Electronic Preprint for Journal of Information Processing Vol.24 No.5

Regular Paper

# Detection of the DNS Water Torture Attack by Analyzing Features of the Subdomain Name

Yuya Takeuchi[1,a) ] Takuro Yoshida[1] Ryotaro Kobayashi[1] Masahiko Kato[2]
Hiroyuki Kishimoto[3]

Received: December 3, 2015, Accepted: June 2, 2016

**Abstract:** The Domain Name System (DNS), whose major function is to manage associations between domain names and IP addresses, plays a major role in managing the Internet. Thus, a DNS impairment would significantly impact society. A major cause of DNS impairment is Distributed Denial of Service (DDoS) attack on authoritative DNS servers. Our study focuses on the recently emerging DDoS attack known as the DNS Water Torture Attack. This attack causes open resolvers, which are improperly configured cache DNS servers that accept requests from both LAN and WAN, to send many queries to resolve domains managed by target servers. Domain names for resolving sent in this attack include varying random subdomains. Cache servers certainly will not have cached data for these queries, and so a huge volume of queries converges to the target authoritative servers via cache servers. In this paper, we propose a detection method for this attack using the Naive Bayes Classifier. Experimental results show that our method is capable of detecting this attack with a 95.59% detection rate. Moreover, the results of performance simulation show that our method is fast enough to process more than 2.3 Gbps of traffic on the fly.

**Keywords:** DNS, DDoS, DNS Water Torture, Anomaly Detection, Naive Bayes Classifier

## 1. Introduction

A vital system underlying the Internet is the Domain Name System (DNS), whose major function is to manage associations between domain names and IP addresses. Considering the strong influence of computers and the Internet on our lives, a DNS impairment or failure would significantly impact society. In particular, Denial of Service (DoS) and Distributed DoS (DDoS) attacks on the DNS are recently emerging threats.

When a DNS client tries to resolve a domain name, it sends a recursive query to a cache server. The cache server instantly returns a response if it has a previously cached result. Otherwise, it sends iterative queries on behalf of the client to authoritative servers that manage the original information associating domain names with IP addresses, thus obtaining the corresponding IP address. A domain name is generally presented as a dot-separated string, where each substring is a label and the position of a label indicates its level in a hierarchy.

The behavior of a cache server that has received a query is as follows. First, the cache server sends a query to a root node, which is an authoritative server maintaining root domains. Then, the root node replies with the address of a next-level authoritative server. This is instead of the IP address of the queried domain because root nodes do not have complete information of every domain in the Internet. If it has data for the queried domain, the next-level server replies an IP address corresponding to the domain. Otherwise, the server introduces a 2nd next-level server as in the previous step. This routine continues until the cache server meets a server that knows the sought domain.

The Water Torture Attack [1] is a type of DDoS attack on the DNS that exploits open resolvers to send a huge volume of queries to target authoritative servers. The open resolvers are vulnerable cache servers that accept all queries from all over the Internet.

In this paper, we propose a detection method for the Water Torture Attack that uses the Naive Bayes Classifier. We perform experiments that show the method is fast enough to be utilized for semi-real-time attack detection. Here, *"semi"* refers to the fact that there is a delay between when an attack query arrives and when it is detected since our method needs a certain volume of packets, called a window, for detection.

The remainder of this paper is organized as follows. Section 2 discusses related studies and details the Water Torture Attack. Sections 3 and 4 describe our proposed method, and its implementation, respectively. Sections 5 and 6 clarify evaluation environment and results of our study, respectively. Section 7 details the effectiveness of our method from the points of view of both the accuracy and the performance. Finally, Section 8 concludes our study and discusses future work.

## 2. Related Work

### 2.1 DNS-Related Malicious Activity Detection

Okayasu and Sasaki [2] proposed a method that discovers communication from botnets by examining parameters of a Command-and-Control (C&C) server's domain, such as whether

[1] Faculty of Engineering, Toyohashi University of Technology, Toyohashi, Aichi 441–8580, Japan
[2] Internet Initiative Japan Inc., Iidabashi Grand Bloom, Chiyoda, Tokyo 102–0071, Japan
[3] ComWorth Co., Ltd., Ota, Tokyo 143–0026, Japan
[a)] takeuchi2015@ppl.cs.tut.ac.jp

© 2016 Information Processing Society of Japan

it can be resolved by reverse lookup, a Time to Live (TTL) value, and address records (A records). This particular method focuses on suspicious domain information, but not all attacks have such characteristics. For example, in the Water Torture Attack, the domain information appears normal, but the queries cast by attackers are the source of the problem. Thus, our study focuses on the characteristics of such queries.

Karasaridis et al. [3] introduced a method to detect DNS Cache Poisoning and DNS Tunneling. Unlike ordinary Intrusion Detection Systems, their method does not rely on known attack patterns called signatures. Rather, it utilizes flow data whose records contain parameters related to the transport and lower layers. These parameters include IP addresses and port numbers of each source and destination, and the numbers of packets and bytes. However, these parameters are insufficient to recognize attack queries and thus cannot be effectively applied to the Water Torture Attack in isolation. Our method, therefore, inspects the contents of queries at the DNS level.

The main point differentiating our study from those above is our focus on the Water Torture Attack. We propose a high-speed detection method for the Water Torture Attack that can observe a large number of queries on the fly.

### 2.2 DNS Water Torture Attack

The DNS Water Torture Attack, also known as the DNS Slow Drip Attack or the Pseudo Random Subdomain Attack, is a type of DDoS attack on DNS servers [4]. This attack affects both authoritative and cache servers, but mainly targets the former [1].

The attacker uses many bots to make the target servers repeatedly resolve characteristic domain names. These domain names consist of the parent domain for which the target has authority and a random, non-existent subdomain. Due to the non-existent subdomain, cache servers receiving such queries never have cached data and thus must send iterative queries to the authoritative servers.

This attack exploits improperly-configured cache servers called open resolvers. While normal servers ignore queries from external networks, open resolvers accept all queries from every possible client. Some of these open resolvers are home routers serving as DNS forwarders to upstream cache servers provided by internet service providers (ISPs). Through them, a large number of queries ultimately converge on the target authoritative servers.

Moreover, during the attack, the slow response of the authoritative servers due to overload causes a drain on the resources of the cache servers, potentially realizing the secondary damage of knocking out the cache servers. In fact, it has been reported that ISP cache servers, which are referred by many home routers, have crashed as a side effect of such attacks [5].

Before carrying out this attack, the attacker prepares the following two lists:

( 1 ) a list of bots participating in the attack, and
( 2 ) a list of open resolvers.

**Figure 1** shows the flow of attack queries. When the attacker commands the bots in List 1 to start the attack, every bot continuously sends multiple queries to the open resolvers listed in List 2. Due to the non-existent subdomain of the query, the open

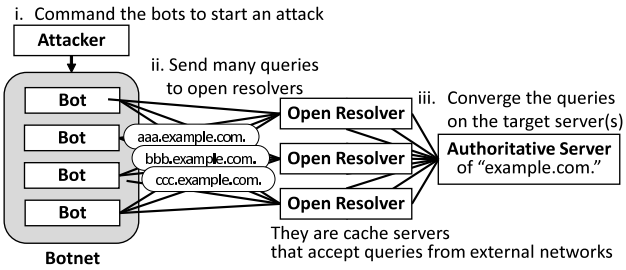


**Fig. 1** Flow of attack queries.

resolvers simultaneously query the authoritative servers, sharply increasing the load of the target servers. Further, the load on the cache servers, especially upstream ISP servers referred by bad home routers, also increases since recursive queries require more resources than iterative ones.

Major countermeasures against this attack include IP address blocking and rate limiting of accepting queries; however, these actions are not complete solutions because they could significantly affect innocent users. As described above, attack queries often go through ISP cache servers and so the queries appear to the attacked server to be from these ISP cache servers. This means that simply denying a specific IP address results in refusing clients from the network of the ISP. With regard to rate limiting, if the intensity of the attack exceeds the limit, all queries are blocked even if they are from innocent clients. To effectively resist this attack, it is vital to detect and selectively block attack queries.

To detect attack queries, we focus on the randomness of their subdomains. There have been proposed a number of studies related to the randomness of domains.

Kazato et al. [6] focused on responses signifying NXDOMAIN errors, which occur when a queried domain name does not exist. The authors classified them into nine patterns, with one pattern being whether they include random words. They predicted whether a domain name included random words via a score calculated by comparing bigrams of domain names of subjects and those of correct domain names. While this study treated random queries merely as one of the causes of NXDOMAIN errors, we noticed that the randomness estimation could be utilized as Water Torture detection, and thus we devised a detection method and evaluate its detection rate and performance.

Yadav et al. [7] proposed a botnet detection method, which focus on domain names *not* made by humans. C&C servers' domains have the feature that they are frequently changed to new ones generated by an algorithm. Such algorithms are referred to as Domain Generation Algorithms (DGAs). This study aimed to detect botnets by distinguishing generated domains using Kullback-Leibler (K-L) divergence of distributions of letters. Instead of botnets, our study focuses on Water Torture detection and on-the-fly processing.

Schiavoni et al. [8] also proposed a detection method for DGA-generated domains. The study utilizes four feature values: the uni-, bi-, and tri-gram normality scores and the ratio of meaningful characters. The authors stated that they employed multiple features to reduce misjudgement. However, the recall rate of the method fluctuated according to data sets. We think that it is be-

© 2016 Information Processing Society of Japan

cause all the feature values have properties similar to each other. On the other hand, we utilize feature values that have different properties in our method, and therefore, we obtained stable experiment results regardless of data sets.

## 3. Proposed Method

### 3.1 Overview

The aim of our method is to detect random queries. When DDoS attack occurs, server administrators have few measures to withstand the attack, and so, it is important for them to cooperate with managers of upstream networks in general. With our system, administrators of authoritative servers can notice that some servers are under attack, and they can also know IP addresses of open resolvers that relay attack queries and which queries are for the attack. If we provide such detailed information about the attack for a manager of the upstream network, the upstream manager can investigate relaying machines far from the attacked server and take more actions to mitigate the attack, such as filtering or bandwidth control based on IP addresses or other information. We presume that our system can be utilized in this way as a countermeasure for the Water Torture Attack.

Our method monitors the traffic from and into the authoritative servers. **Figure 2** shows the structure of an assumed system for implementing our method and the flow of packets.

The router is configured to send a copy of all the traffic to the detection machine. The machine extracts packets related to the DNS from the copy, analyzes these packets, and determines whether an attack is being carried out.

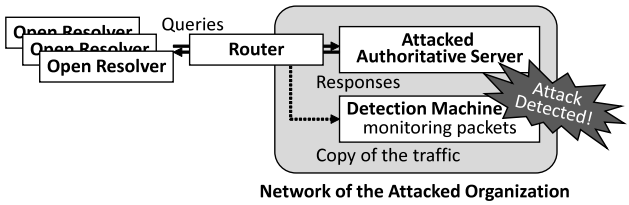


**Fig. 2** Structure of an assumed system.

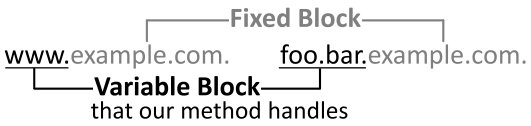


**Fig. 3** Two parts of an FQDN.

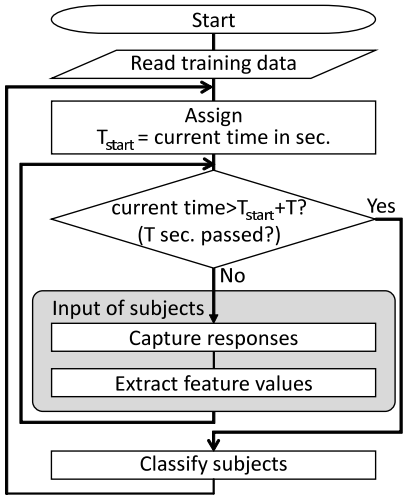


**Fig. 4** Flowchart of the detection algorithm (on-the-fly detection mode).

### 3.2 Analysis Subjects

Recalling that the attack queries use non-existent random domain names, which would result in NXDOMAIN errors, our method focuses on only queried domain names that receive an NXDOMAIN response.

In this study, we divide a fully-qualified domain name (FQDN) of an attack query into two parts: the variable block, which is the changing part of the domain used for different queries, and the fixed block, which is the fixed part of the domain for which the attacked servers have authority (**Fig. 3**). Our method attributes an intention to attack to queried domain names if their variable blocks are random and non-existent. In contrast, all fixed blocks are ignored.

We analyze the pair containing a variable block and its number of occurrences in a single window. Note that a window is a cluster of subjects made from packets captured over T seconds, where the parameter T is called the *window width*.

### 3.3 Detection Algorithm

**Figure 4** shows the behavior of the detection algorithm, assuming that analysis subjects are taken from packets captured in real time.

To determine whether a variable block is random, our method utilizes the Naive Bayes Classifier, which is a supervised-learning algorithm and is detailed in Section 3.5. Note that we choose the Naive Bayes Classifier over the Support Vector Machine or Decision Tree as it runs in linear time by assuming independence between feature values [9]. As we need to process a large number of packets on the fly, speediness is a vital consideration.

Employing the classifier requires first reading training data. Each point of training data is a pair of an input and an output; the input consists of a variable block string and its number of occurrences, and the output states whether the input is random. After this training, we obtain the reference frequency tables for both non-random and random cases. These tables will be referred to by the classifier to classify analysis subjects.

The details of the detection process are as follows. The detection algorithm receives all subjects within a window. The subjects come from either packets captured in real time or an already-prepared file of captured packets. After receiving a window, our method extracts feature values from each subject in the window, and then classifies the subject as random or non-random from these feature values, using the Naive Bayes Classifier. These steps of receiving and classifying run alternately and continue until a user stops the program or the input file ends.

### 3.4 Feature Values

Our method uses these values as feature values of the subjects:

( 1 ) The number of occurrences of the same variable block in a window;

( 2 ) The whole length of a variable block excluding dot delimiters;

( 3 ) The number of labels in a variable block; and,

© 2016 Information Processing Society of Japan

(4) The bigrams made from a variable block.

The reason to take the number of occurrences is that the same domain names should be queried few times since these queries are random.

The reason to take the length of a variable block is that the length of attack queries should be long enough to prevent collision of queried domain names. Accordingly, the number of labels would also increase, which is the reason to take the number of labels.

Bigrams are sequences of a string split by every two adjacent letters. For example, the variable block string "foo.bar." is split into the bigrams "fo", "oo", "o.", ".b", "ba", "ar", and "r." Note that dots terminating variable blocks are not ignored when bigram scores are calculated.

We use bigrams instead of unigrams since it has been reported that we can achieve more precise randomness estimation using bigrams than with unigrams [7], [10]. We expect to achieve a high and stable detection rate by utilizing the bigram score together with feature values unrelated to bigrams.

### 3.5 Naive Bayes Classifier

Here, we assume that feature values are in a four-dimensional vector, and the two classes are random or non-random.

Given feature values $\boldsymbol{X} = (X_1, X_2, X_3, X_4)$, which correspond to the values described in the section above, the classifier calculates the conditional probability $P(Y|\boldsymbol{X})$ in the case $Y = \{y_0, y_1\}$. It then classifies $\boldsymbol{X}$ into the more probable case.

The following steps show the detailed calculation.

(1) Obtain these values by referring to the appropriate reference table:

$$P(Y), P(X_1|Y), P(X_2|Y), P(X_3|Y), P(X_4|Y).$$

$P(X_4|Y)$, a score related to bigrams, is the sum of the probabilities of each bigram.

(2) Assuming that each $\boldsymbol{X}$ is independent from each other, we obtain:

$$P(\boldsymbol{X}|Y) = P(X_1|Y)P(X_2|Y)P(X_3|Y)P(X_4|Y).$$

(3) From Bayes' theorem:

$$P(Y|\boldsymbol{X}) = \frac{P(Y)P(\boldsymbol{X}|Y)}{P(\boldsymbol{X})} \propto P(Y)P(\boldsymbol{X}|Y).$$

(4) Calculate $P(\neg Y|\boldsymbol{X})$ in the same way.

(5) Comparing $P(Y|\boldsymbol{X})$ and $P(\neg Y|\boldsymbol{X})$, attribute $\boldsymbol{X}$ to the more probable case.

## 4. Implementation

### 4.1 Overview

We developed a detection program based on our method in order to evaluate its effectiveness. The program consists of three steps: pre-training, input of subjects, and classification. The pre-training step is a one-time process that runs at the start of the program, while the other steps alternately during detection. The details of each step are described in the following subsections.

### 4.2 Pre-Training

The detection program must read training data beforehand.

**Table 1** Example of Reference Table 0 (after reading a single line).

| Frequency | | Num. of Labels | | Length | | Bigram | |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 1 | 1 | 0 | "fo" | 1 |
| 2– | 1 | 2–3 | 0 | 2 | 0 | "oo" | 1 |
| | | 4–5 | 0 | 3 | 1 | "o." | 1 |
| | | 6–7 | 0 | … | 0 | … | |
| | | 8– | 0 | 10– | 0 | Total | 3 |
| Total of non-random subjects: 1 | | | | | | | |

**Table 2** Example of Reference Table 0 [*1] (after the pre-training is completed).

| Frequency | | Num. of Labels | | Length | | Bigram | |
|---|---|---|---|---|---|---|---|
| 1 | 3,000 (60%) | 1 | 1,750 (35%) | 1 | 50 (1%) | "fo" | 180 (0.18%) |
| 2– | 2,000 (40%) | 2–3 | 1,600 (32%) | 2 | 150 (3%) | "oo" | 195 (0.20%) |
| | | 4–5 | 1,000 (20%) | 3 | 250 (5%) | "o." | 890 (0.89%) |
| | | 6–7 | 600 (12%) | … | | … | |
| | | 8– | 50 (1%) | 10– | 1,600 (32%) | Total | 100,000 (100%) |
| Total of non-random subjects: 5,000 (50%) | | | | | | | |

The training data file is in CSV format and each line consists of an input and an output: the input is a pair of a variable block and its number of occurrences in a single window, and the output is whether the input is random. After training, we obtain two reference frequency tables for the non-random and random cases, which we refer to as Reference Tables 0 and 1, respectively.

First, the program reads each line of the training data file and increases the counters of relevant items in the appropriate table. For example, when the program reads a line that signifies inputs (freq.=2, num. of labels=1, length=3, variable block="foo."), and an output "non-random", the program does the following:

(1) The output being non-random, the program updates Reference Table 0;

(2) In this Reference Table, the counter of total items is increased by 1;

(3) The counter of items whose frequency is more than or equal to 2 is increased by 1 in the Frequency column;

(4) Similarly, the columns regarding the number of labels and length are updated;

(5) The counter of each appropriate item in the Bigram column is increased by 1 (note that the string is split to the bigrams "fo", "oo", and "o."); and

(6) Add 3, the number of the bigrams, to the total counter of bigrams in the Bigram column.

Thus, the Reference Table 0 becomes **Table 1** by this process.

This counting process continues until the end of the file. Then, the program calculates the probabilities of each item. For example, if 5,000 non-random items and the same number of random items are added to the tables, Reference Table 0 is altered to the form like shown in **Table 2**.

#### 4.2.1 Parameters

We consider the intervals used in the Reference Tables and the window width as significant factors that affect a detection rate, and so we treat them as parameters. **Table 3** shows the best parameters that we obtained by the parameter investigation experi-

[*1] The underlined values are referred in Section 4.4.

© 2016 Information Processing Society of Japan

ments described in Section 5.3.

### 4.3 Input of Subjects

In this step, the program receives subjects for analysis. The source of subjects differs according to the situation. In practical use and performance evaluation experiments, subjects are extracted from actual packets captured in real time. Only packets that meet the following parameters are analyzed, and all others are ignored:

- The sender is located in the same network,
- UDP, port 53,
- DNS response (qr = 1),
- NXDOMAIN error (rcode = 3), and
- Authoritative answer (aa = 1).

After inspecting a response, the program obtains an FQDN from the question section, a fixed block string from the authority section, and a variable block string by excluding the fixed block from the FQDN. As previously described, we only consider the variable block.

In evaluation experiments, the program can receive subjects directly from a file rather than by capturing packets. The format of such an input file is the same as that of the training data. In this file-reading mode, the program uses output data from the file, which was originally to be used in pre-training, to verify detection results and calculate a detection rate.

After receiving subjects, the program extracts feature values from the variable block of each subject, and records them on the Subject List.

### 4.4 Classification

After the Input step, the program refers to the appropriate reference table and uses the Naive Bayes Classifier to calculate the probability of randomness and non-randomness for each item in the Subject List, and then classifies each item.

For example, we calculate the non-randomness probability of the subject (freq.=2, num. of labels=1, length=3, variable block="foo.") from the example Reference Table 0 shown in Table 2. Note that $P(X_4|y_0)$ is the sum of the probabilities of the bigrams "fo", "oo", and "o." in this case.

The calculation is as follows:

$$\begin{aligned} P(Y|\boldsymbol{X}) &\propto P(y_0)P(X_1|y_0)P(X_2|y_0)P(X_3|y_0)P(X_4|y_0) \\ &= 50\% \cdot 40\% \cdot 35\% \cdot 5\% \cdot (0.18\% + 0.2\% + 0.89\%) \\ &= 0.004445\% \end{aligned}$$

The randomness probability can be similarly calculated from Reference Table 1. If it was less than the non-randomness probability, the subject would be considered to be non-random.

Note that the sum of the randomness and the non-randomness probabilities are not 100%, since this calculation is an approximation. We, therefore, must perform calculations for both Reference Tables.

**Table 3** Best parameters.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | Intervals of | | | | | | |
| 1 | Number of Occurrences | 1 | 2– | | | | |
| 2 | String Length | 1 | 2 | 3 | ... | 9 | 10– |
| 3 | Number of Labels | 1 | 2–3 | 4–5 | 6–7 | 8– | |
| 4 | Window Width | 10 minutes | | | | | |

## 5. Evaluation

We conducted experiments to examine the detection rate and the performance of our method, using the program detailed in Section 4.

### 5.1 Environment

#### 5.1.1 Detection Rate Evaluation

In this evaluation, the program received a file of analysis subjects, which included correct output data that was set to either random or non-random by humans beforehand. After classifying subjects, the program compared detection results with true outputs and calculated a detection rate and false positive and negative rates. Note that in this evaluation, true positives mean that subjects are random and do not directly mean that they are related to the Water Torture Attack.

#### 5.1.2 Performance Evaluation

To verify that our method can be applied in practice, we constructed an experimental environment that emulated an actual server system. The assumed situation was that the detection machine was trying to detect attack packets in the traffic from and into the authoritative servers located in the same network.

In this evaluation, a detection machine and a traffic generator were directly connected. The latter generated traffic made of attack queries and responses to them. The number of queries was the same to that of responses, but the bitrate of queries and that of responses differed because of the difference of their packet sizes.

**Table 4** shows the specifications of the detection machine, onto which a program that implemented our method installed.

The traffic generator was able to transmit packets at up to 10 Gbps. It could rewrite the contents of sent packets on the file, but not change the packet size.

### 5.2 Data Sets

#### 5.2.1 Training Data

Both the detection rate and the performance evaluation used the same training data (TD). A file of the training data was prepared from the combination of:

- The captured data, which was extracted from packets captured at our university [*2]'s network, where there are many authoritative servers; and
- Randomly generated data.

The outputs of the captured data were manually examined beforehand, and that of the generated data were all considered random.

The captured data of the training data was obtained from

**Table 4** Hardware and software specifications of the detection machine.

| | |
|---|---|
| OS | 64-bit Ubuntu 14.04.1 |
| CPU | 2 x Intel Xeon CPU E5520 (2.26 GHz, 4 cores) |
| Memory | 16.0 GB |
| Capture Card | Fiberblaze fbC2XGhh [11] (dedicated NIC for capturing up to 10 Gbps) |

[*2] Toyohashi University of Technology.

© 2016 Information Processing Society of Japan

**Table 5** Contents of the subjects in the data sets.

| Data Set | TD | AS-1 | AS-2 | AS-3 |
|---|---|---|---|---|
| Total of Responses | 19,836 | 19,023 | 14,482 | 9,245 |
| Total of Subjects | 9,423 | 9,454 | 7,693 | 6,271 |
| Captured Subjects (%) | 50.12 | 50.29 | 50.60 | 50.57 |
| Generated Subjects (%) | 49.88 | 49.71 | 49.40 | 49.43 |
| Non-Random Subjects (%) | 44.79 | 41.55 | 44.22 | 47.14 |
| Random Subjects (%) | 55.21 | 58.45 | 55.78 | 52.86 |

**Table 6** Forms of the queried domain names.

| Form | Num. of Labels | Whole Length [*5] | Packet Size (bytes) | |
|---|---|---|---|---|
| | | | Query | Response |
| 1 | 1 | 3 | 105 | 145 |
| 2 | 1 | 30 | 132 | 172 |
| 3 | 4 | 234 | 339 | 379 |

September 16, 2015 at 9:39:12 through September 17 at 9:39:12 [*3].

We placed restrictions on the generated data according to the Request for Comments (RFC) 1035 issued by the Internet Engineering Task Force (IETF) [12]. Each generated data point consisted of a 17-letter fixed block, and a random number greater than 1 of labels as a variable block. The length of each label in the variable block was greater than or equal to 1 and less than 64. The total length of the variable blocks was greater than or equal to 8 and less than 238. Note that the lower limit came from the fact that random queries tend to be above a certain length, and the higher limit comes from the RFC requirement that the maximum length of domain names be 255, including the last dot. Numbers, lowercase letters, and uppercase letters appear in a variable block. The generated data had the same proportion of one-label and multiple-label variable blocks.

**Table 5** shows the contents of the data sets. Note that the program treated duplicated domain names in the same window as one subject, and so the number of subjects became decreased as the window length increased.

#### 5.2.2 Analysis Subjects for Detection Rate Evaluation

Analysis subjects for the detection rate evaluation were also provided as a file whose form is the same as the training data.

The captured data of the analysis subjects was obtained during the following periods [*4]:

**AS-1** September 17, 2015 at 9:39:12 - September 18 at 9:39:12
**AS-2** September 18, 2015 at 9:39:12 - September 19 at 9:39:12
**AS-3** September 19, 2015 at 9:39:12 - September 20 at 9:39:12
(Data used varied with the type of evaluation)

Table 5 shows the contents of the data sets for the analysis subjects.

#### 5.2.3 Analysis Subjects for Performance Evaluation

The performance evaluation was conducted, varying the lengths and the number of labels of the queried domain names. **Table 6** shows the forms of the queried domain names. In this evaluation, analysis subjects were packets transmitted from the traffic generator, which packets were composed of queries imitating those sent by an attacker and supposed responses for them in the same proportion. The variable blocks of the queried domain names varied by query, but they had fixed lengths due to the limitations of the generator.

The queries had only one question of an A record that asked a random domain, and the responses were an authoritative answer of the NXDOMAIN for the queries. Neither queries nor responses were fragmented, and so one packet was composed of either one query or one response.

We sent the generator-transmitted traffic to the detection machine for 10 minutes and confirmed whether all the packets were analyzed. If the amount of traffic exceeded the capacity of the program, the internal buffer would store waiting subjects, until no more space was available.

**Table 7** Patterns tested in the parameter investigation.

| Param. [*6] | Pattern | Intervals | Detection Rate (%) |
|---|---|---|---|
| 2 | A | 1–9, 10– | 96.24 |
| | B | 1–5, 6–9, 10– | 96.30 |
| | C | 1–3, 4–6, 7–9, 10– | 96.34 |
| | D | 1, 2, 3, 4, 5, 6, 7, 8, 9, 10– | 96.42 |
| 3 | A | 1, 2–7, 8– | 96.31 |
| | B | 1, 2–4, 5–7, 8– | 96.24 |
| | C | 1, 2–3, 4–5, 6–7, 8– | 96.42 |
| | D | 1, 2, 3, 4, 5, 6, 7, 8– | 96.34 |
| Param. | Pattern | Window Width | Detection Rate (%) |
| 4 | A | 10 minutes | 94.57 |
| | B | 1 hour | 94.89 |
| | C | 3 hours | 94.89 |
| | D | 6 hours | 94.57 |

### 5.3 Parameter Investigation

Before conducting the evaluation described above, we investigated the parameters that could achieve the best detection rate. The parameters that we investigated were Parameters 2, 3, and 4. Parameters other than an investigation target were fixed as showed in Table 3.

The best settings for Parameters 2 and 3 were investigated using AS-1, 2, and 3 as input subjects. We found the optimum intervals by the following two steps:

( 1 ) Determine the size of the remaining, right-most division; and
( 2 ) Determine the sizes of intermediate divisions.

#### 5.3.1 Intervals of the Variable Block Length

Conducting experiments, we found that the remaining division should denote subjects whose lengths are 10 or greater. Next, we varied the sizes of the intermediate divisions. **Table 7** shows the patterns tested and the results of detection rates, which indicates the best detection rate was achieved with Pattern D.

#### 5.3.2 Intervals of the Number of Labels

We found that the remaining division should denote subjects with 8 or more labels. The results of the experiments that we varied the sizes of the intermediate divisions are also given in Table 7. The best detection rate was achieved with Pattern C.

#### 5.3.3 Window Width

To improve the latency of detection, a window width should be as short as possible without significantly impairing the detection rate. Note that latency refers to the time between when a suspicious subject arrives and when it is detected, and it can be approximately the same as the window width.

In our experiments, we used only AS-1 for input subjects. The

[*3] The dates are based on Japan Standard Time (JST), which is UTC+09:00.
[*5] excluding dot delimiters.
[*6] The numbers correspond to Table 3.

© 2016 Information Processing Society of Japan

**Table 8** Differences of detection rates when varying employed feature values.

| Combination of Feature Values | Detection Rate (%) | False Positives (%) | False Negatives (%) |
|---|---|---|---|
| 1, 2, 3, 4 | 94.57 | 1.14 | 4.28 |
| 1, 2, 3 | 94.20 | 3.14 | 2.66 |
| 1, 2, 4 | 92.28 | 1.49 | 6.22 |
| 1, 3, 4 | 92.64 | 4.35 | 3.01 |
| 2, 3, 4 | 95.21 | 1.22 | 3.57 |
| 1, 2 | 89.20 | 8.02 | 2.78 |
| 1, 3 | 91.86 | 5.01 | 3.12 |
| 1, 4 | 91.71 | 1.50 | 6.79 |
| 2, 3 | 95.20 | 2.71 | 2.09 |
| 2, 4 | 90.36 | 1.00 | 8.64 |
| 3, 4 | 93.70 | 4.47 | 1.82 |
| 1 | 84.16 | 14.44 | 1.40 |
| 2 | 65.26 | 33.36 | 1.38 |
| 3 | 83.86 | 14.57 | 1.57 |
| 4 | 90.21 | 0.94 | 8.85 |

window width was adjusted to be 10 minutes, 1 hour, 3 hours, and 6 hours. Then, the detection rates for each window width were compared.

Table 7 shows the results. The detection rates did not vary significantly with varying window width; the difference between the best case and the worst case was only 0.32%.

Since a domain name asked by an attack query varies for each query, it occurs once regardless of window width. Thus, window width would only affect the number of occurrences of innocent queries. However, the experiments showed no significant difference, indicating that the shortest window used (10 minutes) was wide enough to detect an attack. Prioritizing quick detection over a possible small improvement in detection rate, we chose to use the 10-minute window.

### 5.4 Effectiveness of Each Feature Value

We conducted experiments to verify the effectiveness of each feature value. In the experiments, we varied a combination of employed feature values and compared detection rates. We used AS-1 as a data set and the parameters of each feature value are set to the best confirmed in Section 5.3. The results are showed in **Table 8**. We obtained higher detection rates when employing these combinations:

( 1 ) Combination of 1, 2, 3, and 4;
( 2 ) Combination of 1, 2, and 3;
( 3 ) Combination of 2, 3, and 4;
( 4 ) Combination of 2 and 3.

In the combinations listed above, the combination of 1, 2, 3, and 4 achieved the least false positive rates. We think that the less the false positives are, the better it is as long as detection rates are almost the same. Therefore, we employ the combination of 1, 2, 3, and 4.

The reason why we focus on less false positives is that many legitimate users are misjudged as attackers if a false positive rate is high. If attack queries are blocked simply based on our method, this type of misjudgment makes many innocent users denied. The more the false positives are, the greater this bad effect becomes.

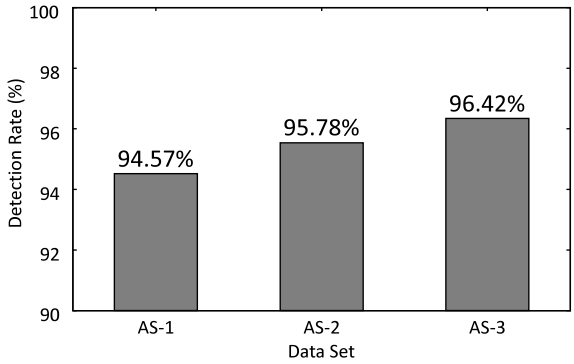


**Fig. 5** Detection rates for each data set.

**Table 9** Confusion matrix with AS-3.

| | | Predicted | |
|---|---|---|---|
| | | Random | Non-Random |
| Actual | Random | 3,132 (33.88%) | 250 (2.70%) |
| | Non-Random | 81 (0.88%) | 5,782 (62.54%) |
| | | Accuracy | 96.42% |
| | | Error Rate | 3.58% |

**Table 10** Results of the performance evaluation.

| Pattern | Responses (pps [*7]) | Queries (Gbps) | Responses (Gbps) | Total (Gbps) | Packet Loss Occurred |
|---|---|---|---|---|---|
| 1 | 1,100,000 | 1.10 | 1.45 | 2.55 | No |
| | 1,150,000 | 1.15 | 1.52 | 2.67 | No |
| | 1,200,000 | 1.20 | 1.58 | 2.78 | Yes |
| 2 | 900,000 | 1.09 | 1.38 | 2.47 | No |
| | 950,000 | 1.16 | 1.46 | 2.62 | No |
| | 1,000,000 | 1.22 | 1.54 | 2.76 | Yes |
| 3 | 400,000 | 1.15 | 1.28 | 2.43 | No |
| | 450,000 | 1.29 | 1.44 | 2.73 | No |
| | 450,000 | 1.44 | 1.60 | 3.04 | Yes |

## 6. Results

### 6.1 Detection Rate

Here, we evaluated the difference in the detection rates between the different data sets. In the experiments, we used AS-1, 2, and 3 as input subjects. **Figure 5** compares the detection rates. The chart indicates that our method has an average detection rate of 95.59% with a worst case of 94.57% for AS-1, and it achieves the stable detection rates regardless of the data sets.

**Table 9** details the accuracy and the error rate of our method with AS-3. The number of false negatives was approximately three times that of false positives. We saw a similar tendency in the other data sets.

Note that true positives mean that queried domain names are just random in this experiments. In Section 7.1, we discuss how to determine whether positive subjects are really related to the Water Torture Attack.

### 6.2 Performance

**Table 10** shows the results of the performance evaluation. Our method can process at least 450,000 responses per second and analyze over at least 2.3 Gbps of traffic without missing any of them.

[*7] packets per second.

© 2016 Information Processing Society of Japan

## 7. Discussion

### 7.1 Use in Real Situations

The ability of our system is to detect whether queried domain names are random, and so, our system itself cannot determine whether they are really related to the Water Torture Attack, because random queries also emerge in C&C communications. However, the difference between them can be noticed by observing the amount of queries since the former is a type of DDoS attack and the latter is not so. Thus, it is easy to determine which type of attacks is occurring.

In addition, our system currently has the delay between arrival of attack queries and detection of them but the existence of the delay is not a critical issue to defend an attack. Even if the detection delays 10 minutes, our system is still effective for attacks that are supposed to continue for 10 or more minutes. Essentially, an ephemeral attack does not cause substantial damage to authoritative servers, and thus, our system focuses on long-lasting attacks.

### 7.2 Detection Rate

According to Table 9, the number of the false negatives is greater than that of the false positives. This tendency is desirable for automated systems that block attacks in real time unless the error rate becomes too high. If denying too many legitimate users, authoritative servers cannot offer sufficient services.

However, the lower the error rate, the better. Focusing on causes of the false negatives, we found that subjects that tended to be judged as non-random had variable blocks with the following features:

( 1 ) They included bigrams recorded as high-probability items on Reference Table 0, the reference table for non-randomness probability;
( 2 ) Their lengths were short; and
( 3 ) The same appeared more than once in the same window.

In other words, variable blocks of innocent subjects had these features.

We first presumed that multiple attack queries with the same variable block rarely appear so that we took the number of occurrence of the same variable block as a feature value. That made random subjects that emerge multiple times tend to be judged as non-random, which caused false negatives. Actually, such attack queries can arrive via different open resolvers, and so we should consider treatment of subjects whose number of occurrence is more than one hereafter.

Currently, $P(X_4|Y)$, a score derived from bigrams, is calculated by simply adding the probability of each bigram. Because of using addition, a subject tends to be judged as non-random if its variable block includes even one bigram that frequently appears in the training data. The reason why we employ addition instead of multiplication is that both the randomness and the non-randomness probabilities could be zero percent if a variable block includes bigrams that did not appear in the training data. However, considering that simple addition caused false negatives in that situation, the current calculation of the score would not be adequate. Thus, we should consider other ways of calculation as future work.

Finally, we should consider not only the feature values that we currently employed but also employing others to diminish errors.

### 7.3 Performance

The results indicate that our method is applicable to high-load systems, which could cover almost all situation. However, considering using our method for super-large-scale servers such as provided by large ISPs, we still do not satisfy the current performance. It has been reported that when an attack occurs, an authoritative server receives 800,000 queries per second [13]. If the lengths of queried domain names are short, our method can analyze all queries and detect an attack. However, if the lengths are excessively long, our method could miss some queries.

Basically, the throughput decreases as the lengths of queried domain names become longer because the program scans almost all of the whole response to seek a variable block due to the structure of DNS packets. However, we have room for improvement regarding the performance. We previously confirmed that the hardware of the detection machine itself had enough capacity to 10 Gbps of traffic. A major cause of the decrease in the throughput is the extraction of variable blocks from responses. The calculations of the randomness and the non-randomness probabilities are simple, floating-point arithmetic operations, and so the extraction of variable blocks occupied a large portion in the whole process. If authoritative servers that our method monitors manage only one zone, our method could run faster by arbitrarily determining a fixed block and omitting to refer to the authority section.

In addition, our program is currently single-threaded. However, the Reference Tables becomes constant after the pre-training step, and so we could analyze subjects in parallel to improve the throughput.

## 8. Conclusion

We presented a detection method for a type of Distributed Denial of Service attack on the Domain Name System, known as the Water Torture Attack. Attack queries have the distinctive feature of asking non-existent domain names with random-string subdomains. To detect the attack, our method picks out such queries using the Naive Bayes Classifier. The evaluation results indicate that our method is capable of detecting the attack with a 95.59% detection rate. Moreover, our method is fast enough to use during an attack alert system, even for large-scale servers. In future work, we plan to reconsider feature values to improve the detection rate and also consider further improvement of the performance for super-large-scale systems.

### References

[1] Secure64: 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] Okayasu, S. and Sasaki, R.: Evaluation of Method for Detecting C&C Server of Botnet Using the Latest Data (in Japanese), *Proc. 2014 Computer Security Symposium*, Vol.2014, No.2, pp.175–182 (2014).
[3] Karasaridis, A., Meier, H.K. and Hoeflin, D.: NIS04-2: Detection of DNS Anomalies using Flow Data Analysis, *Global Telecommunications Conference, GLOBECOM '06*, pp.1–6, IEEE (2006).
[4] Joe, M.: Random dns queries with random sources, The Tri Tech Group (online), available from ⟨http://www.gossamer-threads.com/lists/nanog/users/169123⟩ (accessed 2015-11-30).

© 2016 Information Processing Society of Japan

[5] Nishida, K.: Water Torture: A Slow Drip DNS DDoS Attack on QTNet, Kyushu Telecommunication Network Co., Inc. (online), available from ⟨http://www.slideshare.net/apnic/dnswatertortureonqtnet-1425130417-1425507043⟩ (accessed 2015-11-30).
[6] Kazato, Y., Fukuda, K. and Sugawara, T.: Towards Classification of DNS Erroneous Queries, *Proc. 9th Asian Internet Engineering Conference*, pp.25–32 (2013).
[7] Yadav, S., Reddy, K.K.A., Reddy, A.L.N. and Ranjan, S.: Detecting Algorithmically Generated Malicious Domain Names, *Proc. 10th ACM SIGCOMM Conference on Internet Measurement*, pp.48–61 (2010).
[8] Schiavoni, S., Maggi, F., Cavallaro, L. and Zanero, S.: Phoenix: DGA-Based Botnet Tracking and Intelligence, *Proc. Conference on Detection of Intrusions and Malware & Vulnerability Assessment* (*DIMVA 2014*), pp.192–211 (2014).
[9] Amor, B.N., Benferhat, S. and Elouedi, Z.: Naive Bayes vs Decision Trees in Intrusion Detection Systems, *Proc. 2004 ACM Symposium on Applied Computing*, pp.420–424 (2004).
[10] Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Nimeh, A., Lee, W. and Dagon, D.: From Throw-Away Traffic to Bots: Detecting the Rise of DGA-Based Malware, *Presented as part of the 21st USENIX Security Symposium* (*USENIX Security 12*), pp.491–506 (2012).
[11] 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).
[12] 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).
[13] Weber, 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 2015-11-30).

**Yuya Takeuchi** received his B.E. degree in Computer Science and Engineering from Toyohashi University of Technology in 2014. He is currently a master's-degree student at the same university. His research interests include network security.

**Takuro Yoshida** received his B.E. degree in Computer Science and Engineering from Toyohashi University of Technology in 2015. He is currently a master's-degree student at the same university. 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 in Nagoya University from 2000 to 2008. He is currently a lecturer at Toyohashi University of Technology. 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. degrees in Systems and Information Engineering from University of Tsukuba respectively. He is now working for Internet Initiative Japan Inc. He is currently interested in network security.

**Hiroyuki Kishimoto** received his B.E., 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 40 G/100 G environment.

© 2016 Information Processing Society of Japan