Electronic Preprint for Journal of Information Processing Vol.25

Regular Paper

Detection and Filtering System for DNS Water Torture Attacks Relying Only on Domain Name Information

Takuro Yoshida[1,a)] Kento Kawakami[2] Ryotaro Kobayashi[3] Masahiko Kato[4]
Masayuki Okada[5] Hiroyuki Kishimoto[6]

Received: December 5, 2016, Accepted: June 6, 2017

Abstract: Water torture attacks are a recently emerging type of Distributed Denial-of-Service (DDoS) attack on Domain Name System (DNS) servers. They generate a multitude of malicious queries with randomized, unique subdomains. This paper proposes a detection method and a filtering system for water torture attacks. The former is an enhancement of our previous effort so as to achieve packet-by-packet, on-the-fly processing, and the latter is an application of our current method mainly for defending recursive servers. Our proposed method detects malicious queries by analyzing their subdomains with a naïve Bayes classifier. Considering large-scale applications, we focus on achieving high throughput as well as high accuracy. Experimental results indicate that our method can detect attacks with 98.16% accuracy and only a 1.55% false positive rate, and that our system can process up to 7.44 Mpps of traffic.

Keywords: DNS, DDoS, IPS, water torture attacks, pseudo-random subdomain attacks, naïve Bayes classifier

1. Introduction

A key component of the Internet is the Domain Name System (DNS), which many services, including the World Wide Web (WWW) and electronic mail, depend on. The most important function of this system is name resolution, which refers to conversion between human-memorable *domain names* and *IP addresses* necessary to establish connections. Owing to its significance, the DNS is a common target of cyber attacks.

Water torture attacks [1] are a type of Distributed Denial-of-Service (DDoS) attack that aims to exhaust computational resources, such as processors, memory, and collateral network bandwidth capacity, by overwhelming DNS servers via a huge number of random subdomain queries. Authoritative servers are generally targeted, but secondary damage can be caused to recursive servers relaying malicious queries.

The goal of this study is to provide a detection method for water torture attacks together with a working prevention system. During development, we expect the large-scale application of this system. Such a system needs to detect and prevent malicious queries sufficiently quickly and robustly under immense traffic during attacks.

Several detection methods, including our previous one, have been proposed. However, to the best of our knowledge, there have been no practical prevention systems that selectively block malicious queries. A further novel feature of our study is the empirical testing of not only accuracy, but also throughput in a near-real-world environment.

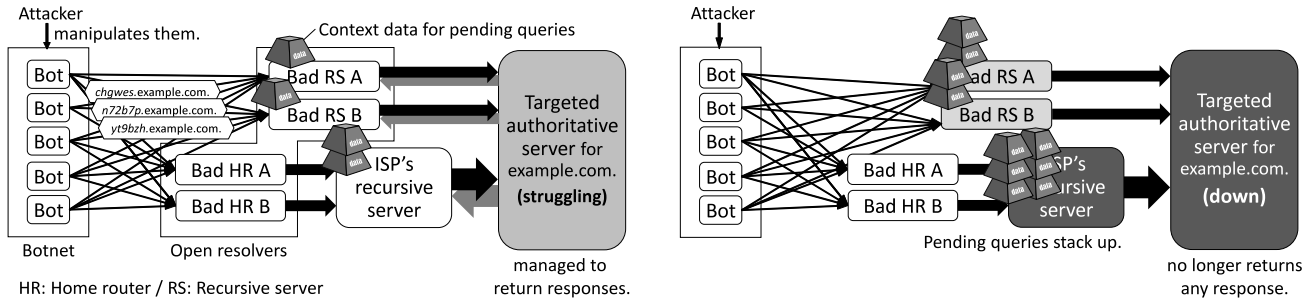
This study focuses on recursive servers. This is because secondary damage suffered by recursive servers is becoming a crucial and urgent problem, especially among network operators who manage massive servers accessed by many users. Furthermore, prevention closer to the source of malicious queries, namely at recursive servers, leads to reduced damage to authoritative servers as well. Thus, recursive-server-side detection and prevention systems are needed and meaningful.

This study is an expansion of our previous study [2] [*1]. Continuing to utilize the potential of the naïve Bayes classifier, we upgraded the detection method in terms of feature selection to achieve better accuracy. Furthermore, our previous detection system was re-designed to become a prevention system.

The rest of this paper is organized as follows. Section 2 introduces the mechanism of water torture attacks and possible mitigation measures for them. Section 3 reports work related to water torture attacks and other similar DNS threats and outlines the differences from our previous study. Sections 4 and 5 describe the rationale and implementation of our detection method and filtering system, respectively. Section 6 clarifies the evaluation methods and environment of this study and its experimental results.

[1] Graduate School of Engineering, Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[2] Faculty of Engineering, Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[3] Faculty of Informatics, Kogakuin University, Shinjuku, Tokyo 163–8677, Japan
[4] Department of Information Security, University of Nagasaki, Nishisonogi, Nagasaki 851–2195, Japan
[5] Japan Network Information Center, Chiyoda, Tokyo 101–0047, Japan
[6] Comworth Co., Ltd., Ota, Tokyo 102–0071, Japan
[a)] yoshida2016@ppl.cs.tut.ac.jp

[*1] We previously published preliminary results [3] of our current study.



(a) First phase (an attacker's purpose)

(b) Second phase (the collateral effect)

Fig. 1 Two phases of a water torture attack.

Based on this, Section 7 discusses the effectiveness and limitations of our method and system. Finally, Section 8 concludes this paper.

2. Water Torture Attacks

2.1 Background and Mechanism

Water torture attacks, also known as DNS slow-drip attacks or (pseudo-)random subdomain attacks, were first reported in 2009 [4] and became widely recognized in early 2014 [1], especially among operators of large recursive servers. The fundamental concept of a water torture attack is quite simple: even if each machine sends only a few queries, ultimately the snowballing number of queries will relentlessly *torture* and collapse the targeted server.

Typically, queries for frequently appearing domains seldom reach authoritative servers because intermediate recursive servers often have cached previous data for the domains and can return responses from their caches. Only when the recursive servers do not have cached data does the query reach the appropriate authoritative servers. In contrast, queries with randomized unique subdomains may pass straight through to the authoritative servers.

Now, assume that an attacker is launching an attack on some authoritative server. We consider an attack to have two phases, illustrated in **Fig. 1**. First, the attacker organizes a *botnet*, a group of remotely manipulated machines called *bots*, to be ready to begin an attack. Then, the bots send malicious queries through open resolvers, which are improperly configured, open-access recursive servers. Note that each query in a water torture attack appears normal and the amount that each bot sends is moderate, making it difficult to distinguish malicious queries from legitimate queries. One of their few characteristics is that they include random subdomains such as

ckyx5yxrkkp9.example.com

to ensure they reach the targeted authoritative server.

Although the traffic that each bot generates is not intensive, the cumulative number of queries toward the targeted server becomes a tremendous flood, eventually overloading the server until it becomes unresponsive. This is the attacker's primary objective but is only the first phase of the attack, as shown in Fig. 1 (a).

After the targeted authoritative server is exhausted, there is a domino effect on relaying recursive servers. If a recursive server receives a query for an uncached domain from a client, then the server repeats this until it finally obtains the sought-after information, which involves asking an appropriate authoritative server and waiting for a response until a timeout expires. While waiting, the recursive server needs to keep track of the contextual data of the connection. If the requested authoritative server becomes unresponsive but the bots continue to send queries, then the contextual data stacks up in the recursive server, gradually consuming resources and, finally, making the recursive server also unresponsive as a result of overload. This is the second phase of the attack, as shown in Fig. 1 (b). If this collateral damage strikes large recursive servers, it could be a considerable disaster for many Internet users, highlighting the need for recursive-server-side prevention.

2.2 Open Resolvers as Accidental Accomplices

Recursive servers are generally set up for a limited number of users, usually on the same network, and should not respond to external users. Open resolvers are improperly configured, open-access recursive servers. They cause vulnerability to many types of DNS threats, including water torture attacks and amplification attacks [5].

There are roughly two types of open resolvers: badly configured recursive servers and home routers that are vulnerable or manufactured with improper initial settings. Although both act like public recursive servers from the outside, their inner behaviors differ. Badly configured recursive servers are genuine recursive servers, which means that they process recursive queries step-by-step from a root server. In contrast, home routers with improper settings do not execute that procedure; instead, they request another server to do it since they are mere DNS forwarders, not complete recursive servers. The substitute server is usually a server managed by an Internet service provider (ISP) with which an owner of a router contracts. Bad routers therefore serve as a proxy for their upstream servers. Eventually acting like open resolvers, an ISP's servers suffer the collateral damage of water torture attacks, though the servers themselves are closed to the public.

2.3 Mitigation Measures

Attacks exploit the specification of the DNS itself, and thus complete prevention by simple rules is difficult. Thus, we explain some measures for mitigation of such attacks. Note that the following description is from the perspective of defending recursive servers.

One recursive-server-side mitigation measure for water torture

attacks is *Recursive Client Rate Limiting* [6] [*2]. This function restricts the maximum number of outstanding queries per domain and/or server. Even if a recursive server receives a huge number of queries toward a certain authoritative server, the recursive server relays only a limited number of them; the remaining queries are treated as server errors. If this rate limiting approach is unavailable for some reason (e.g., when using an older version of server software), then another possible measure would be to clarify the targeted domain and configure a server to deny queries about it.

With these measures, the load on the recursive server is suppressed to a certain extent, but it can also obstruct legitimate users from fetching information that the attacked authoritative server has. This essentially means that the attack has achieved its primary objective of making the targeted server difficult to access or, at worst, completely inaccessible.

We believe that the subject to be denied should be the incoming malicious queries themselves and not the victim domain and the server. Therefore, to form an effective solution, we consider that measures for water torture attacks should focus on the quality of each query, rather than situational information like the number of pending queries and the frequency of requests toward specific servers.

3. Related Work

3.1 Detection of Water Torture Attacks

To the best of our knowledge, currently existing measures do not aim to identify which queries are the cause. Rather, they try to recognize occurrences of attacks by observing changes in communication conditions or in the servers. For example, Alonso et al.'s study [7] treats water torture attacks as a type of flooding attack. The main purpose of their approach is to provide a detection method for such attacks toward recursive servers. Their method observes changes in correspondence relations between IP addresses and domain names in DNS traffic, and carries out detection using a ν-support vector machine (SVM) classifier. Their method performs well for detection but not for prevention. That is, occurrences of attacks can be identified, but the queries causing the attack cannot. In contrast, our method can perceive both occurrences of attacks and the queries that cause them, and thus it is suitable for a prevention system.

Cloudmark, Inc. provide a function to mitigate the damage caused by a huge number of queries with randomized subdomains exhausting server resources [8] [*3]. Their method observes changes in tendency of some factors for every domain, such as query-vs-response and success-vs-failure rates. Although sophisticated, it is still a type of per-domain rate-limiting method and so it has the problem of potential obstruction of legitimate users trying to access domains being attacked.

[*2] The most popular implementation of recursive client rate-limiting is *BIND*, a major DNS server software. This function is available on versions 9.9.8 and above.

[*3] Cloudmark, Inc. call these types of attacks *resource exhaustion*. We consider water torture attacks to be a type of resource-exhaustion attack.

3.2 Detection of Automatically Generated Words

Fake account names used for spamming on social network sites are typically generated from random strings or from a dictionary. Freeman's study [9] utilized a naïve Bayes classifier with n-grams of letters to reveal such fake accounts. Although we use similar techniques in our study, we also employ additional features regarding the DNS and observe that our method works well for water torture attacks.

Similar to water torture attacks, Kaminsky attacks also generate queries with random subdomains to the targeted recursive server. Matsubara et al. [10] proposed two detection models for Kaminsky-like attacks, both calculating a distance between two consecutive queries. One uses the Euclidean distance of IPv4 addresses, each of which is split into four 8-bit parts and treated as a four-dimensional vector, while the other uses the restricted Damerau-Levenshtein distance, a type of edit distance, between domain names of two consecutive queries. This study is of interest, but did not report details of the accuracy of the results.

Domain names yielded by domain generation algorithms (DGAs) are frequently seen in DNS traffic, and these are used for botnet communication. Yadav et al. proposed a detection method for these domains that combined multiple metrics related to distributions of alphanumeric characters [11]. Schiavoni et al.'s method [12] for detecting DGA-based domain names utilizes a combination of dictionary- and n-gram-based approaches.

Ideas for the detection of generated words should also be applicable for water torture attacks. However, such ideas proposed in earlier studies have mainly focused on accuracy while paying less attention to processing speed, because the targeted threats investigated in these studies, such as Kaminsky attacks and botnet communication, do not induce a large number of traffic. On the other hand, as water torture attacks are a type of DDoS attack, a detection mechanism for them requires sufficient capacity to process a huge number of queries instantly. Our method was, accordingly, developed with a deep consideration for processing speed and tested as a practical prevention system. We thus consider our method as being suitable for water torture attacks and this differentiates it from methods for the mere detection of random words.

3.3 Our Previous Study

Our previous effort [2] and the current study, each of which provide a method and a system, aim to detect water torture attacks based on a naïve Bayes classifier. We place a strong emphasis on accomplishing not only high accuracy but also high throughput to make our system practical for real-world situations.

3.3.1 Conceptual Difference

The biggest difference between this study and our previous study is a fundamental notion of how to cope with attacks. In general, victims of DDoS attacks have limited means to withstand them and thus should cooperate with ISPs; in particular, they are completely helpless against bandwidth saturation attacks without others' help. We therefore considered it important to devise an initial detection method with the actual prevention performed at the higher-level network, according to the information retrieved with the method.

However, water torture attacks are not a type of amplification attack. That is, the bandwidth capacity exhausted by an attack does not matter. Instead, the source of the intensity is how many packets are sent to the targeted server, namely, how much processor and memory resources are consumed in name resolution procedures [*4]. We thus constructed an at-the-front filtering system as an effective way of suppressing water torture attacks.

The previous method had a great disadvantage with respect to constructing such a system: it provided only delayed detection. This meant that a system based on the method would take 10 minutes to detect an attack after a malicious query came. This delay resulted from the existence of the *time window* used for calculating the number of occurrences of the same subdomain. The method was thus not directly applicable to an on-the-fly prevention system, in contrast to our new system.

Another difference is that the previous study targeted the authoritative server whereas the current study targets the recursive servers. Recursive servers relaying malicious queries suffer from secondary damage, and there is therefore a demand to shield recursive servers, especially large-scale ones provided by ISPs. This study aims to meet this demand. Note that our current system is designed for recursive servers, but does not depend on them. The system would still be effective for authoritative servers, although this is out of the scope of the current study.

Due to the differences in approach and implementation, our previous and current methods are not comparable in identical conditions. The comparisons of the accuracy given in Section 6 are thus for reference only. In addition, since the functions of the previous and current system are completely different, we do not compare the two systems in this paper. Whereas the current system detects and then filters malicious queries, the previous system only detects them.

3.3.2 Differences in Implementation

A major difference from the previous method is that the current method analyzes queries instead of responses in order to be able to filter queries. Hence, we revised the selected features, detailed in Section 4.3, for a naïve Bayes classifier.

The previous method took the *number of occurrences* of the same subdomain within a certain period of time as a feature, but this impeded implementation of real-time detection. The current method no longer uses this feature and instead instantly analyzes each query and judges whether it is randomly generated. This enables us to employ the method to construct a filtering system.

The feature of the label count was also deleted. Our new method only focuses on the leftmost label, and thus the number of labels is obviously meaningless.

We still use bigram-based features in this study but changed the method of calculation. Whereas the previous method used the sum of the probabilities of occurrence of each bigram, the current method uses their product with Lidstone smoothing. This yields a higher accuracy that is sufficient to compensate for the loss resulting from the disuse of the previously used time-window feature. Moreover, this change allows our method to outperform the previous one in terms of accuracy; in particular, the number of false positives significantly decreases.

[*4] Secure64 reports: *As a side effect, our service provider customers are seeing a spike in DNS traffic resulting in increased CPU and memory usage* [1].

4. Detection Method

4.1 Approach

Our method judges whether each subject is a water torture attack based on the presence or absence of a randomly generated subdomain. This decision relies on a naïve Bayes classifier, which we chose for its advantages in accuracy and speed.

Naïve Bayes classifiers are well tested and famous for their high accuracy despite their simplicity and computational efficiency [13], [14], especially in the field of text classification [15] (e.g., e-mail spam filtering [16]). For handling short texts, the accuracy of naïve Bayes classifiers outperforms that of the SVM [17]. Moreover, implementations accelerated using a field-programmable gate array (FPGA) and general-purpose graphics processing unit (GPGPU) have already been developed [18], [19].

4.2 Multinomial Naïve Bayes Classifier

Naïve Bayes classifiers are a type of supervised learning algorithm based on Bayes' theorem as the name indicates. When given a set of features of a subject, a classifier classifies subjects according to estimated posterior probabilities. In this study, we chose the multinomial model as an event model for our classifier because it works well for text classification compared with other models such as Bernoulli and Gaussian models [15], [20], [21].

Upon classification, we assume two classes: C_0 for non-random subjects and C_1 for random subjects. Given a class variable C_k and a feature vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, a probability $P(C_k|\boldsymbol{x})$ that a subject whose features are expressed as $\boldsymbol{x}$ belongs to C_k, is calculated by Bayes' theorem.

$$P(C_k|\boldsymbol{x}) = \frac{P(C_k)P(\boldsymbol{x}|C_k)}{P(\boldsymbol{x})}. \tag{1}$$

Assuming that each feature is independent of each other, this becomes:

$$\begin{aligned} P(C_k|\boldsymbol{x}) &= \frac{P(C_k)\prod_i P(x_i|C_k)}{P(\boldsymbol{x})} \\ &\propto P(C_k)\prod_i P(x_i|C_k). \end{aligned} \tag{2}$$

The more probable class $\hat{C}$ is determined to be the class with the higher probability:

$$\hat{C} = \arg\max_{k\in\{0,1\}} P(C_k)\prod_i P(x_i|C_k). \tag{3}$$

Note that since $P(\boldsymbol{x})$ is constant within a single input, $P(\boldsymbol{x})$ does not need to be calculated to compare probabilities of different classes. As the equation above states, we need to seek a class probability $P(C_k)$ and a conditional probability $P(x_i|C_k)$, which we estimate according to the maximum *a posteriori* (MAP) estimation. In the multinomial model, parameter estimates $\boldsymbol{\theta_k} = (\theta_{k0}, \ldots, \theta_{kn})$ are obtained by:

$$\theta_{ki} = \frac{N_{yi} + \alpha}{N_y + \alpha n}, \tag{4}$$

where N_{yi} is the number of occurrences of the i-th feature and N_y is the total number of all features for the class C_k in a training set.

The probability $P(C_k)$ can also be estimated in a similar manner, as it is the quotient of the number of items belonging to the class C_k and the total number of items in a training set.

In Eq. (4), α is a smoothing factor of Lidstone smoothing, which significantly affects accuracy. This smoothing technique is required to avoid the so-called zero-frequency problem. Without this technique, once a single subject has any unknown features not in a training set, an estimated probability falls to zero. If probabilities of both classes become zero, the classifier cannot classify such a subject at all. This behavior apparently decreases overall accuracy and should be avoided.

4.3 Feature Selection

In classification, our classifier only focuses on the leftmost label of analysis subjects, and extracts the following two types of features from the label:

Length-based feature The length of the leftmost label.

Bigram-based feature Each bigram of letters of the leftmost label.

Since random subdomains require a certain degree of length to be unique, we consider the feature of length to be suitable. To avoid oversensitivity to rare cases, we set a cutoff point to this feature. This is the value determined to yield the best accuracy in experiments.

A bigram of letters, hereafter called simply a bigram, is two consecutive letters in a string. Furthermore, for more precise classification, strings are regarded to have phantom letters that signify the head and the tail. For example, the word *water* can be decomposed into six bigrams: *^w, wa, at, te, er, r$*, where ^ is a phantom head-of-string letter and *$* is a phantom tail-of-string letter. Bigrams are reported to be an effective factor for short-text classification [9], and our experiments indicate that employing phantom letters produces higher accuracy.

For example, a domain *mail.example.com.* has the following seven features:

- 4 letters in the leftmost label and
- The following of five bigrams, each considered a feature: *^m, ma, ai, il, l$*.

The classifier refers to frequency tables, created by learning from a training dataset, for the relative frequencies of these seven features and estimates probabilities for both random and non-random cases.

5. Filtering System

5.1 Overview

As **Fig. 2** shows, this system is assumed to be placed in front of a recursive server, and it censors all packets to the server and only relays legitimate queries and packets other than DNS queries. That is, all malicious queries are dropped. To be more client-friendly, the system should return responses signifying a server error when dropping malicious queries. However, in the current implementation, it does not do so because of limitations in the employed network capture card with which we built our system.

5.2 Software and Hardware Specifications

Table 1 shows the specifications of the filtering system that we developed and examined. What we should point out is that our system utilizes a hardware-accelerated network card that can capture, process, and transmit a significant number of packets, up to 10 Gbps, instantly by using APIs specially designed for the card. The card and APIs provide a special way to receive and transmit packets totally different from sockets provided by the Linux OS and do not generate interruptions to processors in capturing, which enables far faster packet I/O than ordinary network interface cards and standard APIs. Since we hoped that our system could be used along with large-scale recursive servers, this advantage of the card is exactly what we needed, and thus we employed it.

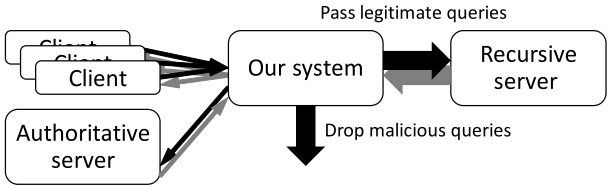


Fig. 2 How the system works.

Table 1 Software and hardware specifications of the fitering system.

OS	Ubuntu 14.04.1 (64-bit version)
Compiler	gcc 6.2.0
Processor	2 x Intel Xeon CPU E5520 (2.26 GHz, 4 cores)
Memory	16.0 GB
Network capture card	Fiberblaze fbC2XGhh [22] (Network card specialized for packet capturing)

5.3 Implementation

The filtering program running on the system was written in the C language and compiled with gcc version 6.2.0. Packet I/O depends on the fbCAPTURE framework (APIs provided by the manufacturer of the network card), and other parts of the program, including packet analysis and classification, do not need any libraries except for APIs provided by the OS.

A step-by-step explanation of the behavior of the system follows. First, before running as a filtering system, the system must undergo learning. This learning stage involves creating frequency tables for random and non-random subjects from a training dataset. Then, the system waits for queries and classifies them using the proposed method as they arrive. The classification step consists of acceptance of a set of features extracted from a received query, probability estimation for both cases, and finally comparison of the two probabilities. Only if the classifier judges the query to be not malicious does the system pass it to a recursive server.

In probability estimation, all intermediate calculations are performed in a logarithmic scale, which has advantages for both accuracy and speed. The main reason is to avoid underflows. Estimated values, represented by a floating-point type, tend to become too big or too small, which linear-scale multiplication cannot precisely handle. A secondary benefit is the speed of classification. Multiplication is converted to addition, which is a faster calculation to process. To avoid repeated linear-to-logarithmic conversions and *vice versa*, the frequency tables store probabilities in a logarithmic scale. The classifier thus does not need to convert calculations during the classification step.

5.4 Whitelisting for Random but Legitimate Domain Names

To make our filtering system more practical and feasible in the real world, the system should have a fail-safe function not to block legitimate queries. For that, we consider a whitelisting function. This is not related to the rationale of our detection method, the naïve Bayes classification, and therefore we treat the function as a supplement.

There are several cases that some domain name is judged to be random by the classifier but is actually legitimate and not related to water torture attacks. They include internationalized domain names (IDNs) and ones used by contents delivery services (CDNs). In IDNs, multilingual (Unicode) characters are converted to ASCII characters in accordance with the rule named Punycode [23], and converted domain names themselves usually look random. (e.g., xn–eckwd4c7cu47r2wf.jp) In case of CDNs, many domain names are assigned for one service for the sake of load balancing, and those domain names sometimes include random-looking, or serial, numbers (e.g., *foobar-elb-251771428.ap-northeast-1.elb.amazonaws.com.*) In order that the system does not filter these domain names out, it should be desirable not to operate the detection and filtering mechanism in case a queried domain name is such a domain name.

There are domain names that tend to have random-looking subdomain names, and thus our approach focuses on the second leftmost label of a domain name. Prior to analyzing a query, the system checks whether the second leftmost label of the queried domain name is on a whitelist, and if so, the query is soon judged to be non-malicious without the naïve Bayes classification. The whitelist is a list of second leftmost labels whose neighbor on the left tends to be random, and it is manually prepared beforehand. For the sake of performance, the whitelist lookup is implemented by utilizing the binary search algorithm.

The reason that the system analyzes only the second leftmost label instead of all labels except the leftmost is to achieve acceptable throughput.

We made a whitelist from the captured items in the datasets used for the accuracy evaluation (Section 6.1.2). We extracted second leftmost labels from items whose desired output is random but legitimate in the datasets, and obtained the number of occurrences for each label. **Table 2** shows the top 10 frequent labels that tend to have a random label as a child. According to our own investigation, 35 items are needed on the whitelist for 90% coverage of random items. Similarly, 89 items for 95%, and 427 items for 99%.

Table 2 Top 10 frequent second leftmost labels that tend to have a random label as a child.

Label	Domain owner	Frequency (%)	Cumulative frequency (%)
www	- [*5]	35.74	35.74
ap-northeast-1	Amazon.com, Inc.	9.68	45.42
cloudfront	Amazon.com, Inc.	8.15	53.58
us-east-1	Amazon.com, Inc.	5.18	58.76
metric	Google Inc.	4.78	63.53
googlevideo	Google Inc.	3.56	67.10
openresolvertest	(unidentified)	3.38	70.48
ns	- [*5]	2.11	72.58
us-west-2	Amazon.com, Inc.	1.83	74.42
fc2	FC2, Inc.	1.59	76.00

[*5] Some websites use domain names such as *srv01.www.example.com., srv02.www.example.com., and srv03.www.example.com. The label ns is also the same case.*

6. Evaluation

We evaluated the performance of our contributions through two types of evaluation: accuracy evaluation for the detection method and throughput evaluation for the filtering system.

6.1 Accuracy Evaluation

This section clarifies the details of the accuracy evaluation that targets our detection method.

6.1.1 Metrics

Our primary metric for evaluating the performance of the classifiers is accuracy. We also regard false positive rates (FPR) as important. A good classifier must yield not only high accuracy but also low FPRs.

The two metrics are defined as:

$$Accuracy = \frac{C_{TP} + C_{TN}}{C_{TP} + C_{TN} + C_{FP} + C_{FN}}, \quad (5)$$

$$FPR = \frac{C_{FP}}{C_{FP} + C_{TN}}, \quad (6)$$

where C_{TP}, C_{TN}, C_{FP}, and C_{FN} stand for the counts of true positives, true negatives, false positives, and false negatives, respectively. *True* subjects are ones whose classification outcomes and desired outputs correspond, while *false* ones are those where they do not. *Positive* subjects are ones determined to be malicious and *negative* ones are those found to be legitimate.

6.1.2 Datasets

To the best of our knowledge, there are currently no public datasets dedicated to water torture attacks. We thus created four datasets that simulated such attacks in order to conduct this evaluation. These datasets can be used for both training and testing. As **Table 3** shows, each item in the datasets is a set of values for each feature and a desired output, namely whether this item is random or not. Each item is either *captured* from actual traffic or *generated* by a program, and as **Table 5** shows, the proportion of the two types in a single dataset is the same.

The four datasets differed in the acquisition dates of the cap-

Table 3 Example of a dataset.

Leftmost label		Is this random?
	Length	
mail	4	No
www	3	No
www	3	No
7rizyrfkde	10	Yes
594hhag88gx22	13	Yes
...		

Table 4 Example of a dataset (modified to neutralize the detection avoidance).

Leftmost label		Is this random?
	Length	
mail	4	No
www	3	No
www	3	No
www7rizyrfkde	13	Yes
www594hhag88gx22	16	Yes
...		

Table 5 Constituents of the datasets.

Dataset No.	Captured		Generated	Total
	Non-random	Random	Random	
1	3,390,068 (43.53%)	503,495 (6.47%)	3,893,563 (50.00%)	7,787,126 (100.00%)
2	3,533,760 (42.99%)	576,618 (7.01%)	4,110,378 (50.00%)	8,220,756 (100.00%)
3	3,361,443 (43.14%)	534,404 (6.86%)	3,895,847 (50.00%)	7,791,694 (100.00%)
4	1,666,058 (42.23%)	306,460 (7.77%)	1,972,518 (50.00%)	3,945,036 (100.00%)

tured parts of the items. The duration for capturing were all 24 hours but the starting times were different. For each of the entries in the column *Dataset No.* in Table 5, the starting times are:

(1) Wednesday, June 8, 2016 at 0:00:00 a.m. JST
(2) Thursday, June 9, 2016 at 0:00:00 a.m. JST
(3) Friday, June 10, 2016 at 0:00:00 a.m. JST
(4) Saturday, June 11, 2016 at 0:00:00 a.m. JST

Captured items, corresponding to legitimate queries, were made from actual DNS queries that asked for an address (A) record [*6]. Desired outputs for the captured items were manually determined, as even ordinary traffic includes a certain amount of random queries, whether or not an attack is being carried out.

Intended to simulate malicious queries, the generated items were randomly created and therefore the desired outputs for them were always random. They all met the following conditions and the distribution was uniform:

- The leftmost label has 10 to 19 letters, and
- Possible characters for labels are the lower-case letters, numerals, and the hyphen (-).

We selected the candidate letters according to the Request for Comments (RFC) for the DNS [24], and the label length based on the implementation of the Metasploit Framework [25] [*7], which included a function to conduct Kaminsky attacks. As we previously noted, emerging queries during water torture attacks are similar to those during Kaminsky attacks. We thus decided to employ the same approach to generate imitated malicious domain names in the evaluation.

We conducted cross-validation when calculating accuracy and FPRs. When evaluating a classifier, we thus tested all possible patterns of combinations of training and testing datasets and finally averaged all the results.

6.1.3 Comparison of Various Settings

The cutoff point for the length-based feature and the smoothing factor are hyperparameters, which significantly affects the accuracy of classifiers, and thus they should be deliberately determined. We therefore conducted a *grid search*: trying every possible hyperparameter and then choosing the one that achieves the best accuracy.

The following shows the search ranges, where boldfaced values mean that a corresponding feature is disabled.

Cutoff point $\mathbf{1}, 2, \ldots, 62, 63$; and

Smoothing factor $\mathbf{0}, 10^{-5}, 10^{-4}, \ldots, 10^{3}, 10^{4}$.

Experimental results indicated the best cutoff point is 12 and that the best smoothing factor is 10^{-3} (0.001). With these best hyperparameters, the classifier achieved **98.16%** accuracy and a **1.55%** FPR. **Table 6** details the classification results for this case.

Table 6 Confusion matrix of results obtained with the best hyperparameters.

		Predicted	
		Random	Non-random
Actual	Random	46,403,257 (55.75%)	976,592 (1.17%)
	Non-Random	555,801 (0.67%)	35,298,186 (42.41%)
		Accuracy	98.16%
		Error rate	1.84%
		FPR	1.55%

Figure 3 and **Fig. 4** illustrate how the hyperparameters affect the performance.

Figure 3 shows the results when the cutoff point was varied with the smoothing factor fixed at 0.001, omitting results when the value was more than 20 because a similar tendency continues until a value of 63. There appeared to be a limit to improvement of performance at higher cutoff points. This evaluation showed that the best cutoff point was 12, but cutoff points greater than 12 also produced acceptable outcomes.

Figure 4, meanwhile, shows the results when the smoothing factor was varied with the cutoff point fixed at 12. We found that the smoothing technique apparently improved accuracy, but that an excessive smoothing factor caused extreme deterioration.

6.1.4 Comparison to Our Previous Method

We applied our previous method to the datasets described in Section 6.1.2. **Table 7** shows the detection results. In order to adjust conditions, the feature of the number of occurrences of the same domain names, which is the feature that the previous method used, was disabled.

6.1.5 Durability against Detection Avoidance by Attackers

If attackers notice that some mechanism works to prevent attacks and that the mechanism is based on bigrams, they may try to deceive the system by adding common words, namely frequently used as a domain name like *www*, to queried domain names.

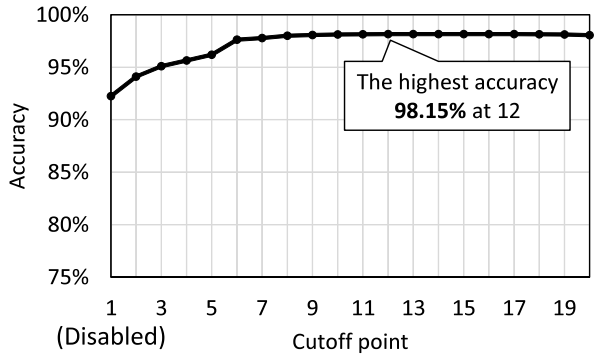
e.g., *wwwqawsedr.example.com.*

To neutralize this trick, it is effective to embed such common words into generated items in a dataset. For example, the datasets alter Table 3 to **Table 4**.

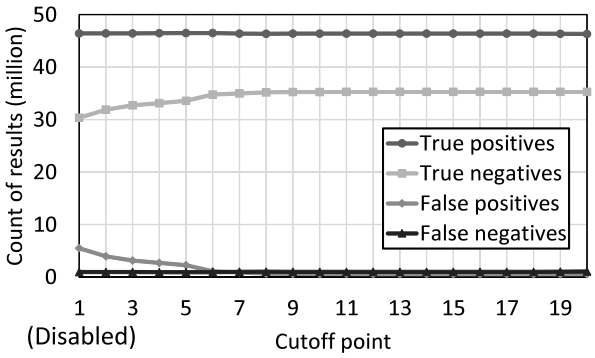
We examined how well this treatment works. Although there are many words common as a domain name, in order to simplify this evaluation, we chose *www*, presumably the most common word in domain names, as an embedded word. In this evaluation, a training dataset was either one: (A) the original version of Dataset No.1 or (B) a modified version of Dataset No.1. *www* is embedded to each generated items in the modified version. A testing dataset was Dataset No.2 that 0, 3, 6, 9, 12, or 15 letters of *w* are embedded into.

Figure 5 shows the results of this evaluation. We observed obvious deterioration in accuracy with the training dataset A, and the more the repetition of *w* increased, the more the accuracy decreased (97.94% → 63.12%). On the other hand, the worst accuracy was obtained when the number of *w* was 0 (97.48%), and the more the repetition of *w* increased, the more the accuracy also

[*6] The packets used for the captured items were captured at the edge router in Toyohashi University of Technology.
[*7] The Metasploit Framework is a well-known, open-source tool for penetration tests.

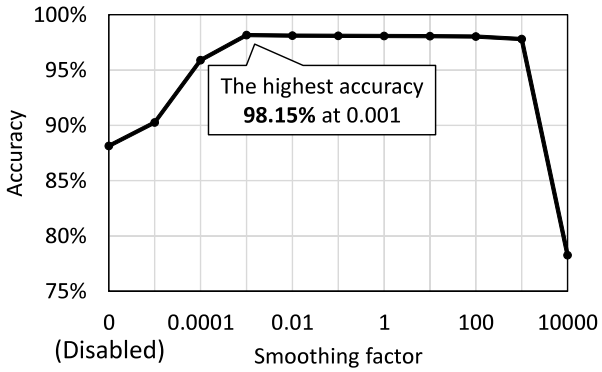


(a) Fluctuation in accuracy

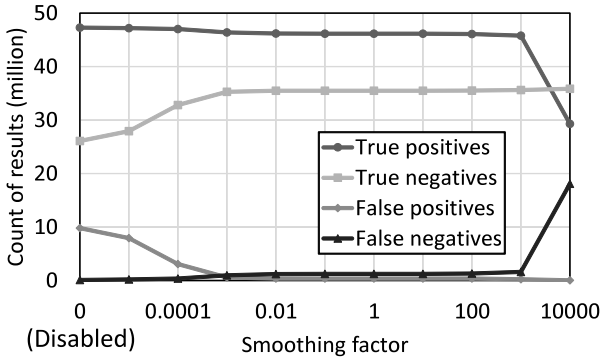


(b) Fluctuation in counts of each type of detection results

Fig. 3 Results when varying the cutoff point.



(a) Fluctuation in accuracy



(b) Fluctuation in counts of each type of detection results

Fig. 4 Results when varying the smoothing factor.

Table 7 Confusion matrix of results obtained by the previous method.

		Predicted	
		Random	Non-random
Actual	Random	45,880,665 (55.12%)	1,499,184 (1.80%)
	Non-Random	2,838,246 (3.41%)	33,015,741 (39.67%)
		Accuracy	94.79%
		Error rate	5.21%
		FPR	7.92%

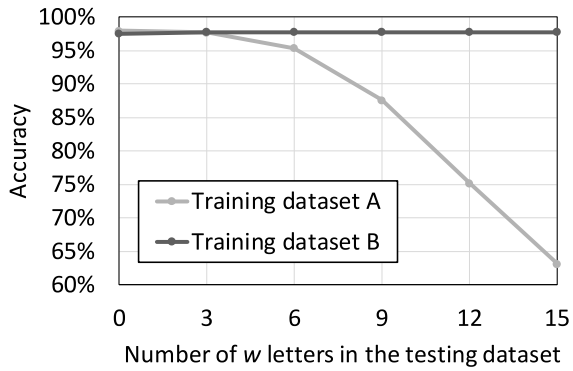


Fig. 5 Accuracy deterioration when an attacker tries to avoid detection.

increased (97.74% at 15). FPRs differed depending on the used training dataset. We achieved the lower rate (0.71%) with the dataset B, while the FPR was 1.38% when we used the dataset A. Note that the number of the repetition of w is not related to FPRs, because all of the generated items are labeled to be random and errors that treat actual random items as non-random are false negatives.

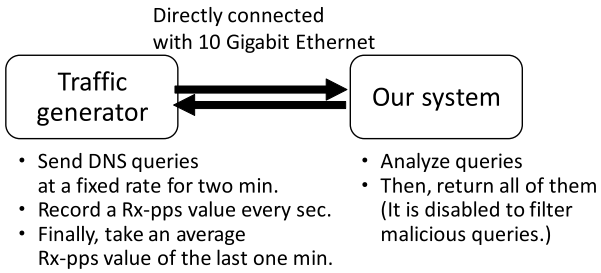


Fig. 6 Throughput evaluation.

6.2 Throughput Evaluation

This evaluation aimed to test the entire system to obtain throughput with generated traffic. In this study, we primarily focus on packet rates, instead of bit rates, as throughput of the system, because water torture attacks do not intend to saturate network bandwidth capacity.

In this section except for Section 6.3, the whitelisting was disabled. The hardware and software specifications of the system are detailed in Section 5.2.

6.2.1 Method

In the throughput evaluation, the system and a traffic generator (Xena C1-M2SFP+) were directly connected with 10 Gigabit Ethernet (**Fig. 6**). Although the system was originally designed to work as a filtering system, in the evaluation, it was set only to

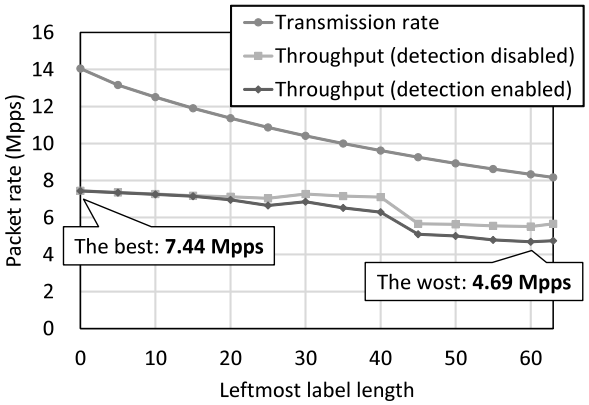


Fig. 7 Experimental results of the throughput of the system.

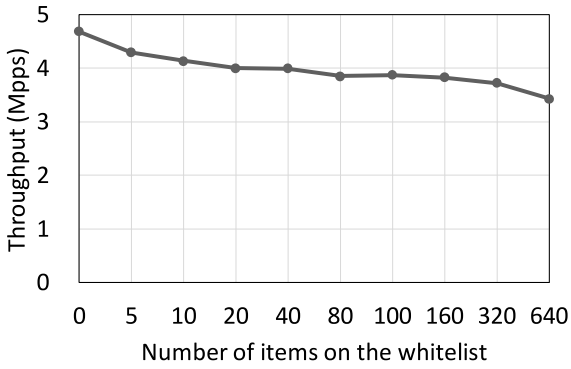


Fig. 8 Throughput deterioration caused by the whitelisting.

analyze queries and not to filter out malicious queries. That is, all analyzed queries were sent back.

The generator observed returned packet rates, which we regarded as throughput. In addition to throughput when the system acted as a detection system, we also obtained throughput when the detection function was disabled, in order to discuss processing time for detection.

Each experiment constituting this evaluation lasted for two minutes (Fig. 6), and during an experiment, the generator transmitted queries at a constant rate. The transmission bit rate was always 10 Gbps among all the experiments, but because of the difference of packet size, resulting from the length of queried domain names, transmission packet rates differed according to each experiment.

Each packet that the generator transmits was a DNS query constructed on an IPv4 UDP packet, which consisted of only one question section of an A record. Among the experiments, queries concerning domain names differed but the other parameters were the same. The queried domain names were one-label strings whose length ranged from 0 (the root domain) to 63, which condition came from the RFC 1035 [24]. Due to the limitation of the generator, all the letters of the queried domain names are *a.* (e.g., *aaaaa.*)

One minute of the first half was a break-in and then we recorded received packet rates every second for one minute of the last half. Finally, we treated an average value of the recorded packet rates as throughput in a current condition. Because we observed that received packet rates did not fluctuate significantly after the one-minute break-in, we consider that one minute is long enough to evaluate throughput.

6.2.2 Results

Figure 7 shows the effects on throughput when the detection function was enabled, when detection was disabled, and lengths of the leftmost label. The best throughput of 7.44 Mpps (million packets per second) was achieved at 0 letters (queries for the root domain), and the worst throughput of 4.69 Mpps was *obtained* at 60 letters.

6.3 Influence of the Whitelisting

The whitelisting aims to suppress legitimate queries from wrong blocking, but it brings about throughput deterioration.

This subsection shows how throughput decreased by implementing the whitelisting explained in Section 5.4. In this evaluation, a queried domain name of transmitted queries was (7 letters).(6 letters) (i.e., *aaaaaaa.aaaaaa.*) in order to obtain average throughout. The numbers of the label lengths were the averages that we obtained by analyzing our captured data.

Figure 8 shows deterioration in throughput when the whitelisting was activated. The throughput decreased according to the number of items registered on a whitelist.

Considering applying our system to large recursive servers, we should also obtain throughput in the worst condition and confirm whether the system is applicable to such servers. Therefore, we changed the experiment condition so that the throughput would be degraded the most. The number of the whitelist items was still 640, but the form of the queried domain name was changed to (63 letters).(63 letters). Then, we obtained the worst throughput of 1.34 Mpps.

7. Discussion

7.1 Detection Accuracy

7.1.1 Accuracy Obtained with the Best Hyperparameters

Table 6 shows that the classifier tends to generate more false negatives than false positives. A subject prone to generating false negatives is a combination of existing words and random strings, for example, *ldr-elb-ext-reader-264821303* and *events-endpoint-c-394794954*. Although these domain names tend to be classified as random, they may not be treated as malicious. The reason is that content delivery network (CDN) providers sometimes use such random-looking domain names. To solve this problem, we propose the whitelisting described in Section 5.4.

On the other hand, false-positive domain names are often short in length and composed of the following kinds of words:

- non-frequent, abbreviated words (e.g., *xmlrpc*, *ipv4only*), and
- non-English, or Romanized Japanese, words (e.g., *mtfuji*, *hokkaido-np*).

Most non-random domain names come from English words, and these words should have specific tendencies of frequent bigrams different from these types of domain names.

7.1.2 Correlation between Accuracy and Hyperparameters

As Fig. 3 and Fig. 4 indicate, both features generate similar results when varied. Although this variation hardly affects the numbers of true positives and false negatives, it causes significantly more true negatives and fewer false positives, which contributed

to the overall accuracy. However, the exception is that an excessive smoothing factor deteriorates accuracy.

Figure 3 shows the accuracy of the length-based feature. It is clear that enabling the length-based feature yields better performance than when it is disabled, showing that adopting this feature is worthwhile. The feature appears to act as a bias that mitigates false positives and false negatives to some extent, as described in Section 7.1.1. This improvement in accuracy demonstrates that the combination of the label-based feature and bigram-based features works well.

However, considering that a too high cutoff point does not yield a proportional improvement in performance and also requires more memory resources, the cutoff point should be as small as is sufficient for accuracy.

According to Fig. 4, an excessive smoothing factor significantly deteriorates detection performance. The reason for this behavior is that in calculations of parameter estimates $\theta_{ki} = (N_{yi} + \alpha)/(N_y + \alpha n)$, the actual feature values N_{yi} and N_y would be respectively overcome by the values of α and αn selected for smoothing. The best smoothing factor value therefore would be determined according to the number of items listed in a training set, and this value should be experimentally sought.

7.1.3 Comparison to Our Previous Method

As Table 6 and Table 7 show, although several features were removed in the current method, the current method outperforms the previous method in both accuracy and FPR. In particular, as an automatic filtering system, denying legitimate users should be avoided as much as possible, and therefore this noticeable improvement in the number of false positives is meaningful.

This desirable outcome arises from the change in the way that bigram-based features are calculated. Whereas the previous method summed up all bigram-based features for a fast and uncomplicated way to avoid the zero-frequency problem, the current method adopts a more accurate measure.

7.1.4 Durability against Detection Avoidance by Attackers

It is obviously effective to embed common words, which attackers may try to use to avoid the filtering system, into a training dataset. In the evaluation, although we embedded only one common word *www*, it would be also effective to use more than one word such as *mail* and *ns*. It is difficult for attackers to know which words are measured, namely embedded, from outside, and thus this treatment brings about more complexity to achieving attacks than the system without the treatment.

However, we would need further discussion about the effects of the treatment on the length-based feature. This treatment makes short-length domain names tend to judge to be non-random due to the length-based feature. This effect is the cause of the improvement of the FPR of the training dataset B, but we think that this is just a fortunate case.

Besides this treatment, it would be possible to simply trim off common words in queried domain names before naïve Bayes classification.

7.2 System Throughput

The throughput effectively decreases with the label length. An unnatural drop is seen at 45, which would be a characteristic of the network card card installed in this system.

The major cause of throughput degradation with the label length, however, is the computation of probability estimates regarding bigram-based features, which requires that the whole of the leftmost label is sought. On the other hand, the calculation of probabilities regarding the length-based feature is fast because a DNS packet itself contains the length of each label.

Even when detection was disabled, the throughput was greatly decreased from transmission rates, which suggests there was a bottleneck at a point unrelated to the detection function. The system is currently single-threaded but appears to need more threads to process a large number of queries. We consider that our method fortunately has a high affinity for multi-threading implementation. After the learning stage, the frequency tables, which are used in probability calculation, are only referred to and no longer rewritten, and thus the procedure of query analysis does not require exclusive control of table access.

Although experimental results hint that we require further improvement in execution performance, we believe that our system, even as it stands now, has sufficient capacity against massive water torture attacks. A report from Nominum, Inc. states that a large server held by a North American ISP underwent approximately 0.8 Mpps of queries when an attack occurred [26]. The capacity of our system far exceeds the scale of this attack even in its worst condition. The throughput that our system achieved should therefore be fast enough to withstand massive water torture attacks on large-scale recursive servers.

7.2.1 Whitelisting

Enabling this function surely deteriorates the throughput, but even in the worst condition, the obtained throughput exceeded 0.8 Mpps, the observed record of the intensity of an attack [26]. We therefore consider that the whitelisting version of the system is also able to counteract large-scale water torture attacks.

8. Conclusion

In this paper, we presented a detection method and a filtering system for water torture attacks, a type of DDoS attack on DNS servers that features random subdomains. Considering large-scale applications, we developed the method and associated system with a deep consideration for high throughput and adopted a naïve Bayes classifier, a fast but high-accuracy supervised learning algorithm.

Experimental results showed that our method can detect malicious queries with 98.16% accuracy and a 1.55% false positive rate, and that our filtering system achieved 7.44 Mpps of throughput in the best case and 4.69 Mpps even in the worst condition if rough whitelisting was disabled. This capacity should be sufficient for most recursive servers.

This study established the fundamentals of detection of water torture attacks based on naïve Bayes classification. For future work, in order to attain more reliable detection, special handling would be needed for random-looking but non-malicious domain names such as used in CDNs and IDNs.

References

[1] Secure64 Software Corporation: Water Torture: A Slow Drip DNS DDoS Attack, Secure64 Software Corporation (online), available from ⟨https://blog.secure64.com/?p=377⟩ (accessed 2015-11-30).

[2] Takeuchi, Y., Yoshida, T., Kobayashi, R., Kato, M. and Kishimoto, H.: Detection of the DNS Water Torture Attack by Analyzing Features of the Subdomain Name, *Journal of Information Processing*, Vol.24, No.5, pp.793–801 (online), DOI: 10.2197/ipsjjip.24.793 (2016).

[3] Yoshida, T., Takeuchi, Y., Kobayashi, R., Kato, M. and Kishimoto, H.: Study on Filtering Method for the DNS Water Torture Attack Utilizing the Naive Bayes Classifier (in Japanese), *IPSJ SIG Notes*, 2016-CSEC-74, No.26, pp.1–7 (2016).

[4] Yuchi, X., Wang, X., Lee, X. and Yan, B.: *A New Statistical Approach to DNS Traffic Anomaly Detection*, pp.302–313 (online), DOI: 10.1007/978-3-642-17313-4_30, Springer Berlin Heidelberg (2010).

[5] US-CERT: DNS Amplification Attacks — US-CERT, United States Computer Emergency Readiness Team (online), available from ⟨https://www.us-cert.gov/ncas/alerts/TA13-088A⟩ (accessed 2016-10-18).

[6] Almond, C.: Recursive Client Rate limiting in BIND 9.9.8 and 9.10.3, Internet Systems Consortium (online), available from ⟨https://kb.isc.org/article/AA-01304/0/Recursive-Client-Rate-limiting-in-BIND-9.9.8-and-9.10.3.html⟩ (accessed 2016-10-11).

[7] Alonso, R., Monroy, R. and Trejo, L.: Mining IP to Domain Name Interactions to Detect DNS Flood Attacks on Recursive DNS Servers, *Sensors*, Vol.16, No.8, p.1311 (online), DOI: 10.3390/s16081311 (2016).

[8] Cloudmark, Inc.: Cloudmark Security Platform for DNS Solution Guide, Cloudmark, Inc. (online), available from ⟨https://www.cloudmark.com/releases/docs/solutionguides/dns-solution-guide-2014-october.pdf⟩ (accessed 2016-10-08).

[9] Freeman, D.M.: Using naive bayes to detect spammy names in social networks, *Proc. 2013 ACM Workshop on Artificial Intelligence and Security - AISec '13*, pp.3–12 (online), DOI: 10.1145/2517312.2517314 (2013).

[10] Matsubara, Y., Musashi, Y., Sugitani, K. and Moriyama, T.: Open DNS Resolver Activity in Campus Network System, *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, pp.145–148 (online), DOI: 10.1109/icinis.2015.30 (2015).

[11] Yadav, S., Reddy, A.K.K., Reddy, A.N. and Ranjan, S.: Detecting algorithmically generated malicious domain names, *Proc. 10th annual conference on Internet measurement - IMC '10*, pp.48–61 (online), DOI: 10.1145/1879141.1879148 (2010).

[12] Schiavoni, S., Maggi, F., Cavallaro, L. and Zanero, S.: *Phoenix: DGA-Based Botnet Tracking and Intelligence*, pp.192–211 (online), DOI: 10.1007/978-3-319-08509-8_11, Springer Science + Business Media (2014).

[13] Domingos, P. and Pazzani, M.: On the Optimality of the Simple Bayesian Classifier under Zero-One Loss, *Machine Learning*, Vol.29, No.2/3, pp.103–130 (online), DOI: 10.1023/a:1007413511361 (1997).

[14] Rish, I.: An empirical study of the naive Bayes classifier, *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, Vol.3, No.22, pp.41–46, IBM New York (2001).

[15] McCallum, A. and Nigam, K.: A Comparison of Event Models for Naive Bayes Text Classification, *Learning for Text Categorization: Papers from the 1998 AAAI Workshop*, pp.41–48 (1998).

[16] Sahami, M., Dumais, S., Heckerman, D. and Horvitz, E.: A Bayesian Approach to Filtering Junk E-Mail, *Learning for Text Categorization: Papers from the 1998 Workshop* (1998).

[17] Wang, S. and Manning, C.D.: Baselines and Bigrams: Simple, Good Sentiment and Topic Classification, *Proc. 50th Annual Meeting of the Association for Computational Linguistics: Short Papers - Volume 2, ACL '12*, pp.90–94 (2012).

[18] Choi, S.-W. and Lee, C.H.: A FPGA-based parallel semi-naive Bayes classifier implementation, *IEICE Electronics Express*, Vol.10, No.19, pp.20130673–20130673 (online), DOI: 10.1587/elex.10.20130673 (2013).

[19] Andrade, G., Viegas, F., Ramos, G.S., Almeida, J., Rocha, L., Goncalves, M. and Ferreira, R.: GPU-NB: A Fast CUDA-Based Implementation of Naïve Bayes, *2013 25th International Symposium on Computer Architecture and High Performance Computing*, pp.168–175 (online), DOI: 10.1109/sbac-pad.2013.16 (2013).

[20] Schneider, K.-M.: A comparison of event models for Naive Bayes anti-spam e-mail filtering, *Proc. 10th Conference on European Chapter of the Association for Computational Linguistics - EACL '03*, pp.307–314 (online), DOI: 10.3115/1067807.1067848 (2003).

[21] Metsis, V., Androutsopoulos, I. and Paliouras, G.: Spam Filtering with Naive Bayes - Which Naive Bayes?, *CEAS*, pp.27–28 (2006).

[22] Fiberblaze: fb2XGhh@V7 series, Fiberblaze A/S (online), available from ⟨http://www.fiberblaze.com/product-details/fb2xg-dual-sfp-port-card-supporting-2x1ge10ge-half-height-pcie-gen-3-x8-lanes/⟩ (accessed 2015-11-30).

[23] Costello, A.: Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA), The Internet Engineering Task Force (online), available from ⟨http://www.ietf.org/rfc/rfc3492.txt⟩ (accessed 2017-03-31).

[24] Mockapetris, P.: DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION, The Internet Engineering Task Force (online), available from ⟨http://www.ietf.org/rfc/rfc1035.txt⟩ (accessed 2015-11-30).

[25] Rapid7, Inc.: GitHub - rapid7/metasploit-framework: Metasploit Framework, Rapid7, Inc. (online), available from ⟨https://github.com/rapid7/metasploit-framework⟩ (accessed 2016-09-27).

[26] Waber, R.: Random Subdomain Attacks Plaguing the Internet, Nominum, Inc. (online), available from ⟨https://indico.uknof.org.uk/materialDisplay.py?contribId=15&materialId=slides&confId=31⟩ (accessed 2016-11-24).

Takuro Yoshida received his B.E. degree in Computer Science and Engineering from Toyohashi University of Technology in 2015 and is currently a master's student at the same university. His research interests include network security.

Kento Kawakami is currently a bachelor's student at Toyohashi University of Technology. His research interests include network security.

Ryotaro Kobayashi received his B.E., M.E., and D.E. degrees from Nagoya University in 1995, 1997, and 2001, respectively. He had been a research assistant at Nagoya University from 2000 to 2008, a lecturer at Toyohashi University from 2008 to 2015, and an associate professor at Toyohashi University in 2016. He is currently an associate professor at Kogakuin University. His research interests include computer architecture, parallel processing, and network security.

Masahiko Kato received his B.E. and M.E. degrees in Engineering from Toyohashi University of Technology and D.E. degree in Systems and Information Engineering from University of Tsukuba respectively. He is now a professor at University of Nagasaki. He is currently interested in network security.

Masayuki Okada works in the Engineering Department at JPNIC. He experienced a BGP operation of an academic network since 2000. Mr. Okada joined JPNIC in 2004 and is responsible for the development and operation of the IP resource management system related to routing and JPIRR research, as well as the use of IRR. In recent years, he has focused his efforts on outreach about RPKI technology and its operational deployment. He finished his Ph.D. in 2016 in Computer Science.

Hiroyuki Kishimoto received his B.E. and M.E. degrees from Hosei University in 1988, 1990, respectively. He is now working for ComWorth Co., Ltd. since 1991. He is currently interested in high speed network DPI and lossless packet capturing method under 40G/100G environment.