate can contribute to detecting malicious communications.

We evaluated the proposed method with real proxy logs taken from an enterprise network. The results show that the proposed method is better at detection than the BoW-based method. In particular, the results show that the proposed method effectively detects an infected host communicating with a malicious URL that includes a partially modified string from original malicious logs.

Overall, our research makes three contributions.

1. We first apply the compression algorithm feature to supervised machine learning to detect malicious communications and infected hosts.

2. We evaluate the proposed method with real enterprise proxy logs and demonstrate that the proposed method performs better than a BoW-based classifier.

3. We analyze true positive and false negative use cases and clarify that the proposed method effectively detects partially modified strings from original malicious logs.

2 RELATED WORK

2.1 Classification Based on Compression Algorithm

Benedetto et al. [2] proposed relative entropy. Although patterns of the same consecutive code or similar repeated code are effectively compressed, patterns of different code are not. Relative information volume of data sequence x against data A is linked to how well data is compressed. Based on this observation, Benedetto et al. define relative entropy as how well new data x will be compressed with existing data A. Consequently, this is formulated as follows.

$$C_A(x) = Z(A\ cat\ x) - Z(A) \tag{1}$$

where Z is the function to output the data size after compression, and *cat* is the function to concatenate the first and second data sequences. Sometimes, normalized relative entropy is also used, which is defined to divide relative entropy by the size of data x.

Relative entropy has been applied to classification problems in several research areas [3] - [7]. To classify data x into group A and B, data x is normally classified into the more similar group. Relative entropy can be used as one index of expressing similarity; when relative entropy with group X is small, data x is similar to group X.

Bratko et al. [5] applied relative entropy to classify spam e-mails. They reported that it was more accurate than BoW based classification.

Nishida et al. [6] introduced a smoothing parameter and set the score in accordance with the following equation to classify malicious tweets from twitter logs.

$$Score = \frac{C_A(x) + \gamma}{C_B(x) + \gamma} \tag{2}$$

where γ is a smoothing parameter that should be set large to alleviate the impact of minor letters appearing a few times in a data string. Data x is classified as A if the score is small and B otherwise. This scoring technique enables us to apply a compression algorithm for a classification problem of comparably long data. Nishida et al. [6] also demonstrated that classification of twitter logs with this scoring mechanism has better accuracy than feature extraction with morphological analysis and classification with a CW(Confidence-Weighted linear classification) method [8].

Different compression algorithms are used depending on their purposes. It is reported that LZSS (gzip), LZW (compress), PMP (rar) are applicable for text data [3]-[6]. Adachi et al. [7] reported that bzip is applicable for music pieces.

2.2 Method of Extracting Feature from URL String

The BoW method is widely used to extract features from strings. BoW decomposes string text into words by separation of letters or morphological analysis and then generates each word as a one-dimensional feature. Since a URL is deemed as a one text string, BoW features can be extracted. Kumagai et al. [9] proposed BoW-based feature generation to apply LR supervised learning with L1 regularization and demonstrated that their method has better area under curve (AUC) than blacklist based detection.

Nelms et al. [10] proposed describing a URL attribute with a regular expression and applying unsupervised learning to generate a malware-specific URL access template. By comparing a target URL and the above template, the method successfully detects malware communication even when malware slightly modifies its access pattern.

In the security context, on the basis of knowledge on malware analysis, many kinds of statistical features have been proposed [11] such as the length of a URL and ratio of vowels in a URL.

3 PROPOSAL

We propose applying a compression algorithm based feature to apply supervised learning to detect malicious URLs and infected hosts. Since a large part of malware uses HTTP as a communication protocol with C2 servers, it can be mixed with normal Web access and is hard for operators to distinguish. Thus, in our research, we focus on analyzing HTTP proxy logs and detecting malicious URLs to find infected hosts.

An important observation on malware communication in HTTP is that they tend to access C2 servers with a slightly modified URL string in order to slip through blacklist-based detection with minimum engineering effort. In this case, simple blacklist matching does not

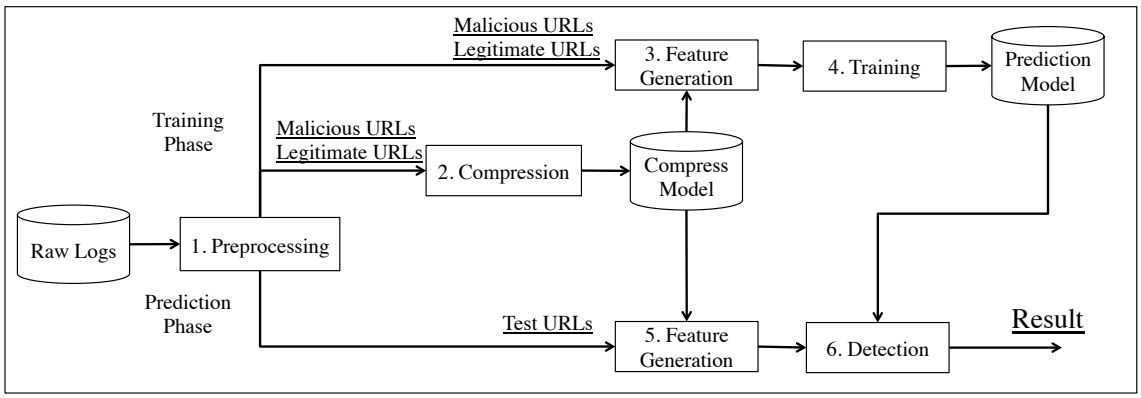

Figure 1: Overview of proposed method

Table 1: Dataset

	Number of hosts	Number of logs	Collection Period	Log Type
Malicious Logs	71,310	7,152,479	Feb. 2015 - Jul. 2015	Sandbox logs
Legitimate Logs	1,940	36,581,398	Feb. 2014 - Mar. 2014	Proxy logs in enterprise

catch up with malicious URLs since malware may have various URL access patterns even if its modifications are small. In this sense, we expect the compression algorithm based feature to correctly describe the similarity between a slightly modified malicious URL and a known malicious URL.

To the best of our knowledge, our proposal is the first to apply a compression algorithm to detect malicious communication URLs and infected hosts. In addition, our method is different from existing compression algorithm based methods in that we use a compression algorithm-based score as a feature in supervised learning and the feature can be combined with other features. Furthermore, our research considers a URL structure that has many kinds of attributes such as FQDN, PATH, and QueryString to generate a multi-vector compression algorithm feature for each attribute.

Figure 1 shows an overview of the proposed method. The flow of our proposal is as follows.

1. Input raw logs and execute preprocessing to obtain malicious URLs, legitimate URLs and test URLs

2. Compress malicious URLs and legitimate URLs to generate compress model

3. Input malicious URLs and legitimate URLs with application of compress model to generate compression algorithm features, namely Z_{pos}(Malicious Compression Rate) and Z_{neg}(Legitimate Compression Rate). Z_{pos} and Z_{neg} are defined in equation (3) and (4) respectively. Features are calculated for each attribute of a URL.

4. Train classifier with compression algorithm features and generate prediction model

5. Generate compression algorithm features from test URLs with application of compress model

6. Detect malicious URLs and infected hosts with application of prediction model

As with preprocessing, suitable data must be selected in machine learning for correctly estimating a classifier's performance. We execute two-phase cleansing in this process. First, we delete duplicate URLs in legitimate and malicious logs. This is because hosts may access the same URLs repeatedly. To correctly estimate a classifier, we leave first-to-appear logs in a dataset and eliminate duplicate logs.

Second, we eliminate URLs included in both malicious and legitimate logs, since having the same logs in both datasets may degrade the classifier's performance. In fact, there are many cases in which the same URLs are included in both logs. For instance, some service URLs are automatically accessed from specific applications installed in many environments. Search engine URLs are also often accessed from infected hosts for connectivity checks and included in malicious logs.

As for the compression algorithm features, we define Z_{pos} and Z_{neg} as follows.

$$Z_{pos}(x) = \frac{C_{pos}(x) + \gamma}{L(x) + \gamma} \tag{3}$$

$$Z_{neg}(x) = \frac{C_{neg}(x) + \gamma}{L(x) + \gamma} \tag{4}$$

where C_{pos} and C_{neg} are relative entropy between data x and malicious $\log(pos)$ or legitimate $\log(neg)$, L is data size of x, and γ is a smoothing parameter.

4 EVALUATION METHOD

4.1 Dataset

The dataset used for all evaluations is shown in table 1.

Malicious logs are taken from an in-house sandbox [12] where we run over 70K malware downloaded on a daily basis from a malware-sharing site and collect pcaps to extract URL information. Legitimate logs are taken from real-environment proxy in an enterprise network.

4.2 Evaluation Indices

Evaluations are executed on the basis of a holdout test that uses previous data in time series as the training dataset and evaluates with later data in time. Evaluation indices are AUC, partial AUC ($pAUC$) [13], and true positive rate $(TPR)_{0.5\%}$ [14].

AUC is the area under the curve drawn on a 2D surface of a false positive rate (FPR) and TPR by changing the score threshold. $pAUC$ is the area under the curve of a limited range of a FPR $[p1, p2]$. Considering the TPR as a function having a FPR as a variable, AUC and $pAUC$ are defined as follows.

$$AUC = \int_0^1 TPR\, dFPR \tag{5}$$

$$pAUC = \int_{p1}^{p2} TPR\, dFPR \tag{6}$$

Through our evaluation, we set $[p1, p2] = [0, 0.1]$.

$TPR_{0.5\%}$ is the TPR value for a low FPR, specifically $FPR = 0.5\%$. In security operations, a low FPR is crucial since the final judgment is done by operators. $pAUC$ and $TPR_{0.5\%}$ are important indices to estimate detection capability with a low FPR.

4.3 Selection of Compression Algorithm

The first evaluation is aimed at selecting a suitable compression algorithm. Several kinds of compression algorithms are used in existing research, so through our evaluation, we can select one algorithm that performs with both good accuracy and the least CPU (central processing unit) time.

We tested major compression algorithms LZSS (zip, LZ77), LZT [15] (a variant of compression algorithms LZW and LZ78), bzip2, and LZMA. To limit CPU time, since some compression algorithms take a very long time, we sampled 10 K malicious logs from Dec. 2014 and Jan. 2015 Sandbox logs and 10 K legitimate logs from Feb. 2015 and Mar. 2015 Proxy logs. In addition, we take the URL as only one attribute to generate a feature vector and execute a simple scoring calculation as follows.

$$Score = \frac{Z_{neg}}{Z_{pos}} \tag{7}$$

The process of registering a training dataset to a compression algorithm feature generator differs depending on how the compression algorithm works. Now, we overview the compression algorithm and its characteristics.

LZSS utilizes a sliding dictionary, which compresses data by only recording relative position and data sizes when the target data matches the longest data in the previous sliding window. Thus, one characteristic of LZSS is that data outside of the sliding window are not considered for compression. In general, a 32 kB sliding window is widely utilized. However, we implemented LZSS with a 20 kB sliding window for lowering computational cost. The compression rate of target data x is calculated by combining x with each 20 kB window data and applying LZSS and then returning the minimum compression rate as the final score.

LZT is a variant of LZW, and both are utilized in gif format files and compression commands. The same as LZW, LZT compression is based on dynamic dictionary insertion where new data sequences are added in a dictionary and target data x is recognized as the dictionary ID whose data sequence has the longest match with the target data. In LZT, dictionaries are composed with a Trie tree. In our evaluation, the compression rate of target data x is calculated by looking up the dictionary. Unlike LZW, which discards new data sequences when the dictionary is full, LZT swaps the LRU (least recently used) data sequence for a new data sequence.

bzip2 and LZMA utilize block sort and Markov algorithm based compression, respectively. The same as LZSS, bzip2 needs a blocking area, and LMA needs a sliding dictionary area. However, these areas are much larger than those for LZSS. Hence, our implementation stores all data for the compression algorithm where the compression score is calculated by simply combining target data x with the training dataset and compressing it.

For implementation, we utilized an existing python library for LZSS (libz), bzip2 (libzip2), and LZMA(liblzma). LZT is implemented in-house with cython.

In this evaluation, we set the classifier as an SVM, and TPR/FPR as a per log-basis calculation. We set the smoothing parameter as $\gamma = 10$ for all evaluations.

4.4 Selection of URL Attributes and Classifier

A URL has several attributes such as the URL itself, FQDN, Path, and QueryString. (For simplicity, in this research, we use Path to mean both Path and QueryString.) Through our evaluation, we can select the best URL attributes for detecting malicious URLs.

We evaluate the accuracy of URL attributes. We take an URL itself, FQDN, Path, and combination of FQDN and Path for testing. In this evaluation, we set the classifier as an SVM, the compression algorithm as LZT, and TPR/FPR as a per host-basis calculation.

The classifier is another factor for selection. We evaluate detection capability between different classifiers: Ridge regression, linear SVM, LDA(Linear Discriminant Analysis), NB (Naive Baise), and Adaboost. To conduct fair

Table 2: Classifier Performance and Execution Time Comparison Between Compression Algorithm

Algo.	Configs	CPU Time	AUC	pAUC	$TPR_{0.5\%}$
LZSS	level 6	16490s	0.968	0.0797	60.90%
LZT	24bit dict	20s	0.973	0.0825	68.10%
bzip2	level 9	68690s	0.521	0.0057	0.40%
LZMA	—	98112s	0.975	0.0828	68.30%

comparison, we execute grid-search to select best hyper-parameters for each classifiers. Implementation is done by using a python scikit-learn library. In this evaluation, we set the URL attribute as the FQDN and Path combination, the compression algorithm as LZT, and TPR/FPR as a per host-basis calculation.

4.5 Comparative Evaluation

We evaluate the proposed method in comparison with the conventional BoW-based detection method. First, we compared detection capabilities of the proposed and conventional methods. We also measured detection accuracy over time to find out how fast trained models deteriorate. Computational efforts are another important factor for practical use, so we measure CPU time and memory usage of the proposed and BoW method. BoW of a URL is extracted by setting $/, ?, =, \&$ as a separator and splitting the URL. In this evaluation we set the classifier as SVM, the compression algorithm as LZT, and TPR/FPR as a per host-basis calculation.

5 EVALUATION RESULT

5.1 Selection of Compression Algorithm

Table 2 shows the evaluation results for different compression algorithms. LZMA gives the best AUC, $pAUC$ and $TPR_{0.5\%}$ but takes the longest to compute. In contrast, LZT gives similar AUC, $pAUC$, and $TPR_{0.5\%}$ to LZMA and computes very fast. Hence, we selected LZT as the default compression algorithm in the later evaluation. bzip2 and LZSS do not perform as well as LZMA and LZT.

5.2 Selection of URL Attributes and Classifier

Table 3 shows the evaluation results of different attributes of a URL. From these results, the combination of URL attributes FQDN and Path gives the best $TPR_{0.5\%}$ and $pAUC$. Hence, we select the FQDN and Path combination as the default URL attribute in the later evaluation.

To select a suitable classifier, first, we visualize malicious and legitimate features. We calculated the compression algorithm feature and mapping onto a 2D surface in Fig. 2 where the horizontal axis is Z_{pos} (i.e., the malicious compression rate) and the vertical axis is Z_{neg} (i.e., the legitimate compression rate).

Table 3: Evaluation between URL attributes

URL attributes	AUC	pAUC	$TPR_{0.5\%}$
URL	0.8965	0.0720	41.80%
FQDN	0.8956	0.0631	43.60%
Path	0.7990	0.0562	43.20%
FQDN, Path	0.9306	0.0825	65.30%

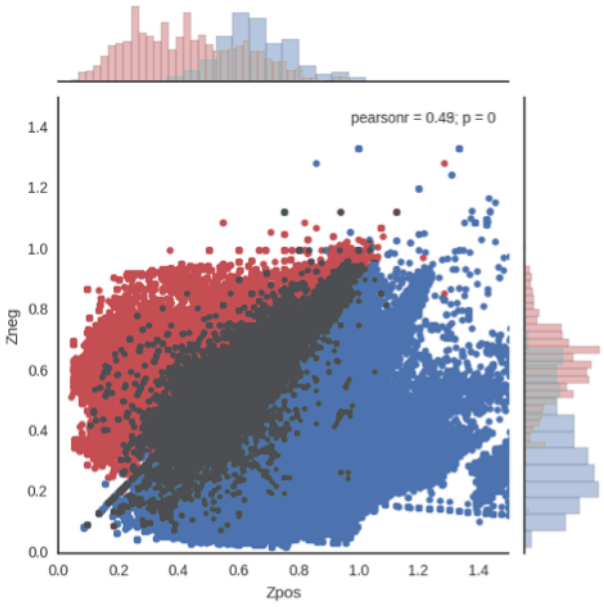

Figure 2: Compression algorithm feature mapping of malicious logs (red) and legitimate logs (blue)

This visualization shows that although some overlapping areas exist, malicious logs (red legend) and legitimate logs (blue legend) are mapped in the upper-left and lower-right areas, respectively. Many of malicious and legitimate logs seems to be linearly separated on the basis of Z_{pos} and Z_{neg}. Hence, these visualized results suggest that linear classification works well.

Table 4 shows the detection capability of different classifiers. SVM gives the best $TPR_{0.5\%}$ and $pAUC$. Hence, we select SVM as default classifier in later evaluation.

5.3 Comparative Evaluation

Table 5 shows the evaluation results for the proposed and BoW-based classification methods. From these results, the proposed method has better $TPR_{0.5\%}$ and $pAUC$ than the conventional BoW-based classification method.

Figure 3 shows the $TPR_{0.5\%}$ deterioration over time where the vertical axis is $TPR_{0.5\%}$ and the horizontal axis is time in weeks. This figure shows that $TPR_{0.5\%}$ gradually decreases over time. However, the proposed method always achieves a higher $TPR_{0.5\%}$ than the BoW method until 14 weeks have past.

Table 6 and 7 show comparison with BoW method on CPU time and memory usage, respectively. Proposed

Table 4: Evaluation between classifiers

Classifier	AUC	pAUC	$TPR_{0.5\%}$
Ridge α=0.1	0.9268	0.0791	59.40%
SVM C=0.025	0.9306	0.0825	65.30%
LDA	0.9291	0.0784	57.30%
NB	0.8166	0.0602	13.50%
AdaBoost	0.9420	0.0785	58.30%

Table 5: Evaluation with conventional BoW-based classification method

Method	AUC	pAUC	$TPR_{0.5\%}$
Proposed	0.9306	0.0825	65.30%
BoW	0.9030	0.0657	32.00%

method consumes most of CPU time for compression process and its time is longer than any other process of BoW method. Still, once compression is completed, feature generation, training and detection are finished with less CPU time than BoW method. As for memory usage, proposed method consumes small memory for compression process and less memory for feature generation, training and detection compared with BoW method. Although BoW method generates one-hot vector for every single word appeared in URL so that memory usage tends to increase, proposed method generates compression algorithm feature vector in several dimensions so that memory usage does not steeply increase.

6 CONSIDERATION

We consider the reason the compression algorithm feature contributes to better classifying malicious and legitimate logs. Figure 4 shows the histogram of Z_{pos} of URL attributes for both malicious and legitimate logs, where the red and blue zone are histograms of malicious and legitimate logs, respectively. The histogram of malicious logs contains three peaks: A) the compression rate is very small, B) the compression rate is as high as that for legitimate logs, and C) the compression rate is very high.

A sample URL that belongs to pattern A is shown in table 8. For security reason, FQDN is masked with 'www.example.com' and QueryString values are masked with meta words. The first row shows the original URL string and its length, the second row shows the LZT compressed state and relative entropy with malicious logs, and the third row shows that with legitimate logs, where '|' shows that data sequences between '|' marks are expressed in 1 code. In compressing with malicious logs, the table shows that a 1,352-bit-long URL is compressed to 220 bits and many data sequences are expressed as 1 code. Especially in QueryString of URL, almost one key (e.g. "dstid=1") or one combination of a key and value (e.g. "countryid=...") is compressed as 1 code. This observation suggests that a QueryString key and

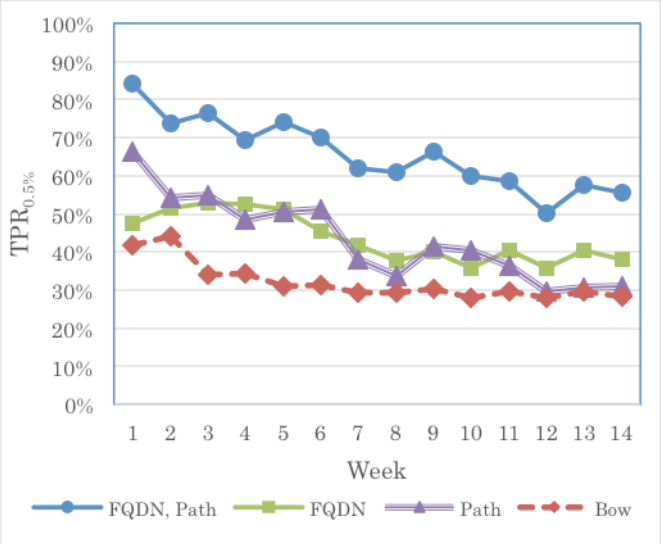

Figure 3: *TPR* deterioration over time

Table 6: CPU Time Comparison with Conventional Method (seconds)

Method	Compress	Generate Feature	Train	Detect
Proposed	13,809	3,752	22	438
BoW	-	7,145	950	408

value combination that exists in the training dataset is automatically recognized and compressed as 1 code. In contrast, a key and value combination that does not exist in the training dataset is automatically split. This is one use case that QueryString key exists but its value is modified in malware communication.

Other examples of pattern A for the FQDN attribute are FQDNs having sequential numbers in host names such as host1.example.com and host2.example.com. These FQDNs are recognized as totally different strings by exact matching, but in the compression algorithm that has the characteristic of longest matching, two FQDNs are recognized as similar strings. In fact, host2.example.com is compressed as |host|2.|example.com| after training data host1.example.com. This is another use case that FQDN is partially modified to similar FQDN.

URLs belongings to pattern B tend to have same FQDNs existing both in malicious URLs and legitimate URLs. Since Path of these URLs are different, compression rate does not get so small against both malicious logs and legitimate logs and makes detection difficult.

A typical URL belongings to pattern C is shown in table 9. This URL has an encoded or encrypted string. The second row shows compression results of the URL. It shows that the compression rate becomes large for both malicious and legitimate logs and makes classification difficult.

From this consideration, the proposed method is capable to detect malicious URL strings that are similar

Table 7: Memory Usage Comparison with Conventional Method (MB)

Method	Compress Generate Feature	Train	Detect
Proposed	2,401.0 15,999.8	4.7	424.4
BoW	- 56,988.2	1,241.2	529.4

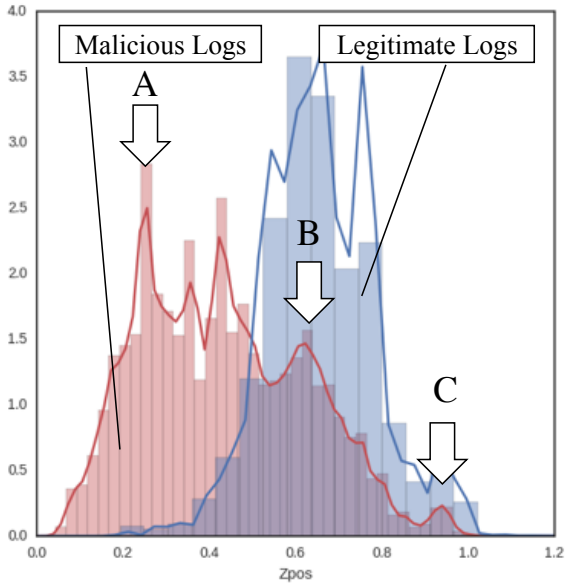

Figure 4: Histogram of Z_{pos}.

to but slightly different from existing malicious URLs. Of course, attacker can totally change URL strings from past attack vectors, in this case, proposed method does now work well. However, assuming that many attackers tend to use existing attackers' tool kit to set up their attack vectors and these tools are not so often drastically modified, proposed method should still be feasible.

7 CONCLUSION

We proposed a novel method for detecting malicious communication of infected hosts by generating a compression algorithm feature of URL attributes and classifying with supervised learning. Through evaluation, we demonstrated that the proposed method has higher detection capability than the conventional BoW-based detection method. In particular, its *TPR* in a low *FPR* area (0.5%) is over 30% higher than that of the BoW-based method. In addition, we clarified how the compression algorithm works in classification and demonstrated a real use case in which the proposed method detected malicious URL strings that are similar to but slightly different from existing malicious URLs.

REFERENCES

[1] https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_20162017.pdf, "ECURITY REPORT 2016/17", AV-TEST, (2017).

Table 8: Effectively classified URL and compression state

Uncompressed URL: 1352bit:
http://www.example.com//offers/DynamicOfferScreen?offerid=foo&distid=bar&leadp=baz&countryid=qux&sysbit=quux&dfb=corge&hb=grault&isagg=garply&version=waldo&external=fred&external=plugh&
Relative Entropy for Legitimate Logs: 900bit
|http://www.example.com/|/of|fers|/D|yna|mic|Off|erS|cre|en|?o|ffe|rid=foo|&dis|tid=b|ar|&le|adp|=ba|z&c|ountry|id=qu|x&s|ys|bit|=quux&|dfb|=corge&|hb=|grault&is|agg|=garply&ver|sion=|waldo|&ex|ternal|=fred&e|xter|nal|=plugh&|
Relative Entropy for Malicious Logs: 220bit
|http://www.example.com//offers/DynamicOfferScreen?offerid=foo|&distid=b|ar|&leadp=baz&|countryid=qux&sysbit=quux&df|b=corge&hb=grault|&isagg=garply&versio|n=wal|do&e|xternal=fred&e|xternal=plugh&|

Table 9: Poorly classified URL and compression state

Uncompressed URL: 4688 bits
http://www.example.com/api/vp/1?clk=gLg_PHWA9aSyioXkt-F4b3J9cI1ybf-t-x7VxWH5dmAWXwln-z ...(omit)
Relative Entropy for Legitimate Logs: 4420bit
|http://www.example.com/api/|vp|/1|?cl|k=|gLg|_PH|WA9|aS|yio|Xkt|-F4|b3J|9cI|1y|bf-|t-|x7|VxW|H5d|mAW|Xwl|n-|z⋯(omit)
Relative Entropy for Malicious Logs: 4980bit
A|http://www.example.com/api/v|p/1|?cl|k=|gL|g_|PH|WA|9a|Sy|io|Xk|t-|F4|b3|J9|cI|1yb|f-|t-x|7V|xWH|5d|mA|WX|wl|n-|z⋯(omit)

[2] D. Benedetto, E. Caglioti, and V. Loreto, "Language trees and zipping", Phys. Rev. Lett., vol.88, (2002).
[3] E. Keogh, S. Lonardi, and C. A. Ratanamahatana, "Towards Parameter-Free DataMining", KDD, pp. 206-215, (2004).
[4] Y. Marton, N. Wu, and L. Hellerstein, "On compression-based text classification", European Conference on Information Retrieval, pp. 300-314, (2005).
[5] A. Bratko, G. V. Cormack, B. Filipič, T. R. Lynam, and B. Zupan, "Spam filtering using statistical data compression", Journal of Machine Learning Research, vol.7, pp.2673-2698, (2006).
[6] K. Nishida, R. Banno, K. Fujimura, and T. Hoshide, "Tweet-Topic Classification using Data Compression", DBSJ Journal, Vol.10, No.1, pp.1-6, (2011).
[7] H. Adachi, M. Okabe, and K. Umemura, "Recognition of Music Composer with Compression-based

Dissimilarity Measure”, DEIM2013.
[8] M. Dredze, K. Crammer, and F. Pereira, “Confidence-weighted classification”, Proceedings of 25th International Conference on Machine Learning, pp.264–271, (2008).
[9] A. Kumagai, Y. Okano, K., and M. Tanikawa, “Supervised Classification for Detecting Malware Infected Host in HTTP Traffic and Long-time Evaluation for Detection Performance using Mixed Data”, IEICE-ICSS, Vol.116, No.522, pp.43-48, (2016).
[10] T. Nelms, R. Perdisci, and M. Ahamad, “ExecScent: mining for new C&C domains in live networks with adaptive control protocol templates”, 22nd USENIX Conf., pp.589–604, Aug. (2013).
[11] K. Bartos, M. Sofka, and V. Franc, “Optimized invariant representation of network traffic for detecting unseen malware variants”, in USENIX Security Symposium, pp. 807–822, (2016).
[12] K. Aoki, T. Yagi, M. Iwamura, and M. Itoh, “Controlling malware http communications in dynamic analysis system using search engine”, Cyberspace Safety and Security (CSS), 2011 Third International Workshop onIEEE, pp.1–6 (2011).
[13] Y. Okano, A. Kumagai, M. Tanikawa, Y. Oshima, K. Aiko, K. Umehashi, and J. Murakami, “Proposal of selection of training data using misdetected goodware for preventing misdetection of a static detector of malware”, IEICE-PRMU, Vol.115, No.224, pp.163-170, (2015).
[14] L. E. Dodd and M. S. Pepe. “Partial AUC estimation and regression”, Biometrics, 59(3), pp.614–623, (2003).
[15] P. Tischer, “A modified Lempel-Ziv-Welch data compression scheme”, Aust. Comp. Sci. Commun. 9, 1, pp.262-272, (1987).

(Received November 28, 2019)
(Revised March 30, 2020)

Yasushi Okano He received a Master degree in Interdisciplinary Environmental Science from Kyoto University in 1995, and presently a senior research engineer at NTT Secure Platform Laboratories. He works on network and IoT security researches.

Kazunori Kamiya He received a Master Degree in Frontier Science from the University of Tokyo in 2004, and presently a senior research engineer at NTT Secure Platform Laboratories. He works on network security and network operation researches.

Atsutoshi Kumagai He received the B.S. degree in informatics and mathematical science form Kyoto University in 2010, and the M.S. degree in informatics from Kyoto University in 2012. In 2012, he joined NTT and is currently a researcher of NTT Software Innovation Center and NTT Secure Platform Laboratories, Tokyo, Japan. His research interests include machine learning, data mining, and cyber security.

Taishi Nishiyama He received his bachelor's degree in Undergraduate Course Program of Aeronautics and Astronautics from Kyoto University in 2014, and master degree in Aerospace Engineering from University of Tokyo in 2016. He has worked in machine learning and network security at NTT Secure Platform Laboratories.

Bo Hu Bo Hu received an M.S. in wireless network engineering from Osaka University in 2010 and joined NTT the same year. He has mainly been engaged in researching network security technology, machine learning and inter-cloud technology. He developed a security orchestration architecture, a machine learning pipeline for large-scale traffic analysis, and inter-cloud protocols. He has also worked on cloud computing standardization activities in the Telecommunication Standardization Sector of the International Telecommunication Union and other standardization organizations.

Masaki Tanikawa He received a Master degree in Systems Science from Tokyo Institute of Technology in 1995, and presently a senior research engineer, supervisor at NTT Secure Platform Laboratories. He works on network security and network operation researches.

Kazuhiko Ohkubo Kazuhiko Ohkubo is the CISO of Kyowa Exeo Corporation. He received his M.S. in electrical engineering from the University of Tokyo in 1989. He received his Ph.D. in business administration and computer science from the Aichi Institute of Technology in 2019.He is a member of IEEE.