HAL open science

Name Filter: A Countermeasure against Information Leakage Attacks in Named Data Networking

Daishi Kondo, Thomas Silverston, Vassilis Vassiliades, Hideki Tode, Tohru Asami

To cite this version:

Daishi Kondo, Thomas Silverston, Vassilis Vassiliades, Hideki Tode, Tohru Asami. Name Filter: A Countermeasure against Information Leakage Attacks in Named Data Networking. IEEE Access, 2018, pp.65151 - 65170. 10.1109/ACCESS.2018.2877792 . hal-01946259

HAL Id: hal-01946259

https://hal.science/hal-01946259v1

Submitted on 5 Dec 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Name Filter: A Countermeasure against Information Leakage Attacks in Named Data Networking*

DAISHI KONDO[1,2], (STUDENT-MEMBER, IEEE), THOMAS SILVERSTON[3], VASSILIS VASSILIADES[4], HIDEKI TODE[5], (MEMBER, IEEE), AND TOHRU ASAMI[6], (MEMBER, IEEE)

[1]University of Lorraine, LORIA (CNRS UMR 7503), France
[2]Inria Nancy - Grand Est, France
[3]Shibaura Institute of Technology, Japan
[4]University of Cyprus, Cyprus
[5]Osaka Prefecture University, Japan
[6]Advanced Telecommunications Research Institute International, Japan

Corresponding author: Daishi Kondo (e-mail: daishi.kondo@loria.fr).

*An earlier abridged version of this paper was presented at IEEE INFOCOM WORKSHOP (NOM) [8].
"This work is supported by the DOCTOR Project and funded by the French National Research Agency (ANR-14-CE28-0001). Also, this work is partly supported by the JSPS KAKENHI grant number JP17H00734."

ABSTRACT Named Data Networking (NDN) has emerged as a future networking architecture having the potential to replace the Internet. In order to do so, NDN needs to cope with inherent problems of the Internet such as attacks that cause information leakage from an enterprise. Since NDN has not yet been deployed on a large scale, it is currently unknown how such attacks can occur, let alone what countermeasures can be taken against them. In this study, we first show that information leakage in NDN, can be caused by malware inside an enterprise, which uses steganography to produce malicious Interest names encoding confidential information. We investigate such attacks by utilizing a content name dataset based on uniform resource locators (URLs) collected by a web crawler. Our main contribution is a name filter based on anomaly detection that takes the dataset as input and classifies a name in the Interest as legitimate or not. Our evaluation shows that malware can exploit the path part in the URL-based NDN name to create malicious names, thus, information leakage in NDN cannot be prevented completely. However, we illustrate for the first time that our filter can dramatically choke the leakage throughput causing the malware to be 137 times less efficient at leaking information. This finding opens up an interesting avenue of research that could result in a safer future networking architecture.

INDEX TERMS Firewall, Information leakage attack, Name filter, Named data networking

I. INTRODUCTION

Currently, in the business world, information leakage through a targeted attack is an issue for many companies because it drastically impacts on their benefits and profitability [1]. For example, in 2013, the retail chain "Target" suffered from a 46% drop in profits after an attack and spent more than $100 million on a system upgrade to prevent another attack [2]. In order to perform information leakage, an attacker must first send malware to an employee of the targeted company, for example, through an email that appears to be legitimate so the employee opens the email and infects their computer with the malware. The malware then creates a communication channel to the outside attacker, who can remotely control the malware and finally steal confidential information from the company. As this security threat occurs as a result of accessing suspicious media, one countermeasure would be to provide employees with cybersecurity education [3]. However, it is almost impossible to eliminate all human error. Thus, it is critical to investigate the prevention of information leakage after a computer has been infected by malware under the assumption that it is difficult for the computer to avoid the infection.

In recent years, information centric networking (ICN) [4] has been proposed as a promising future networking architecture; it performs a shift from a host-to-host to a host-to-content communication paradigm. An ICN request packet includes a desired content name rather than an address of the content producer (in the Internet, an IP address of the

content server), and the packet moves toward a nearby content producer. As for the content producer, not only the actual content publisher but also an ICN node installing cache (i.e., in-network cache) can reply to this request. This means that the architecture focuses not on *where the content is* but *what content is being requested*, and therefore ICN is more suitable for host-to-content oriented applications such as video services, which are one of the major use cases in the current Internet. As one of the ICN architectures, we adopt named data networking (NDN) [5] since many researchers are currently paying attention to NDN and are attempting to deploy it for future Internet.

To be adopted at Internet scale, NDN must resolve inherent issues of the current Internet. Since information leakage is a big issue in the Internet and it is absolutely crucial to assess the risk before replacing the Internet with NDN completely, we investigate whether a new security threat causing the information leakage happens in NDN. Assuming that (i) a computer is located in the enterprise network that is based on an NDN architecture, (ii) the computer has been already compromised by suspicious media such as a malicious email, and (iii) the company installs a firewall connected to the NDN-based future Internet, we focus on a situation where the compromised computer (i.e., malware) attempts to send leaked data to the outside attacker.

NDN is essentially a "pull"-based architecture, and there are only two types of packets: *Interest* and *Data*, which are request and response packets, respectively. In order to retrieve content, a consumer first sends the Interest to the NDN network and then obtains the corresponding Data from the producer or the intermediate NDN node. In other words, they cannot send a Data unless they receive the Interest packet. Therefore, as one of the naive methods to mitigate information leakage through a Data, an enterprise network firewall can carefully inspect a Data to publish and produce it instead of the inside employee in the network (i.e., a whitelist). In this case, all the publicly-accessible content is on the firewall as the originals or their caches.

However, the firewall cannot manage a naming policy on the outside content and NDN forwarding nodes do not verify whether the name actually exists. This results in the risk of information leakage through an Interest since the malware can hide information such as customer information in the Interest name using *steganography* and send it toward the outside attacker. The malware can pretend to access outside content, so it is very difficult for the firewall to detect the information leakage attack. We argue that the information leakage attack through an Interest in NDN should be one of the essential security attacks at protocol level and it is important to develop a detection method for this attack. Against the information leakage attack through an Interest, we first propose a name filter using search engine information. When the filter receives an Interest, it performs a request to a search engine in order to check whether the Interest name is indexed by the search engine. This filter judges the indexed name as legitimate since the malicious name created by the malware is not published anywhere and therefore cannot be indexed. However, this filter cannot predict whether the unindexed names are legitimate because they are not always malicious (e.g., unindexed names such as newly generated or password-protected content should be legitimate). To overcome the limitation of the filter, we propose a name filter using an isolation forest [6]. An isolation forest is an unsupervised learning algorithm for anomaly detection, and therefore it is useful when there is an insufficient number malicious samples. We evaluate the performances of the proposed name filters by collecting uniform resource locators (URLs) from the data repository provided by Common Crawl [7]. Our experiments show that (1) the path part is useful for malware to create malicious names leaking data and hide the activities, (2) it is difficult to completely prevent information leakage even in NDN, and (3) the filters can dramatically choke the information leakage throughput and make malware send 137 times more Interest packets to leak information than when filters are not used.

To the best of our knowledge, our previous work [8] was the first to perform risk analysis on information leakage attacks through an Interest, and this paper is an extended version. We extend the previous work with the following contributions:

- investigating vulnerability of available elements other than a name in an Interest to perform information leakage,
- adopting a different algorithm (i.e., isolation forest) to build a name filter using name datasets,
- preparing larger name datasets and analyzing them deeply, and
- proposing and evaluating different ways to create malicious names – one of which can bypass the name filter more efficiently and achieve higher information leakage throughput.

The remainder of this paper is organized as follows. Section II describes an overview of NDN and explains a URL-based NDN name under the assumptions made in this paper. Section III explains information leakage attacks in NDN. Against the attack, Section IV proposes a name filter using search engine information and using an isolation forest. Also, a simple name filter against information leakage attacks through a Data is proposed. Section V describes the NDN name dataset corresponding to the collected URL dataset and its statistics. Using the dataset, Section VI evaluates the proposed name filters. Section VII discusses our results, and Section VIII presents related works. Finally, Section IX concludes the paper and proposes future research directions.

II. OVERVIEW OF NDN AND URL-BASED NDN NAME

In this section, we present an overview of NDN [5], and then we define a URL-based NDN name, which we adopt in this study.

A. OVERVIEW OF NDN

An NDN node consists of three components: a *forwarding information base (FIB)*, *pending Interest table (PIT)*, and *content store (CS)*. The FIB has routing information to forward Interests. PIT keeps all pending Interests and the arrival interfaces of the Interests into entries, and each entry is removed when the matching Data is received or a timeout occurs. The CS is used for caching Data packets with the caching replacement policy. When a producer publishes content, the producer must first advertise the name prefix (e.g., /DOCTOR/pub) for name routing to the NDN network. Then, in order to retrieve content, a consumer specifies the desired content name, including the producer's name prefix (e.g., /DOCTOR/pub/video), in the Interest and sends it to the NDN network. The corresponding Data can be returned from the content producer or the intermediate NDN node with the cached Data. After receiving the Data and the content producer's signature, the consumer verifies the signature.

B. URL-BASED NDN NAME

The NDN naming policy [9] refers to a uniform resource identifier (URI) defined by RFC 3986 [10], which means that the NDN name has hierarchical structure. The naming policy specifies `ndn` in the <scheme> part but ignores the <authority> part. In order to request a specific Data, the NDN name in the Interest may include the SHA-256 digest of the entire corresponding Data packet as the last name component, but the full name of every Data must include the digest. Moreover, according to "NDN Technical Memo: Naming Conventions" [11], the name can carry the segmenting, versioning, time stamping, and sequencing.

The URL in the Internet is similar to the name in NDN. RFC 1808 [12] defines a URL as <scheme>:// <net_loc>/<path>;<params>?<query>#<fragment> and the <net_loc> part indicates a host generating the URL (e.g., a fully qualified domain name (FQDN)). A user or an organization who owns this host creates URLs by following its own naming policy, and URLs must be unique. According to a report from Google [13], in 2008 there were one trillion unique URLs on the Internet, so we believe that people are accustomed to current URLs.

Considering the high affinity between many already published URLs and people who are familiar with them, we believe it is highly likely that NDN naming policy will become the natural evolution from the current URL naming policy. Schnurrenberger [14] has also discussed content names based on URL datasets. Moreover, in the case that the NDN naming policy is extended from the current URL naming policy, it would be very easy to translate the current numerous content names distributed in the Internet to the corresponding NDN names. Thus, we predict that the naming policy of NDN will be based on that of the URL.

We adopt the NDN naming policy, except for in the <authority> part. As mentioned previously, in the NDN naming policy, the <authority> part, which corresponds to the <net_loc> part in RFC 1808, is ignored. However, we consider that an authority described in the <net_loc> part can define the name uniquely, which means that the authority should not be ignored. So this idea is different from the NDN naming policy. Fig. 1 shows the URL-based NDN name we adopt in this paper. The <path> part is constructed from several name components. We omit the <params> part because this part is an option in file transfer protocol (FTP).

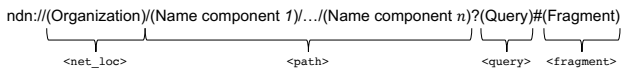


FIGURE 1: URL-based NDN name.

III. INFORMATION LEAKAGE ATTACK IN NDN

This section first introduces an information leakage attack through a Data and then one through an Interest. Moreover, as for the information leakage attack through an Interest, this section shows a more sophisticated attack to exploit an Interest name rather than simply adding leaked data into a name; this involves generating a steganography-embedded Interest name, which looks like a legitimate name at first glance, but is in fact a malicious one to leak data.

A. INFORMATION LEAKAGE ATTACK THROUGH DATA

After collecting confidential information from an enterprise network, malware can create Data packets of the information and associate them with the names that the outside attacker knows. In this case, in order for the attacker to retrieve the Data packets using the corresponding Interests, the name prefix used in the names must be routable from the attacker to the malware.

For example, in Fig. 2 it is assumed that the two name prefixes "/DOCTOR/priv" and "/DOCTOR/pub", which are created by the enterprise, are utilized to share the private and the public content, respectively. When employees perform private content sharing between them inside the enterprise network, the name prefix "/DOCTOR/priv" should be used. If an Interest with the name prefix is sent to the enterprise network from the outside network, based on the name prefix "/DOCTOR/priv", the firewall drops the Interest (i.e., a blacklist). The name prefix "/DOCTOR/pub" is used to publish the public content such as a web page, but the malware can also exploit the name prefix for information leakage as follows. In advance, the attacker and the malware share the same name components, which should be appended to the name prefix "/DOCTOR/pub", and then the attacker attempts to retrieve the Data created by the malware, which should be named "/DOCTOR/pub/*(the shared name components)*", using the corresponding Interest. Goergen et al. [15] propose a firewall to prevent some types of content (such as .doc and .pdf documents) from being shared externally. However, they do not discuss an attack caused by malware at all, so their firewall is not sufficient to protect against a more sophisticated attack.

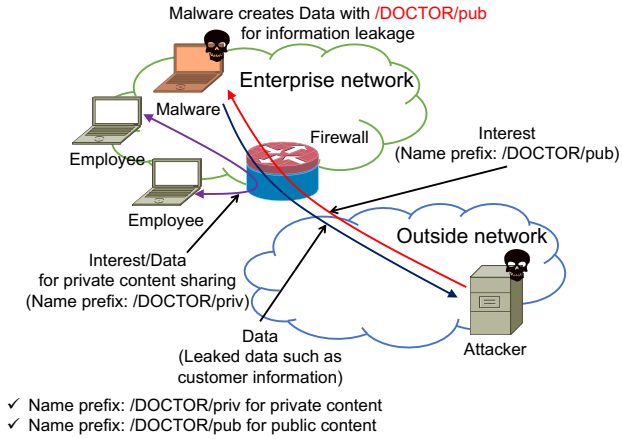


FIGURE 2: Information leakage attack through Data.

B. INFORMATION LEAKAGE ATTACK THROUGH INTEREST

Assuming that an attacker's name prefix is routable from the malware, by hiding (or simply adding) leaked data into Interests and sending them out from the enterprise network, the malware can leak the confidential information to the attacker. In particular, when the malware exploits the content names in the Interests to hide the data, it is very difficult for the company firewall to detect the information leakage since the Interests seem to attempt to retrieve an outside producer's Data packets associated with the names. This paper focuses only on information leakage attacks through an Interest name. Exploitation of the other available elements in the Interest such as Nonce [16] to perform an information leakage attack is discussed briefly in Section III-B4.

1) Taxonomy of Information Leakage Attack through Interest

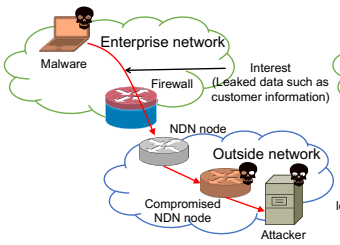


FIGURE 3: One-way Interest.

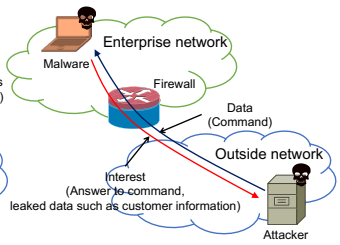


FIGURE 4: Interest/Data.

There are two possible methods for performing an information leakage attack through an Interest: the (*one-way Interest method* shown in Fig. 3) and the (*Interest/Data method*, which exploits an Interest with the corresponding Data, shown in Fig. 4). The features of these methods are summarized in Table 1.

As for the one-way Interest method, the malware transmits the leaked data from the enterprise network by sending the malicious Interests to the attacker. The malware must know the name prefixes toward the compromised NDN nodes (e.g., Wi-Fi AP installed by the attacker) in order to forward the Interests to the attacker. In this method, the attacker does not reply to the malware with any Data packets, so the attacker cannot have fine-tuned control of the malware. Therefore, in the case that some Interests have been lost, the attacker cannot request a retransmission of the missing information. This can happen when the firewall drops the Interests or the PIT is overflowed. In order to deal with the dropped Interests, the attacker may use erasure coding such as LT codes [17] and Raptor codes [18]. Although this method is not the most efficient to leak information, the main advantage of it is the preservation of attacker anonymity since the attacker only receives malicious Interests and does not reply to them with any Data packets, which include the attacker's signature.

TABLE 1: Taxonomy of information leakage attack through Interest

Feature	One-way Interest	Interest/Data
Malware remote control	No	Yes
Retransmission	No	Yes
Attacker anonymity	Yes	No†
Erasure coding	Yes	No
PIT overflow	Yes	No

† Yes for some cases such as exploiting bots.

As for the Interest/Data method, assuming that the attacker's name prefix is routable from the malware, the attacker explicitly controls the malware by utilizing Interest and Data packets. Thus, the attacker can communicate with the malware. In the case of a packet drop, the attacker can request the retransmissions of the missing Interest packets by exploiting the Data packets, so there is no need to utilize erasure coding. This method is more efficient than the one-way Interest method, but once the attack is detected, it is possible for the attacker to be tracked as the attacker's signature has been included in the Data packets. The attacker, however, can control bots remotely and avoid being tracked.

Hereafter, in order to assess the risk of information leakage, we focus on the Interest/Data method since it is more efficient than the one-way Interest method.

2) Information Leakage Attack through Steganography-Embedded Interest Name

Generating a *steganography-embedded Interest name* to act as a legitimate one (in fact, a malicious one to leak data) should be a more sophisticated attack to exploit an Interest name than simply adding leaked data into a name. As a similar use case of steganography, Mason et al. [19] propose English Shellcode to encrypt and hide information, which transforms the Shellcode to be similar to English prose. To create a steganography-embedded Interest name, as shown in Fig. 5, the attacker and the malware must share the same table for steganography, which includes each token (strings) and the corresponding digits. Note that in order to describe how to generate and utilize the steganography embedded

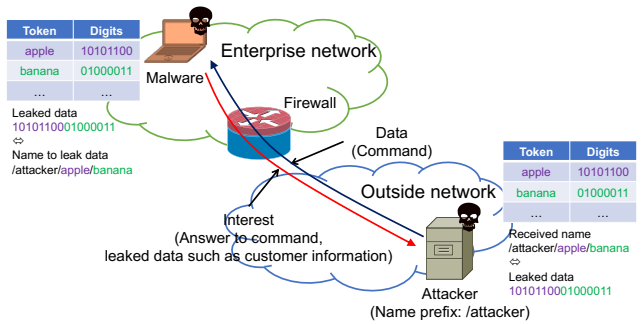


FIGURE 5: Information leakage attack through steganography-embedded Interest name.

Interest name, Fig. 5 adopts the Interest/Data method, but the description is also available in the one-way Interest method. Assuming that the malware knows the attacker's name prefix "/attacker" and the name prefix is routable from the malware, the malware converts the leaked data 1010110001000011 to the Interest name "/attacker/apple/banana" using the table of steganography. After the malware sends the Interest along with the name to leak data, the attacker decrypts the name using the shared table and obtains the leaked data 1010110001000011.

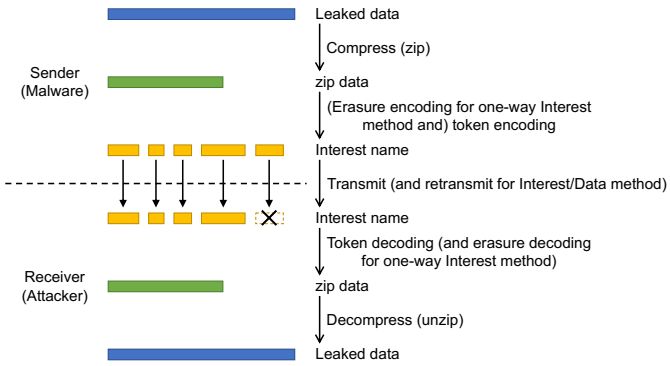


FIGURE 6: Transmission of leaked data from malware to attacker.

Fig. 6 illustrates a general framework for transmitting leaked data from malware to the attacker. First, the malware compresses collected data to a format such as zip. For the one-way Interest method, the malware further encodes the compressed data with erasure encoding. To bypass a firewall and perform information leakage through an Interest name, the malware further encodes the output data with token encoding for steganography. Then, it creates and adds a malicious name to leak the data into an Interest.

3) Propterties of Malware Generating Malicious NDN Names

When malware creates a malicious name to leak data, there is a naive idea to ameliorate the information leakage throughput: *adding the data into the name as much as possible*. However, the properties of the generated name may not be similar to those of a legitimate name. For example, when the leaked data is composed of hexadecimal digits, the name, which includes as much leaked data as possible, may be much longer than the legitimate name and the frequencies of letters and the other printable characters in the malicious name could be different from the ones in the legitimate name. Therefore, it is potentially easy to detect the name as malicious.

As discussed in Section III-B2, malware can create a steganography-embedded Interest name as a more sophisticated attack. In this attack, each token corresponds to the digits, and after the attacker gathers several tokens from the malware, they can be decoded and then become the leaked data. Thus, to improve the throughput, it is essential to *add as many tokens into the name as possible*. To do so, the malware can exploit the path and the query in the name shown in Fig. 1. As for the fragment, in the Internet, the fragment is resolved by a browser and is not sent to the server by the browser [20], so the fragment cannot be used to add the tokens. When adding the tokens, the malware must concatenate them with delimiters in order to allow the attacker to decrypt the steganography-embedded Interest name safely (e.g., assuming that "/attacker.com/aaa" is a steganography-embedded Interest name and the token table includes "a", "aa", and "aaa", the leaked data cannot be decrypted uniquely). As for the delimiters between the tokens, we use a slash character ("/") in the path and equals and ampersand characters ("=", "&") in the query. Since the tokens are used, the frequencies of letters and the other printable characters in the path and the query of the created name may follow the ones of a language used in the tokens (e.g., English).

A naive information leakage throughput improvement method by malware will adopt four properties, as shown in Table 2. By adding as many tokens into the name as possible, the number of "/" in the path and "=" in the query increases (*Property 1*), and therefore the lengths of the path and query become large (*Property 2*). In addition, to improve the throughput, the length of token, which corresponds to the leaked data, should be small (*Property 3*), and by exploiting the tokens, the frequencies of letters and the other printable characters in path and query follow the ones of a language used in the tokens (*Property 4*). The number of equals characters is related to the number of ampersand ones since they are used to construct and concatenate key-value pairs (e.g., key1=value1&key2=value2), and therefore we do not consider the number of ampersand characters as one of the *Properties*.

TABLE 2: Properties of malware generating steganography-embedded Interest name

Property 1	The number of "/" in the path and "=" in the query will be large.
Property 2	The length of path and query will be large.
Property 3	The length of the token will be small.
Property 4	The frequencies of letters and the other printable characters in the path and query will follow ones of a language used in the tokens.

4) Vulnerability of the Other Available Elements in Interest

In the current specification of an Interest [16], available elements other than a name are CanBePrefix, MustBeFresh, ForwardingHint, Nonce, InterestLifetime, HopLimit, and Parameters. All elements except the name are optional, but as for the Nonce, which is used to detect looping Interests, it is required when an Interest is forwarded over the network links. Thus, we assume that for malware to send malicious Interests to the attacker, the malware must create them with at least a name and a Nonce.

In the elements, CanBePrefix and MustBeFresh cannot have a value. The InterestLifetime and the HopLimit can include only a non-negative integer and a 1-byte unsigned integer as a value, respectively; this restriction makes it hard for the malware to utilize these elements to hide the data. On the other hand, the name, ForwardingHint, Nonce, and Parameters can be exploited to perform an information leakage attack through an Interest since they are of variable length, except for the Nonce, which is a randomly-generated 4 byte string. However, other than the name and the Nonce, these are optional, as mentioned previously. Therefore, we do not take the ForwardingHint and the Parameters into account to perform an information leakage attack through an Interest.

In this paper, as for the Nonce, we utilize it to express a sequence number that is needed for a retransmission of some missing Interests in the Interest/Data method. Essentially, the Nonce should be randomly generated, so the malware and the attacker must produce random numbers corresponding to the sequential numbers and share them in advance. Otherwise, their malicious activity may be detected because of the Nonce, which is a sequential number. As mentioned in Section II-B, the name may include the SHA-256 digest of the corresponding entire Data packet as the last name component, segmenting, versioning, time stamping, and sequencing, which are all optional. The size of the SHA-256 digest is 32 bytes, and therefore it could be useful to leak the data. However, in the Interest/Data method, after receiving the SHA-256 digest from the malware, the attacker must create and send the Data whose SHA-256 digest corresponds to the digest appended in the Interest. To generate the Data, the attacker must break the second pre-image resistance property of the SHA-256, so it should be extremely difficult to produce the corresponding Data. Thus, the SHA-256 digest cannot be used to leak the data. As for the segmenting, versioning, time stamping, and sequencing, similarly to InterestLifetime and HopLimit, they have a restriction (e.g., a sequential number), and therefore they are not useful for the attack.

Hereafter, we assume that a malicious Interest is composed of a name and a Nonce, and the name does not include the SHA-256 digest, segmenting, versioning, time stamping, or sequencing.

IV. NAME FILTER

We propose several NDN name filters against an information leakage attack through a Data and through an Interest, as introduced in Section III. The proposed name filters are installed in the enterprise firewall.

A. NAME FILTER AGAINST INFORMATION LEAKAGE ATTACK THROUGH DATA

NDN is a "pull"-based architecture, so a producer cannot send the Data unless they receive the corresponding Interest from the consumer. Thus, although malware sends the Data toward the firewall before receiving the corresponding Interest, the firewall cannot forward it to the outside network since there is no PIT entry for the Data. Thanks to this big advantage in NDN, an information leakage attack through a Data can be prevented easily by making a name filter based on a whitelist. When the firewall builds the name filter, it must follow the policies below.

- *Policy 1*: The firewall must carefully inspect content and determine whether it can be published to the outside network.
- *Policy 2*: The firewall must accept only Interests including whitelisted names from the outside network, otherwise it must drop the other Interests from the outside network.
- *Policy 3*: The firewall must publish all the content accepted to be public to the outside network instead of the inside employees.

In this case, all the publicly accessible content is on the firewall, and therefore when employees attempt to publish content to the outside network, they must ask the firewall beforehand. Once the firewall follows these policies and creates the whitelist-based name filter, the name filter can eliminate the information leakage attack through a Data.

B. NAME FILTER AGAINST INFORMATION LEAKAGE ATTACK THROUGH INTEREST

Considering regular employee behavior to access outside content from the enterprise network, we first make the below assumptions.

- An employee accesses an outside content name via a search engine.
- The employee also contacts an outside content name, which is not indexed by the search engine, directly (e.g., password-protected content).
- By managing a content access policy, the firewall can prohibit access to unwanted content names and also explicitly define a whitelist with names that can be accessed.

1) Name Filter Using Search Engine Information

Assuming that search engines will still exist even in future Internet, the search engine may serve to detect a legitimate name since a malicious name to leak the information from the enterprise network is not indexed by the search engine (the malicious name is created by malware, and therefore the name is not published anywhere). Thus, we propose a name filter using search engine information. When the filter

receives an Interest, it performs a request to a search engine to determine whether the Interest name is indexed by the search engine. If the name is found by the search engine, the filter considers the Interest as legitimate. It is simple to implement this filter, but, as will be explained next, there is a limitation in terms of today's Internet use.

According to [21], search engines index only 4% of all content (referred to as Surface Web), and the remaining 96% are not indexed (referred to as Deep Web). In addition, the number of Internet users using search engines has decreased from 55% in 2014 to only 49% in 2015 [22]. One of the factors contributing to this decrease is the mobile era. Since it is difficult for users to search content with a small-size mobile screen, they prefer accessing content via other methods such as social networks. However, from an enterprise perspective, it is usually prohibited for employees, who work with company-supplied personal computers, to use social networks in order to increase labor productivity and prevent information leakage. Following these conventions, the firewall in an enterprise must define a content access policy to allow employees to access outside content using a search engine or an authorized link toward the content.

As most content comes from the Deep Web, much of it is still accessible by employees. For example, accessing content that is unreachable from a search engine, such as newly generated or password-protected content, should be legitimate behavior. Therefore, a naive name filter using search engine information will not be accurate enough to determine whether a name is legitimate. Indeed, this filter cannot be aware of most content names that are not indexed. Thus, there is a need to add further information to improve the efficiency of the name filter.

2) Name Filter Using Isolation Forest

To overcome the established limitations, we propose a name filter using an isolation forest [6]. The isolation forest is an unsupervised learning algorithm for anomaly detection, and therefore it is useful in the case that there are too few malicious samples. Regarding NDN architecture, as it has not been deployed yet, there is neither a legitimate NDN name dataset nor a malicious one available. Thus, assuming that an NDN name will be based on URL as stated in Section II-B and that the legitimate NDN name dataset is derived from the URL dataset, we extract 122 features from each of the NDN names and build the filter (see Table 3). We also extract the frequencies of characters in the NDN name, and for this, we use the function F, which is defined below.

Let $\mathcal{S}$ be the set of all printable characters according to RFC 3986 [10]. Let $\mathcal{L} \subset \mathcal{S}$ be the set of letters (i.e., a ...z and A ...Z), and let $\mathcal{C} = \mathcal{S} \setminus \mathcal{L}$ be the set of the other printable characters that are not letters (i.e., 0123456789~!@#$%&*()-_=+;:',./?[]). We use $\mathcal{N}$ to denote the set of all possible NDN names constructed with characters in $\mathcal{S}$.

We define the function

$$F(\cdot,\cdot) : \mathcal{N} \times \mathcal{S} \to \mathbb{R},$$
$$F(n,c) = \frac{\text{\# of occurrences of } c \text{ in } n}{\text{size of } n},$$

which takes an NDN name n and a character c as input from $\mathcal{N}$ and $\mathcal{S}$, respectively, and returns the frequency of c in n.

TABLE 3: Features extracted from NDN name

Notation	Feature variable
$N_{/}$	# of "/" in path
$N_{=}$	# of "=" in query
L_{Path}	Length of path
L_{Query}	Length of query
$F(Path,\ell)$	Frequency of the letter $\ell \in \mathcal{L}$ in path (26 dimensions)
$F(Path,c)$	Frequency of the character $c \in \mathcal{C}$ in path (33 dimensions)
$F(Query,\ell)$	Frequency of the letter $\ell \in \mathcal{L}$ in query (26 dimensions)
$F(Query,c)$	Frequency of the character $c \in \mathcal{C}$ in query (33 dimensions)

V. NDN NAME DATASET AND ITS STATISTICS

This section presents the NDN name dataset derived from the URL dataset and describes the statistics of the dataset.

A. NDN NAME DATASET

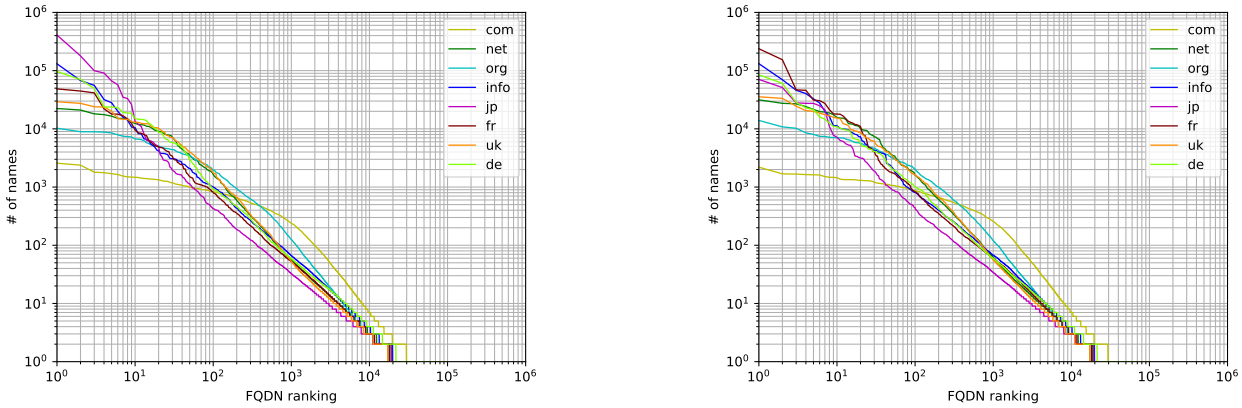
In order to infer properties of names commonly used in the Internet, we collect URLs from a data repository provided by Common Crawl [7]. At first, we obtain the crawl archive for February 2016, which holds more than 1.73 billion URLs, and we extract unique URLs belonging to eight top-level domains (TLDs): "com", "net", "org" and "info", which are generic top-level domains (gTLDs), and "jp", "fr", "uk", and "de", which are country code top-level domains (ccTLDs). In this dataset, the number of URLs varies greatly for each TLD (millions to billions), and therefore, in order to obtain the same number of URLs for each TLD, we extract two million URLs for each TLD randomly. In other words, we use 16 million URLs.

Then, assuming that a protector and an attacker collect a name dataset from their own crawlers independently and these two obtained datasets have no overlapping FQDNs, we divide the dataset, which includes 16 million URLs, into two datasets that include randomly chosen unique FQDNs: *one for the protector to make a name filter* (see Section VI-A1) and *the other for the attacker to create a malicious name* (see Section VI-A2).

Table 4 summarizes the name datasets for the protector and for the attacker. Fig. 7 shows FQDN ranking versus the number of names for each dataset. As for the definition of the FQDN ranking, an FQDN is ranked by the number of unique names it holds in each TLD dataset. Fig. 7 indicates that, despite having no overlap, the protector and the attacker's datasets have a similar relationship between FQDN ranking and the number of names.

B. STATISTICS

We analyze nine name attributes in each of the datasets:



(a) Protector's name dataset.

(b) Attacker's name dataset.

FIGURE 7: FQDN ranking vs. the number of names.

TABLE 4: Summary of name datasets

	Protector		Attacker	
	# of FQDNs	# of NDN names	# of FQDNs	# of NDN names
com	100,660	986,530	100,294	1,013,470
net	56,342	947,613	56,348	1,052,387
org	48,538	997,139	48,382	1,002,861
info	58,199	969,734	58,141	1,030,266
jp	58,900	1,383,230	58,847	616,770
fr	57,057	751,649	57,074	1,248,351
uk	53,459	1,008,608	53,457	991,392
de	65,109	1,091,233	65,036	908,767

1. (the number of slash characters ("/") in the path),
2. (the number of equal characters ("=") in the query),
3. (the length of the path),
4. (the length of the name component),
5. (the length of the query),
6. (average frequencies of letters in the path),
7. (average frequencies of the other printable characters in the path),
8. (average frequencies of letters in the query), and
9. (average frequencies of the other printable characters in the query).

For some names, no name component exists because some default pages such as index.html are often omitted (e.g., http://www.doctor-project.org/). In such a case, we set the length of the name component to zero. To obtain the average frequencies of letters and the other printable characters in the path and query, we first calculate their frequencies from each name, respectively. Then, we sum the obtained frequencies and average them within each TLD.

Table 5 shows the computed 90th, 95th, and 99th percentiles of attribute 1 to 5 for each TLD. After summing each percentile for eight TLDs, we divide the sum by eight to obtain the average (i.e., "overall"). The percentiles of these attributes from each TLD in the protector and attacker's name dataset are similar. Also, the percentiles of these attributes in the protector's name dataset are similar to the ones in the attacker's name dataset. The cumulative distribution function (CDF) of each attribute is given in Appendix A.

Figs. 8, 9, 10, and 11 show the average frequencies of letters and the other printable characters in the path and in the query. These figures all show that the average frequencies of letters and of the other printable characters from each TLD in the protector and attacker's name datasets are similar. Moreover, by calculating the cosine similarities[1] of "average frequencies of letters and the other printable characters in the path and query" between the protector's name dataset and the attacker's one, it can be seen that the respective average frequencies resemble each other (Table 6). Specific characters such as "-", "_", ".", and "%" are used more often than the others.

As explained in the following, the properties of these attributes can be discussed by considering search engine optimization (SEO). SEO is optimization to improve the rank of a web site in search engines such as Google so it appears in the top level of retrieval results by search engines. One of the methods to improve rank is to modify the structure of the URL. The documentation on SEO provided by Google [23] states that a URL should be composed of information that is easily understood by a crawler constructing a search engine as well as a user seeing the web site. Then, it is convenient for the crawler to obtain information easily and for the user to cite a link, and therefore this operation improves the rank. Moreover, Moz, which is an SEO company, reports policies to modify the structure of a URL [24]. These policies include "to make the URL human-readable", "to add keywords", "to shorten length", "to make URL suitable to the title of the web page", "to remove stop words such as "and", "or", and "but" while considering readability and length of the URL", "to avoid utilizing the unsafe characters written in RFC 1738 [25]", "to decrease the number of name components",

[1] $CosineSimilarity = \mathbf{A} \cdot \mathbf{B} / \|\mathbf{A}\| \|\mathbf{B}\|$.

TABLE 5: Computed percentiles

(a) gTLDs (com, net, org, info)

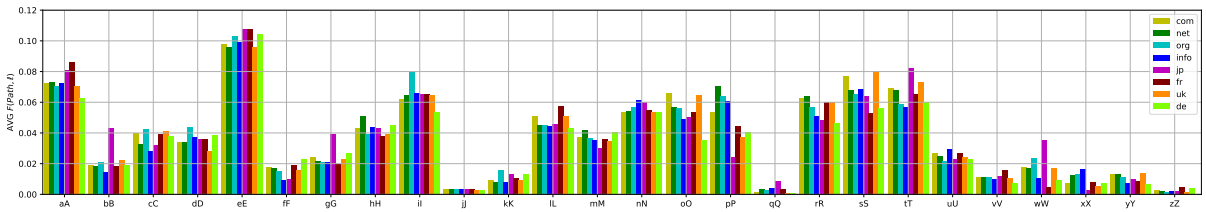
Attributes	com						net						org						info					
	Protector			Attacker			Protector			Attacker			Protector			Attacker			Protector			Attacker		
	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%
# of "/" in path	4	5	8	4	5	8	4	5	7	4	5	7	4	6	8	4	6	8	4	5	5	3	4	5
# of "=" in query	4	5	10	4	5	12	3	4	6	2	3	6	4	5	10	4	5	7	4	4	8	3	5	22
Length of path	73	88	127	74	89	135	68	80	109	64	79	124	68	86	128	65	84	130	70	84	120	57	70	101
Length of name component	26	40	69	26	41	69	25	40	71	23	35	62	19	32	66	20	32	66	25	37	64	27	39	64
Length of query	103	134	217	105	142	209	71	87	136	75	99	161	96	134	209	90	118	205	86	130	193	63	99	222

(b) ccTLDs (jp, fr, uk, de)

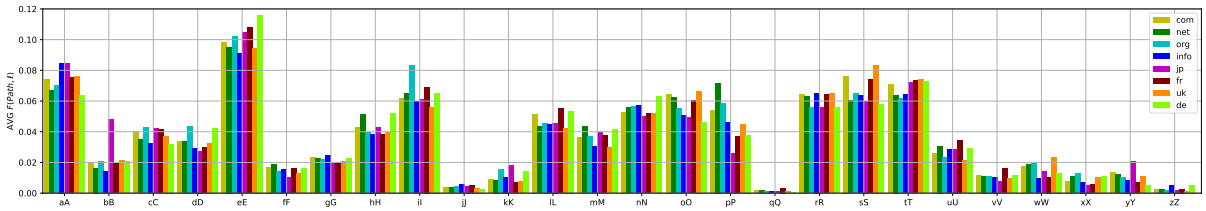
Attributes	jp						fr						uk						de					
	Protector			Attacker			Protector			Attacker			Protector			Attacker			Protector			Attacker		
	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%	90%	95%	99%
# of "/" in path	3	4	6	4	4	6	3	4	6	4	5	5	4	5	7	4	5	9	4	6	12	4	5	8
# of "=" in query	3	3	7	3	4	6	6	7	9	13	14	14	6	6	11	9	11	13	3	4	8	5	6	11
Length of path	112	185	190	82	99	200	89	110	115	95	106	126	80	95	126	74	88	117	73	92	121	97	108	129
Length of name component	83	104	117	18	48	99	35	57	77	55	82	108	29	44	73	27	43	76	20	33	67	39	64	101
Length of query	42	48	108	79	99	133	160	203	250	174	179	212	124	140	225	131	195	258	224	238	260	95	165	227

(c) Average (overall)

Attributes	overall					
	Protector			Attacker		
	90%	95%	99%	90%	95%	99%
# of "/" in path	4	5	7	4	5	7
# of "=" in query	4	6	9	5	7	14
Length of path	81	100	164	80	96	129
Length of name component	28	50	97	30	48	91
Length of query	112	178	238	100	144	221



(a) Protector's name dataset.



(b) Attacker's name dataset.

FIGURE 8: Average frequencies of **letters in the path**.

TABLE 6: Cosine similarities of average frequencies of letters and of the other printable characters between the protector and attackersâĂŹ name datasets

(a) Path

TLD	com	net	org	info	jp	fr	uk	de
Letter	1.00	0.998	0.999	0.989	0.985	0.992	0.996	0.994
Other	1.00	0.999	0.999	0.995	0.937	0.990	0.998	0.983

(b) Query

TLD	com	net	org	info	jp	fr	uk	de
Letter	1.00	0.995	0.998	0.981	0.922	0.973	0.994	0.945
Other	1.00	0.997	0.999	0.982	0.705	0.975	0.997	0.943

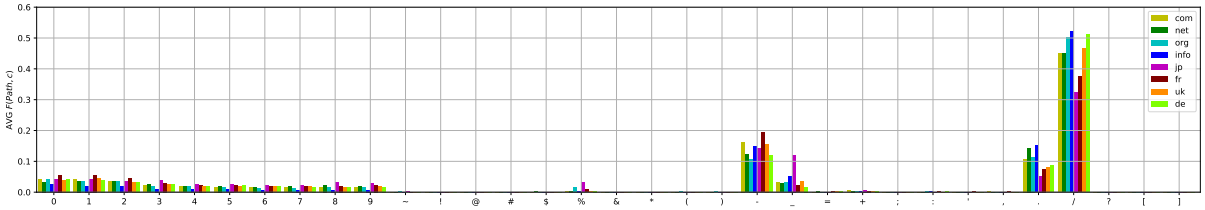
"to avoid utilizing the hash value", "to use "-" or "_" to separate words", "to avoid utilizing keywords repeatedly". Therefore, current URLs possibly follow the above rules to improve their rank, and consequently SEO affects the properties of these attributes.

VI. EXPERIMENTS

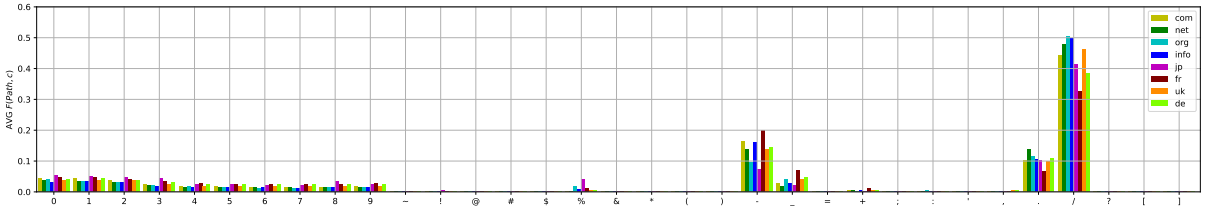
This section evaluates the performance of the proposed NDN name filters against steganography-embedded Interest names.

A. EXPERIMENTAL SETUP

In general, security exploits can be distinguished by two different types: *a1*) an attacker does not know the countermeasure by the protector and *a2*) an attacker knows the countermeasure but not its parameters. Similarly, counter-

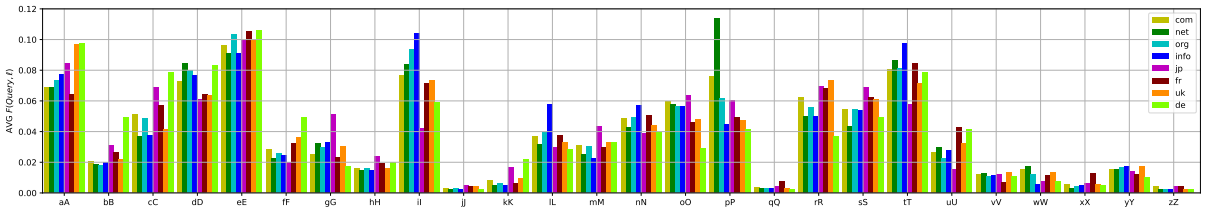


(a) Protector's name dataset.

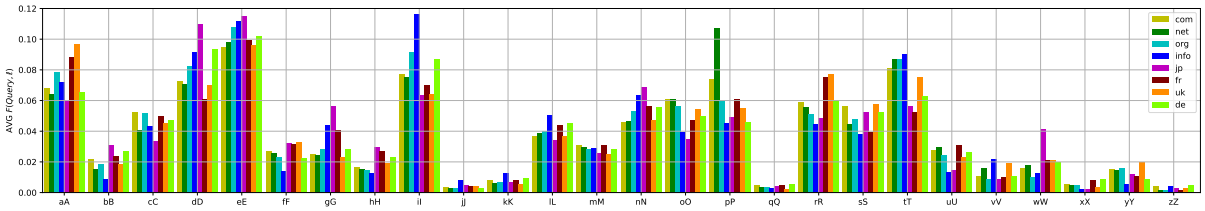


(b) Attacker's name dataset.

FIGURE 9: Average frequencies of **the other printable characters in the path**.

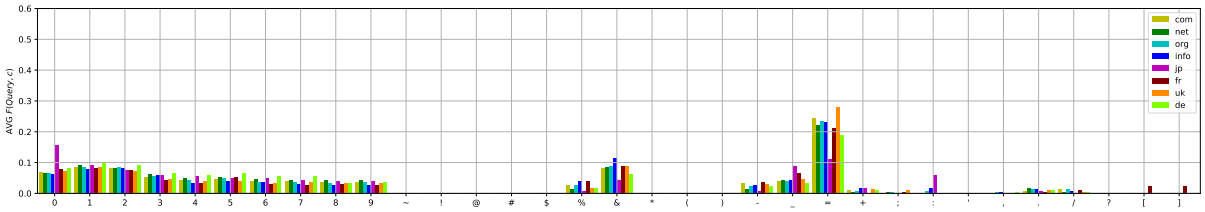


(a) Protector's name dataset.

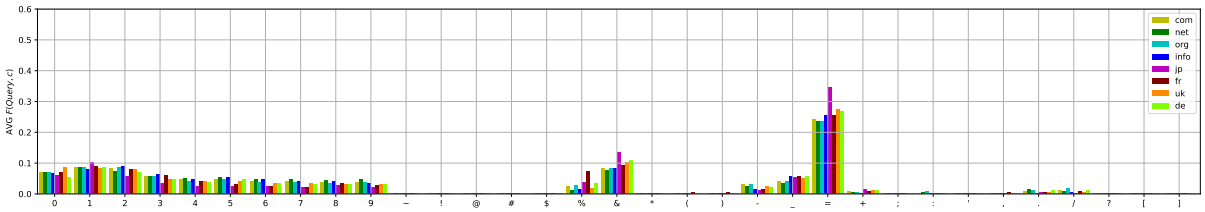


(b) Attacker's name dataset.

FIGURE 10: Average frequencies of **letters in the query**.



(a) Protector's name dataset.



(b) Attacker's name dataset.

FIGURE 11: Average frequencies of **the other printable characters in the query**.

measures can be also categorized by two different types: *p1*) a protector does not know the attack method and *p2*) a protector knows the attack method but not its parameters. We consider a scenario where the protector knows the attack method (i.e., information leakage attack through an Interest) and prepare the protection method (i.e., we consider *p2*). In order to study the upper limit of the information leakage throughput in the case *p2*, we analyze the risk of attack in the case that the attacker knows the countermeasure but not its parameters (i.e., we consider *a2*), and this case is beneficial to the attacker.

1) Protector

Fig. 12 demonstrates a flow to check an NDN Interest name using two name filters proposed in Section IV-B. The name should first be checked by the name filter using the search engine information, and when the name is dropped by the first filter, the name filter using the isolation forest (the second filter) inspects the name. The name filter using the search engine information is not efficient enough as malicious names are not indexed by nature. Thus, this section evaluates the sophisticated name filter using the isolation forest. The evaluation of the name filter using the search engine information is discussed in Section VII.

To build the name filter using the isolation forest, we use the legitimate NDN name dataset for the protector as the training dataset, which is summarized in Table 4. The dataset is based on eight name datasets separated by eight TLDs, and we prepare eight name filters using the isolation forest. For example, when an Interest carries a name including the "com" domain, the name should be checked by the filter derived from the "com" dataset. In addition, to utilize the isolation forest, as a parameter of the protector side, we must configure the proportion of outliers R_O^P in the dataset, which is one of the parameters of the isolation forest[2]. We prepare six outlier proportions: 0.01, 0.05, 0.1, 0.2, 0.3, and 0.4. Since the name dataset is considered as legitimate, the outlier proportion can be the false positive rate, which is defined as the ratio of the number of legitimate names identified as malicious to the total number of legitimate names.

2) Attacker

As mentioned previously, in order to analyze the risk of an information leakage attack, we consider the case that the attacker knows the countermeasure of the protector but not its parameters, which is beneficial to the attacker. Thus, to create malicious names with steganography to leak data, we assume that the attacker builds their own isolation forest-based filter using the NDN name dataset, which is shown in Table 4, in the same way as the protector's dataset.

We prepare three PDF files (Y.4001/F.748.2, Y.4412/F.747.8, Y.4413/F.748.5) from latest ITU-T recommendations [28] as leaked data. These PDF files include a variety of text and figures, which are common in technological documents. Specifically, in Y.4001/F.748.2, Y.4412/F.747.8, and Y.4413/F.748.5, there are 6, 4, and 9 figures and 18, 22, and 24 pages, respectively. Then, we compress and convert these files into a single ZIP file (3.4 MB). Hereafter, we consider how an attacker obtains this file.

Fig. 13 describes a flow to create malicious names with steganography to leak data following the properties summarized in Table 2. To ameliorate the information leakage throughput, as discussed in Section III-B3, an essential key is to *add as many tokens into the name as possible*. Thus, the main principles to create malicious names are

(i) to extract NDN names classified as the inliers by the isolation forest whose proportion of outliers is set to R_O^A (i.e., 0.01, 0.05, 0.1, 0.2, 0.3, or 0.4) as a parameter of the attacker side;
(ii) to select one name with the maximal sum of $N_/$ and $N_=$ among inliers;
(iii) to extract $N_/$, $N_=$, L_{Path}, and L_{Query} from the name as the thresholds to create the malicious names; and
(iv) to create the malicious names with steganography considering the thresholds.

Table 7 indicates the thresholds of $N_/$, $N_=$, L_{Path}, and L_{Query} extracted from each TLD name dataset using the isolation forest. Moreover, to generate the malicious names with steganography, the attacker must prepare a table with each token and its corresponding digits. The table is configured as follows; using "/", "=", and "&" as delimiters, we collect the tokens from the path and from the query in the inliers identified by the isolation forest and assign four hexadecimal digits (i.e., two bytes of information) to each token in the table.

As for token type, our previous paper [8] utilizes the dictionary words from WordNet [26] as tokens. However, in this paper, we use tokens collected from the real NDN names or the NDN names converted from the real URLs used in the current Internet. The number of tokens required is $16^4 = 65,536$, and from the attacker's NDN name dataset shown in Table 4, we extract frequently used tokens in the inliers preferentially. Occasionally, we could find some tokens whose length is large (e.g., percent-encoding [10]), and in terms of the information leakage throughput, smaller length is preferable. Thus, when there are several tokens with the same frequency of appearance in the inliers, we take them in ascending order of length and ignore those with a length greater than the 90th percentile of the length of name components shown in Table 5. Table 8 shows the number of tokens collected from the path and query in the inliers. As for the tokens from the path, in all cases, the number of tokens is 65,536. However, in some cases, the number of tokens from the query does not reach 65,536. In this case, each missing token is covered by the corresponding four hexadecimal digits. As a comparison to the steganography-embedded Interest name (hereafter, we call this *Token*), we prepare malicious names, and instead of the token collected

[2]We use scikit-learn [27] with default values for the remaining parameters.

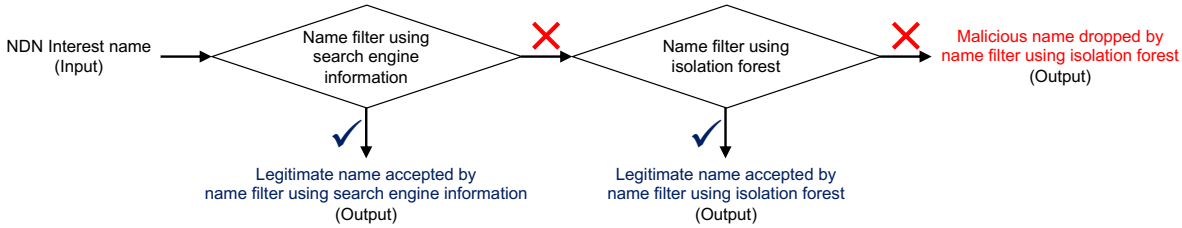


FIGURE 12: Flow to check NDN Interest name using two name filters.

TABLE 7: Thresholds of $N_{/}$, $N_{=}$, L_{Path}, and L_{Query} to create malicious names

TLD	$R_O^A = 0.01$				$R_O^A = 0.05$				$R_O^A = 0.1$				$R_O^A = 0.2$				$R_O^A = 0.3$				$R_O^A = 0.4$			
	$N_{/}$	$N_{=}$	L_{Path}	L_{Query}	$N_{/}$	$N_{=}$	L_{Path}	L_{Query}	$N_{/}$	$N_{=}$	L_{Path}	L_{Query}	$N_{/}$	$N_{=}$	L_{Path}	L_{Query}	$N_{/}$	$N_{=}$	L_{Path}	L_{Query}	$N_{/}$	$N_{=}$	L_{Path}	L_{Query}
com	32	0	260	0	32	0	260	0	32	0	260	0	32	0	260	0	32	0	260	0	32	0	260	0
net	38	0	216	0	38	0	216	0	38	0	216	0	38	0	216	0	38	0	216	0	21	0	222	0
org	1	66	10	1381	1	66	10	1381	3	41	25	726	2	27	24	248	2	27	24	248	2	25	24	231
info	3	24	26	257	3	24	26	257	3	24	26	257	2	25	26	249	17	0	107	0	17	0	107	0
jp	2	29	27	244	2	29	27	244	27	0	214	0	21	0	112	0	21	0	112	0	21	0	112	0
fr	1	61	1	2352	1	61	1	2352	1	61	1	2352	3	26	32	234	18	0	118	0	18	0	118	0
uk	2	32	20	175	2	32	20	175	2	32	20	175	3	22	18	169	3	22	18	169	3	22	18	169
de	3	25	23	237	3	25	23	237	3	25	23	237	24	0	173	0	24	0	173	0	24	0	173	0

TABLE 8: Number of tokens collected from the path and query in inliers

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Path	Query	Path	Query	Path	Query	Path	Query	Path	Query	Path	Query
com	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	61,967
net	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	31,737
org	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	42,258
info	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	48,855	65,536	7,647
jp	65,536	32,159	65,536	22,930	65,536	16,111	65,536	149	65,536	32	65,536	16
fr	65,536	65,536	65,536	65,536	65,536	65,536	65,536	33,788	65,536	89	65,536	8
uk	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	65,536	51,745	65,536	19,711
de	65,536	65,536	65,536	65,536	65,536	65,536	65,536	47,449	65,536	214	65,536	15

from the real NDN names, the corresponding four hexadecimal digits are directly used as the tokens (hereafter, we call this *Hex*).

The tokens from the path and the query are inserted in the path and the query in the malicious names, respectively. In particular, in the query, we place reserved keys such as "key1" before the equals characters (e.g., key1=*token1*&key2=*token2*) to identify the order of tokens. The reason for this convention is as follows. When the malware uses the tokens as the keys and the values in the query (e.g., *token1=token2&token3=token4*), the protector's name filter can shuffle the order of the key-value pairs, which does not have an impact on legitimate content retrieval processes, but it is detrimental to the attacker since they cannot successfully decode the malicious names unless they decrypt the tokens in the query while considering the order of them. Finally, the features of the created malicious names should be similar to those of the name selected in the abovementioned principle (ii).

B. RESULTS

Tables 9, 10, and 11 show the performances of each of the NDN name filters (R_O^P = 0.01, 0.2, and 0.4) against the malicious names to leak data (R_O^A = 0.01, 0.05, 0.1, 0.2, 0.3, and 0.4) in terms of

- the true positive rate, which is defined as the ratio of the number of malicious names identified as malicious to the total number of generated malicious names, and
- the information leakage throughput per Interest, which has unit **bytes/Interest**.

As an example, when a malicious name "ndn://attacker.com/*token1*/*token2*/*token3*?key1=*token4*&key2=*token5*" bypasses the filter successfully, the name can convey 10 bytes of leaked data to the attacker, and **the information leakage throughput can be calculated by the total volume of leaked data obtained from the malicious names successfully bypassing the filter divided by the total number of malicious names created**. In these tables, the averaged value marked in blue boldface indicates maximum true positive rate or minimum information leakage throughput (i.e., the best performance from the protector's point of view), while the averaged value marked in red boldface indicates minimum true positive rate or maximum information leakage throughput (i.e., the best performance from the attacker's point of view). The performances of each of the NDN name filters for R_O^P between 0.01 and 0.4 (i.e. 0.05, 0.1, or 0.3) show an increase in true positive rate and a decrease of information leakage throughput as R_O^P increases. Among them, it is interesting that the true positive rate for *Token* is higher than the one for *Hex* for the malicious names including "fr" as the

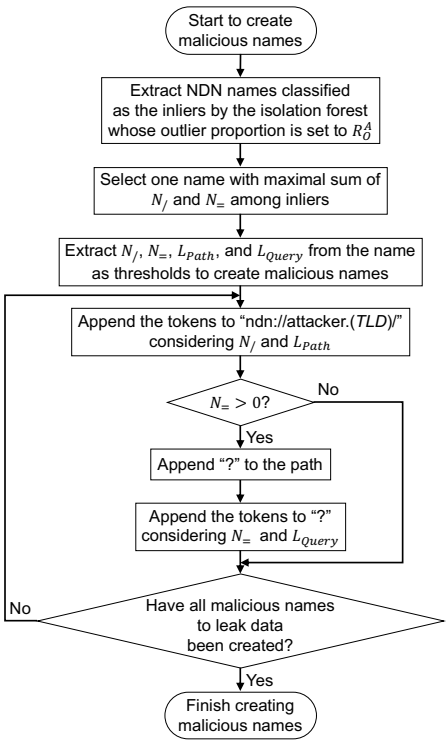


FIGURE 13: Flow to create malicious names.

TABLE 9: Performances of NDN name filter against malicious names (R_O^P = 0.01)

(a) True positive rate

TLD	R_O^A = 0.01		R_O^A = 0.05		R_O^A = 0.1		R_O^A = 0.2		R_O^A = 0.3		R_O^A = 0.4	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
net	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
org	0.593	0.131	0.593	0.133	0.972	0.411	0.890	0.379	0.890	0.383	0.868	0.403
info	1.00	0.863	1.00	0.851	1.00	0.815	0.999	0.773	0.00	0.00	0.00	0.00
jp	0.994	0.880	0.994	0.864	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fr	0.00	0.00	0.00	0.00	0.00	0.00	0.821	0.193	0.00	0.00	0.00	0.00
uk	0.620	0.249	0.620	0.254	0.620	0.247	0.731	0.215	0.731	0.200	0.731	0.208
de	0.446	0.218	0.446	0.213	0.446	0.225	0.00	0.00	0.00	0.00	0.00	0.00
AVG	**0.457**	0.293	**0.457**	0.289	0.380	0.212	0.430	0.195	0.203	**0.0729**	0.200	0.0764

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	R_O^A = 0.01		R_O^A = 0.05		R_O^A = 0.1		R_O^A = 0.2		R_O^A = 0.3		R_O^A = 0.4	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	64.0	51.2	64.0	51.2	64.0	51.3	64.0	50.8	64.0	49.4	64.0	46.0
net	76.0	44.9	76.0	43.7	76.0	43.0	76.0	44.9	76.0	46.7	42.0	41.9
org	54.5	116	54.5	116	2.43	50.8	5.49	26.4	5.49	26.4	6.07	23.8
info	0.00429	5.77	0.00429	6.30	0.00429	7.91	0.0708	9.78	34.0	20.8	34.0	21.2
jp	0.310	5.11	0.310	6.11	54.0	52.3	42.0	28.5	42.0	29.2	42.0	30.3
fr	124	124	124	124	124	124	8.93	21.8	36.0	17.9	36.0	19.0
uk	13.7	19.2	13.7	19.4	13.7	20.1	10.2	21.1	10.2	20.9	10.2	25.4
de	27.7	26.7	27.7	27.6	27.7	26.7	48.0	23.7	48.0	24.8	48.0	26.1
AVG	45.0	49.1	45.0	**49.3**	45.2	47.0	31.8	**28.4**	39.5	29.5	35.3	29.2

TABLE 10: Performances of NDN name filter against malicious names (R_O^P = 0.2)

(a) True positive rate

TLD	R_O^A = 0.01		R_O^A = 0.05		R_O^A = 0.1		R_O^A = 0.2		R_O^A = 0.3		R_O^A = 0.4	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.0154	0.00	0.0154	0.00	0.0154	1.50e-05	0.0154	0.00	0.0154	0.00	0.0154	0.00
net	0.000155	0.00	0.000155	0.00	0.000155	0.00	0.000155	0.00	0.000155	0.00	9.81e-05	1.22e-05
org	1.00	0.928	1.00	0.922	1.00	0.988	1.00	0.989	1.00	0.990	1.00	0.989
info	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.0395	1.82e-05	0.0395	1.86e-05
jp	1.00	1.00	1.00	1.00	0.00585	0.00	0.0104	0.00	0.0104	0.00	0.0104	8.86e-06
fr	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.996	0.00	0.00	0.00	0.00
uk	0.999	0.855	0.999	0.851	0.999	0.844	1.00	0.798	1.00	0.792	1.00	0.825
de	1.00	0.875	1.00	0.870	1.00	0.877	2.80e-05	0.00	2.80e-05	0.00	2.80e-05	0.00
AVG	**0.752**	0.707	**0.752**	0.705	0.628	0.589	0.503	0.473	0.258	**0.223**	0.258	0.227

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	R_O^A = 0.01		R_O^A = 0.05		R_O^A = 0.1		R_O^A = 0.2		R_O^A = 0.3		R_O^A = 0.4	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	63.0	51.2	63.0	51.2	63.0	51.3	63.0	50.8	63.0	49.4	63.0	46.0
net	76.0	44.9	76.0	43.7	76.0	43.0	76.0	44.9	76.0	46.7	42.0	41.9
org	0.00524	9.71	0.00524	10.5	0.00	1.05	0.00	0.477	0.00	0.430	0.00	0.419
info	3.15e-05	7.38e-05	3.15e-05	0.00	3.15e-05	0.00	0.00	0.00	32.7	20.8	32.7	21.2
jp	0.00	0.00	0.00	0.00	53.7	52.3	41.6	28.5	41.6	29.2	41.6	30.3
fr	0.00442	0.00	0.00442	0.00	0.00442	0.00	0.00	0.103	36.0	17.9	36.0	19.0
uk	0.0180	3.62	0.0180	3.77	0.0180	4.04	0.00590	5.27	0.00590	5.25	0.00590	5.48
de	0.00657	4.23	0.00657	4.48	0.00657	4.13	48.0	23.7	48.0	24.8	48.0	26.1
AVG	17.4	**14.2**	17.4	**14.2**	24.1	19.5	28.6	19.2	**37.2**	24.3	32.9	23.8

TLD when R_O^A = 0.01, 0.05, or 0.1 and R_O^P = 0.05 or 0.1. We discuss this exception in Appendix B.

From Tables 9a, 10a, and 11a, for all configurations of R_O^A, every true positive rate for *Token* is less than that for *Hex*, because by using the tokens actually obtained from the attacker's name datasets, the frequencies of letters and other printable characters in the path and query of *Token* become similar to those of the inliers in the protector's name datasets. Thus, a steganography-embedded Interest name can circumvent a protector more efficiently than a naive name such as one that only includes raw data. Moreover, for every configuration of R_O^A, the greater R_O^P is, the greater the true positive rate becomes; this is because by increasing R_O^P, the name filter is built by a name dataset that excludes more outliers, though the compensation for higher true positive rate is to obtain higher false positive rate. With a higher true positive rate, the attacker can create malicious names to bypass the filter by increasing R_O^A.

From Tables 9b, 10b, and 11b, when the true positive rate for *Hex* is the same as that for *Token*, the information leakage throughput using *Hex* is larger than when using *Token*, since *Hex* does not encode the data with steganography, which causes throughput degradation in the case of *Token*. However, as mentioned previously, *Hex* can be detected as anomalous more easily than *Token*, so the information leakage throughput using *Token* becomes larger than that using *Hex* when R_O^P becomes larger. Assuming that the protector adopts the name filter whose R_O^P is 0.4, the average information leakage throughput for all the domains is as high as 22.6 bytes/Interest (see Table 11b, the AVG line with R_O^A = 0.3).

As shown in Table 7, there are three types of malicious name: name composed of only the path, name composed of only the query, and name composed of both. According to Tables 9, 10, and 11, regardless of the type of TLD, the generated malicious names that consist of only the path can bypass the name filters more easily than the other two types of malicious name. This may indicate that, for malware, it is preferable to exploit only the path to create malicious names leaking data and to hide activities.

To verify the hypothesis that the path is useful for malware to circumvent an NDN name filter and send leaked data to an outside attacker, we also create malicious names exploiting only the path (see Fig. 14). Table 12 shows thresholds of $N_/$ and L_{Path} to create the malicious names. Tables 13, 14, and 15 show the performances of each of the NDN name filters (R_O^P = 0.01, 0.2, 0.4) against the malicious names exploiting only the path (R_O^A = 0.01, 0.05, 0.1, 0.2, 0.3, 0.4) in terms of the true positive rate and the information leakage throughput per Interest. As expected, the generated malicious names can bypass the name filters, and therefore

TABLE 11: Performances of NDN name filter against malicious names (R_O^P = 0.4)

(a) True positive rate

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.698	1.49e-05	0.698	2.99e-05	0.698	1.50e-05	0.698	0.00	0.698	0.00	0.698	8.06e-05
net	0.694	2.62e-05	0.694	0.00	0.694	1.26e-05	0.694	5.25e-05	0.694	1.36e-05	0.377	0.000183
org	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
info	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.985	0.0888	0.985	0.0871
jp	1.00	1.00	1.00	1.00	0.999	0.114	0.994	0.144	0.994	0.135	0.994	0.149
fr	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.997	0.0741	0.997	0.0507
uk	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.999	1.00	1.00
de	1.00	1.00	1.00	1.00	1.00	1.00	0.839	0.0107	0.839	0.0115	0.839	0.0092
AVG	**0.924**	0.750	**0.924**	0.750	**0.924**	0.639	0.903	0.519	0.901	0.289	0.861	**0.287**

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	19.3	51.2	19.3	51.2	19.3	51.3	19.3	50.8	19.3	49.4	19.3	46.0
net	23.3	44.9	23.3	43.7	23.3	43.0	23.3	44.9	23.3	46.7	26.2	41.9
org	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
info	3.15e-05	0.00	3.15e-05	0.00	3.15e-05	0.00	0.00	0.00	0.524	18.7	0.524	19.2
jp	0.00	0.00	0.00	0.00	0.0673	46.3	0.246	24.2	0.246	25.0	0.246	25.6
fr	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.110	16.4	0.110	17.9
uk	0.00	0.00487	0.00	0.00456	0.00	0.00252	0.00	0.00473	0.00	0.0229	0.00	0.00216
de	0.00	0.00	0.00	0.00	0.00	0.00	7.75	23.5	7.75	24.6	7.75	26.0
AVG	**5.33**	12.0	**5.33**	11.9	**5.33**	17.6	6.32	17.9	6.40	**22.6**	6.77	22.1

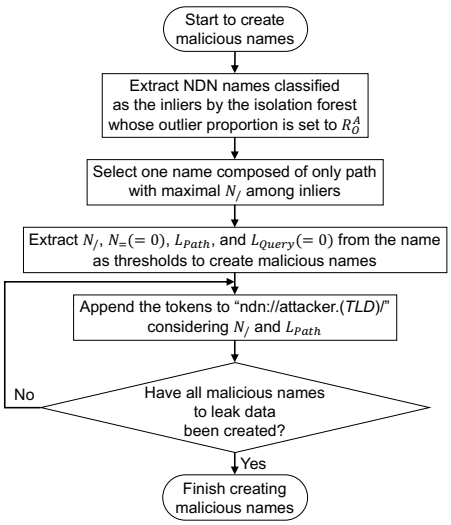


FIGURE 14: Flow to create malicious names exploiting only the path.

the true positive rate decreases and the information leakage throughput increases. Assuming that the protector adopts the name filter whose R_O^P is 0.4, the average information leakage throughput for all the domains is as high as 32.1 bytes/Interest (see Table 15b, the AVG line with $R_O^A = 0.1$).

In terms of true positive rate and information leakage throughput, the above method to create malicious names using the tokens obtained from the NDN names in the real Internet is more efficient to leak data than the method proposed in [8], which uses the dictionary words from WordNet. As for the method in this paper, the frequencies of letters and the other printable characters in the path become similar to those of the inliers in the protector's name datasets. In addition, we select one name with maximal sum of $N_/$ and $N_=$ among inliers, while [8] obtains the longest name, so many more tokens can be added.

When the proposed filter is not used, the attacker can fill the Interest name with the leaked data in hexadecimal digits. Selecting the longest name in the attacker's name datasets, which the attacker can add 2,352 hexadecimal digits to, the information leakage throughput reaches 1.18 Kbytes/Interest, which is maximal since 1 byte of leaked data is mapped into 2 hexadecimal digits. Thus, by using the proposed filter, the malware must send 36.8 times (=1.18 K/32.1) more Interests to the attacker. Moreover, theoretically, in NDN, much longer names are acceptable. Since a Data packet must include the corresponding name in the Interest, the length of the name should be less than the maximum size of a Data packet (8800 bytes by default [29]). Thus, assuming that the maximum length name can consist of 8800 digits (i.e., 4.40 Kbytes/Interest), by using the proposed filter, the malware must send 137 times (=4.40 K/32.1) more Interests to the attacker.

VII. DISCUSSION

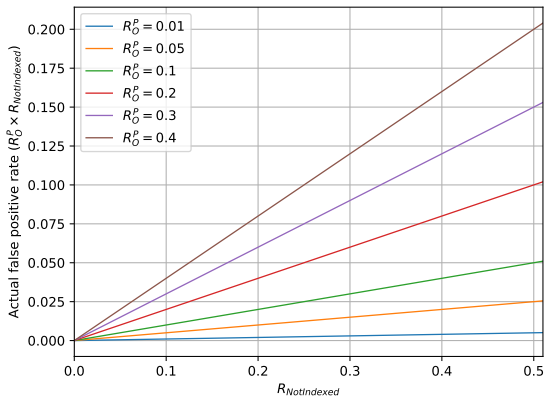


FIGURE 15: Actual false positive rate using two name filters.

The false positive rate R_O^P introduced in Section VI is caused by the name filter using the isolation forest. The actual false positive rate should be computed after checking Interest names with the name filter using the search engine information and the isolation forest. Thus, the actual false positive rate is $R_O^P \times R_{NotIndexed}$, where $R_{NotIndexed}$ is the probability that legitimate users are accessing real Deep Web content (content not indexed by the search engine). Fig. 15 shows that the actual false positive rate for each R_O^P depends on $R_{NotIndexed}$, which will be very small if the enterprise network is properly managed. Therefore, the proposed filters can keep the false positive rate relatively low thanks to the name filer using the search engine information.

The proposed algorithms to create malicious names shown in Figs. 13 and 14 may not be optimal for increasing the information leakage throughput. These algorithms concatenate several tokens using a slash character ("/") in the path and equals and ampersand characters ("=", "&") in the query as delimiters. In the case that L_{Path} or L_{Query} is relatively long, which means that there is large space to append as many tokens as possible to the path or query, by concatenating several tokens using some other delimiters such as "-" and "_" and adding the concatenated tokens to the

TABLE 12: Thresholds of $N_/$ and L_{Path} to create malicious names exploiting only the path

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	$N_/$	L_{Path}	$N_/$	L_{Path}	$N_/$	L_{Path}	$N_/$	L_{Path}	$N_/$	L_{Path}	$N_/$	L_{Path}
com	32	260	32	260	32	260	32	260	32	260	32	260
net	38	216	38	216	38	216	38	216	38	216	21	222
org	21	151	21	151	21	151	21	151	21	151	21	151
info	17	107	17	107	17	107	17	107	17	107	17	107
jp	27	214	27	214	27	214	21	112	21	112	21	112
fr	18	118	18	118	18	118	18	118	18	118	18	118
uk	24	189	24	189	24	189	24	189	24	189	24	189
de	24	173	24	173	24	173	24	173	24	173	24	173

TABLE 13: Performances of NDN name filter against malicious names exploiting only the path (R_O^P = 0.01)

(a) True positive rate

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
net	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
org	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
info	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
jp	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
fr	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
uk	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
de	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
AVG	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	64.0	51.2	64.0	51.2	64.0	51.3	64.0	50.8	64.0	49.4	64.0	46.0
net	76.0	44.9	76.0	43.7	76.0	43.0	76.0	44.9	76.0	46.7	42.0	41.9
org	42.0	30.8	42.0	30.6	42.0	31.4	42.0	32.4	42.0	33.5	42.0	34.5
info	34.0	19.9	34.0	19.3	34.0	18.7	34.0	20.1	34.0	20.8	34.0	21.2
jp	54.0	52.1	54.0	52.2	54.0	52.3	42.0	28.5	42.0	29.2	42.0	30.3
fr	36.0	14.7	36.0	13.8	36.0	13.6	36.0	15.6	36.0	17.9	36.0	19.0
uk	48.0	31.5	48.0	32.3	48.0	33.3	48.0	34.8	48.0	36.2	48.0	38.1
de	48.0	20.7	48.0	21.5	48.0	22.4	48.0	23.7	48.0	24.8	48.0	26.1
AVG	**50.3**	33.2	**50.3**	33.1	**50.3**	33.3	48.8	**31.4**	48.8	32.3	44.5	32.1

TABLE 14: Performances of NDN name filter against malicious names exploiting only the path (R_O^P = 0.2)

(a) True positive rate

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.0154	0.00	0.0154	0.00	0.0154	1.50e-05	0.0154	0.00	0.0154	0.00	0.0154	0.00
net	0.000155	0.00	0.000155	0.00	0.000155	0.00	0.000155	0.00	0.000155	0.00	9.81e-05	1.22e-05
org	0.000540	0.00	0.000540	0.00	0.000540	0.00	0.000540	0.00	0.000540	0.00	0.000540	0.00
info	0.0395	6.98e-05	0.0395	3.95e-05	0.0395	1.64e-05	0.0395	1.17e-05	0.0395	1.82e-05	0.0395	1.86e-05
jp	0.00585	0.00	0.00585	0.00	0.00585	0.00	0.0104	0.00	0.0104	0.00	0.0104	8.86e-06
fr	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
uk	1.40e-05	0.00	1.40e-05	0.00	1.40e-05	0.00	1.40e-05	0.00	1.40e-05	0.00	1.40e-05	1.11e-05
de	2.80e-05	0.00	2.80e-05	0.00	2.80e-05	0.00	2.80e-05	0.00	2.80e-05	0.00	2.80e-05	0.00
AVG	7.69e-03	8.73e-06	7.69e-03	4.94e-06	7.69e-03	3.93e-06	**8.25e-03**	**1.46e-06**	**8.25e-03**	2.28e-06	**8.25e-03**	6.35e-06

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	63.0	51.2	63.0	51.2	63.0	51.3	63.0	50.8	63.0	49.4	63.0	46.0
net	76.0	44.9	76.0	43.7	76.0	43.0	76.0	44.9	76.0	46.7	42.0	41.9
org	42.0	30.8	42.0	30.6	42.0	31.4	42.0	32.4	42.0	33.5	42.0	34.5
info	32.7	19.9	32.7	19.3	32.7	18.7	32.7	20.1	32.7	20.8	32.7	21.2
jp	53.7	52.1	53.7	52.2	53.7	52.3	41.6	28.5	41.6	29.2	41.6	30.3
fr	36.0	14.7	36.0	13.8	36.0	13.6	36.0	15.6	36.0	17.9	36.0	19.0
uk	48.0	31.5	48.0	32.3	48.0	33.3	48.0	34.8	48.0	36.2	48.0	38.1
de	48.0	20.7	48.0	21.5	48.0	22.4	48.0	23.7	48.0	24.8	48.0	26.1
AVG	**49.9**	33.2	**49.9**	33.1	**49.9**	33.3	48.4	**31.4**	48.4	32.3	44.2	32.1

malicious names (e.g., /apple_banana), the malware could improve the information leakage throughput. However, as introduced in Section V-B, in terms of SEO, "-" and "_" should be used to separate words, and therefore the concatenated words should become a meaningful sentence following the grammar of the relevant language (e.g., in https://moz.com/blog/15-seo-best-practices-for-structuring-urls, 15-seo-best-practices-for-structuring-urls should be "15 seo best practices for structuring urls" so it follows English grammar). The sequence of tokens depends on the data to

TABLE 15: Performances of NDN name filter against malicious names exploiting only the path (R_O^P = 0.4)

(a) True positive rate

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.698	1.49e-05	0.698	2.99e-05	0.698	1.50e-05	0.698	0.00	0.698	0.00	0.698	8.06e-05
net	0.694	2.62e-05	0.694	0.00	0.694	1.26e-05	0.694	5.25e-05	0.694	1.36e-05	0.377	0.000183
org	0.160	9.90e-05	0.160	8.04e-05	0.160	9.16e-05	0.160	9.47e-05	0.160	5.87e-05	0.160	7.05e-05
info	0.985	0.170	0.985	0.122	0.985	0.0765	0.985	0.0775	0.985	0.0888	0.985	0.0871
jp	0.999	0.108	0.999	0.112	0.999	0.114	0.994	0.144	0.994	0.135	0.994	0.149
fr	0.997	0.107	0.997	0.0748	0.997	0.0641	0.997	0.0729	0.997	0.0741	0.997	0.0507
uk	0.351	0.000147	0.351	0.000208	0.351	7.79e-05	0.351	0.000101	0.351	0.000127	0.351	0.000167
de	0.839	0.00896	0.839	0.00891	0.839	0.00904	0.839	0.0107	0.839	0.0115	0.839	0.00920
AVG	**0.715**	0.0493	**0.715**	0.0398	**0.715**	**0.0330**	**0.715**	0.0382	**0.715**	0.0387	0.675	0.0371

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	$R_O^A = 0.01$		$R_O^A = 0.05$		$R_O^A = 0.1$		$R_O^A = 0.2$		$R_O^A = 0.3$		$R_O^A = 0.4$	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	19.3	51.2	19.3	51.2	19.3	51.3	19.3	50.8	19.3	49.4	19.3	46.0
net	23.3	44.9	23.3	43.7	23.3	43.0	23.3	44.9	23.3	46.7	26.2	41.9
org	35.3	30.8	35.3	30.6	35.3	31.4	35.3	32.4	35.3	33.5	35.3	34.5
info	0.524	16.3	0.524	16.8	0.524	17.1	0.524	18.3	0.524	18.7	0.524	19.2
jp	0.0673	46.4	0.0673	46.3	0.0673	46.3	0.246	24.2	0.246	25.0	0.246	25.6
fr	0.110	12.9	0.110	12.6	0.110	12.5	0.110	14.3	0.110	16.4	0.110	17.9
uk	31.1	31.5	31.1	32.3	31.1	33.3	31.1	34.8	31.1	36.2	31.1	38.1
de	7.75	20.6	7.75	21.4	7.75	22.2	7.75	23.5	7.75	24.6	7.75	26.0
AVG	**14.7**	31.8	**14.7**	31.9	**14.7**	**32.1**	**14.7**	30.4	**14.7**	31.3	15.1	31.2

be leaked, and therefore, if the malware simply connects the tokens with such delimiters, the connected tokens would neither be meaningful nor follow any language's grammar. The generated awkward name could be detected as anomalous by some natural language processing (NLP) techniques [30]. Thus, it is extremely difficult for the malware to simply concatenate tokens using some delimiters, which means information leakage throughput is decreased.

In the Internet, in order to detect malicious URLs that induce infection by malware (i.e., drive-by download attack) or access phishing/spam webpages, some countermeasures rely on token-based filtering based on the fact that specific tokens are used in the malicious URLs ([31]–[33]). However, this type of token-based filtering is not efficient to mitigate the information leakage attack through an Interest. For example, phishing URLs must attract users using attractive tokens, such as "paypal" and "ebay", and make them click the URLs. On the other hand, since the purpose of generating the name to perform the information leakage attack is not cheating users but leaking data, the name need not include such tokens. Thus, the malware can generate the name to leak data without any restrictions and the protector cannot filter out the name using tokens.

In general, in the Internet, a host generating the URL (e.g., an FQDN) follows its own naming policy to effectively identify each Web page from the others using a small set of

keywords. This may reduce the number of tokens following the name prefix. If one name prefix creates many names by appending many different types of tokens, that name prefix and the generated names may be detected as anomalous. Thus, to protect from an information leakage attack, we have to know the statistical distribution of the number of possible tokens appearing after each slash character ("/") based on the many Web pages in the Internet. Once the name prefix is blacklisted, the malware and the attacker must exploit the other name prefixes that are shared between them.

The features used for creating the name filter (Table 3) are very simple as they do not reflect any character ordering. An extension to use n-gram in the features could probably be more effective to detect *Hex* as anomalous. However, as for *Token*, the character ordering is derived from the extracted real tokens, so the above extension might be ineffective to judge *Token* as anomalous.

As the above discussions, the further improvement of the proposed name filter taking semantic and correlation of tokens or characters into account, needs reasonable modelling and assumptions from several other aspects, such as modelling the naming policy of malicious name for the information leakage by non-existing potential attackers. This paper, therefore, treats this kind of extensions as our future work.

As a precondition, to utilize the proposed NDN name filter, a policy to allow an enterprise firewall to inspect traffic from the employees carefully is required. For example, it is possible for a consumer to obfuscate the Interest name in some cases such as by avoiding censorship ([34]–[36]). Similar to dropping *Hex*, the proposed name filter would also drop such an obfuscated name even if the name is legitimate and is not exploited to perform the information leakage attack. Moreover, as for an enterprise, in terms of risk hedge, the enterprise should manage any activities of its employees in order to not only detect the malicious Interest names created by malware but also avoid incidents such as malware infection caused by, for example, a drive-by download attack and employee's leaking information to the outside intentionally.

The proposed name filters can drastically choke information leakage throughput per Interest. However, to counteract the filters, malware can send numerous Interests within a short period of time to ameliorate the speed of an attack. Moreover, the malware can exploit an Interest with an explicit payload in the name (similarly to an HTTP POST message in the Internet), which is out of scope of the name filters, and can increase information leakage throughput by adopting a longer payload. These malicious activities can be detected by a filter considering an NDN flow, which we propose as one of our future works.

VIII. RELATED WORK

An information leakage attack through an Interest in NDN imitates one through *DNS tunneling* in the Internet. By exploiting domain names in DNS queries and the corresponding DNS responses, an outside attacker and the malware inside an enterprise network bypass a firewall and perform tunneling of data and commands, and this threat is called DNS tunneling. Leijenhorst et al. [37] show that DNS tunneling can achieve up to 110 KB/s in throughput with DNScat [38], which is as DNS tunneling application, but it adds huge traffic overhead. Merlo et al. [39] compare several DNS tunneling tools in terms of throughput, RTT, and overhead. In order to detect DNS tunneling, some countermeasures have been proposed. Born et al. [40] and Qi et al. [41] analyze character frequencies of domain names and detect DNS tunneling. Alternatively, Farnham [42] investigates not only such a payload analysis, but also traffic analysis such as analyzing count and frequency of queries. Ellens et al. [43] also perform traffic analysis using a flow defined in RFC 3917 [44], and they report that examples of appropriate metrics to detect the tunneling are bytes per flow or the number of flows over time. Kara et al. [45] focus on DNS TXT record and detect DNS tunneling activities. Aiello et al. [46] analyze simple statistical properties, such as inter-arrival time of packets and packet size, and apply them to machine learning techniques. These countermeasures, however, are only effective to detect attacks generated by some specific tools such as DNScat or some malware such as Morto worm [47], which means that the countermeasures cannot necessarily eliminate the threat of the attack. Xu et al. [48] conclude that DNS-based botnet C&C channel, which is based on DNS tunneling, is "feasible, powerful, and difficult to detect and block". Thus, we argue that an information leakage attack through an Interest in NDN should be one of the essential security threats in the protocol level such as interest flooding attack [49] and content poisoning attack [50].

As for an Interest name, there are three types: a human-readable name, a non-human-readable name, and a combination of the two. NDN team encourages the use of human-readable clear-text strings [11]. On the other hand, in order not to use a public key for data authentication, Baugher et al. [51] propose self-verifying names, which utilize a hash value (i.e., non-human readable) in the names obtained from Catalog. Ghali et al. [52] apply a cryptographic hash function to (usually human-readable) application-layer names in order to translate them into network-layer names, which provides some benefits for FIB, PIT, and CS. Moreover, in order to realize some cases such as evading censorship [34]–[36], an encryption method must be applied to Interest names. However, these research works do not discuss information leakage attacks through an Interest name in NDN at all.

IX. CONCLUSION

We presented information leakage attacks through a Data and through an Interest in NDN. Particularly, we investigated such attacks through an Interest and showed two possible methods to perform the attack. Moreover, we introduced a steganography-embedded Interest name to leak information as a more sophisticated attack to exploit an Interest name than simply adding leaked information into a name. Then,

as a countermeasure against the attack, we proposed a name filter using search engine information and using an isolation forest to classify a name in the Interest as legitimate or not. In order to build the filter using an isolation forest, we collected URLs from a data repository provided by Common Crawl. Our experiments show that (1) the path part in the URL-based NDN name is useful for malware to create malicious names leaking information and hide the activities; (2) it is difficult to completely prevent information leakage even in NDN; and (3) the filters can choke information leakage throughput dramatically and malware needs to send 137 times more Interest packets to leak information than when filters are not used. By using our filters, we are able to mitigate the speed of an information leakage attack by two orders of magnitude, which makes our filters very efficient to prevent information leakage in NDN.

Proposing the name filter is a first step to prevent the information leakage attack, and in order to address malware's possible counter-attacks against the name filter such as sending numerous Interests within a short period of time to ameliorate the speed of an attack, the next step is to introduce a flow filter.

APPENDIX A

Figs. 16, 17, 18, 19, and 20 show the cumulative distribution function (CDF) of attributes 1 to 5 for each TLD. After summing each CDF for eight TLDs, we divide the sum by eight to obtain the average CDF (i.e., "overall" in the legend). In all figures, the CDF of "overall" in the protector's name dataset is similar to that in the attacker's name dataset, while the CDF of each of the TLDs in the protector's name dataset is slightly different from that in the attacker's name dataset.

APPENDIX B

Table 16 shows the performances of the NDN name filters (R_O^P = 0.1) against the malicious names to leak data (R_O^A = 0.01, 0.05, 0.1, 0.2, 0.3, 0.4) in terms of the true positive rate and the information leakage throughput per Interest. The discussion is much the same as in Section VI-B with the exception of the case of creating malicious names including "fr" as the TLD. For these names, when R_O^A is set to 0.01, 0.05, or 0.1, the true positive rate for *Token* is higher than that for *Hex*. This is because the created malicious names consist of only the query (see Table 7). Figs. 9 and 11 show that the average frequency of the percentage character ("%") in the query is larger than that in the path. Thus, percent-encoding may be used more in the query than in the path. In such a case, the true positive rate for *Hex* may be lower than that for *Token*.

TABLE 16: Performances of NDN name filter against malicious names (R_O^P = 0.1)

(a) True positive rate

TLD	R_O^A = 0.01		R_O^A = 0.05		R_O^A = 0.1		R_O^A = 0.2		R_O^A = 0.3		R_O^A = 0.4	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
net	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
org	0.999	0.623	0.999	0.619	1.00	0.898	1.00	0.901	1.00	0.907	1.00	0.911
info	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00
jp	1.00	1.00	1.00	1.00	0.00	0.00	2.45e-05	0.00	2.45e-05	0.00	2.45e-05	0.00
fr	0.178	0.399	0.178	0.343	0.178	0.687	0.999	0.789	0.00	0.00	0.00	0.00
uk	0.988	0.706	0.988	0.703	0.988	0.693	0.997	0.642	0.997	0.627	0.997	0.656
de	0.990	0.752	0.990	0.743	0.990	0.752	0.00	0.00	0.00	0.00	0.00	0.00
AVG	**0.644**	0.560	**0.644**	0.551	0.520	0.504	0.500	0.417	0.250	**0.192**	0.250	0.196

(b) Information leakage throughput per Interest [bytes/Interest]

TLD	R_O^A = 0.01		R_O^A = 0.05		R_O^A = 0.1		R_O^A = 0.2		R_O^A = 0.3		R_O^A = 0.4	
	Hex	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*	*Hex*	*Token*
com	64.0	51.2	64.0	51.2	64.0	51.3	64.0	50.8	64.0	49.4	64.0	46.0
net	76.0	44.9	76.0	43.7	76.0	43.0	76.0	44.9	76.0	46.7	42.0	41.9
org	0.178	50.6	0.178	51.1	0.00	8.81	0.00438	4.19	0.00438	3.96	0.00618	3.54
info	3.15e-05	0.00219	3.15e-05	0.00354	3.15e-05	0.00718	0.00	0.0146	34.0	20.8	34.0	21.2
jp	0.00	0.00	0.00	0.00	54.0	52.3	42.0	28.5	42.0	29.2	42.0	30.3
fr	102	74.5	102	81.4	102	38.9	0.038	5.69	36.0	17.9	36.0	19.0
uk	0.425	7.43	0.425	7.60	0.425	8.07	0.128	9.41	0.128	9.57	0.128	10.9
de	0.477	8.38	0.477	8.87	0.477	8.37	48.0	23.7	48.0	24.8	48.0	26.1
AVG	30.4	29.6	30.4	30.5	37.1	26.3	28.8	**20.9**	**37.5**	25.3	33.3	24.9

REFERENCES

[1] IT Security Risks Survey 2014, https://media.kaspersky.com/en/IT_Security_Risks_Survey_2014_Global_report.pdf

[2] Understanding Targeted Attacks: The Impact of Targeted Attacks, https://www.trendmicro.com/vinfo/us/security/news/cyber-attacks/the-impact-of-targeted-attacks

[3] R. Beuran, C. Pham, D. Tang, K. Chinen, Y. Tan, and Y. Shinoda, "CyTrONE: An Integrated Cybersecurity Training Framework," in ICISSP 2017.

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," IEEE Communications Magazine, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[5] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," ACM SIGCOMM CCR, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[6] F. T. Liu, K. M. Ting, and Z. H. Zhou, "Isolation Forest," in IEEE ICDM 2008.

[7] Common Crawl, http://commoncrawl.org/

[8] D. Kondo, T. Silverston, H. Tode, T. Asami, and O. Perrin, "Risk Analysis of Information-Leakage through Interest Packets in NDN," in IEEE INFOCOM WORKSHOP (NOM) 2017.

[9] NDN Packet Format Specification (Name), https://named-data.net/doc/NDN-packet-spec/current/name.html

[10] RFC 3986, https://tools.ietf.org/html/rfc3986

[11] NDN Technical Memo: Naming Conventions, https://named-data.net/wp-content/uploads/2014/08/ndn-tr-22-ndn-memo-naming-conventions.pdf

[12] RFC 1808, https://tools.ietf.org/html/rfc1808

[13] We knew the web was big... (Google Official Blog), https://googleblog.blogspot.jp/2008/07/we-knew-web-was-big.html

[14] U. Schnurrenberger, "Comparing Apples to Apples in ICN," in IEEE CCNC WORKSHOP (FI4D) 2017.

[15] D. Goergen, T. Cholez, J. Francois, and T. Engel, "A Semantic Firewall for Content-Centric Networking," in IFIP/IEEE IM Mini-Conference 2013.

[16] NDN Packet Format Specification 0.3 documentation, https://named-data.net/doc/NDN-packet-spec/current/

[17] M. Luby, "LT Codes," in IEEE FOCS 2002.

[18] A. Shokrollahi, "Raptor Codes," IEEE TIT, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.

[19] J. Mason, S. Small, F. Monrose, and G. MacManus, "English Shellcode," in CCS 2009.

[20] URL Fragments and Redirects, https://blogs.msdn.microsoft.com/ieinternals/2011/05/16/url-fragments-and-redirects/

[21] B. Hawkins, "Under The Ocean of the Internet - The Deep Web," https://www.sans.org/reading-room/whitepapers/covert/ocean-internet-deep-web-37012

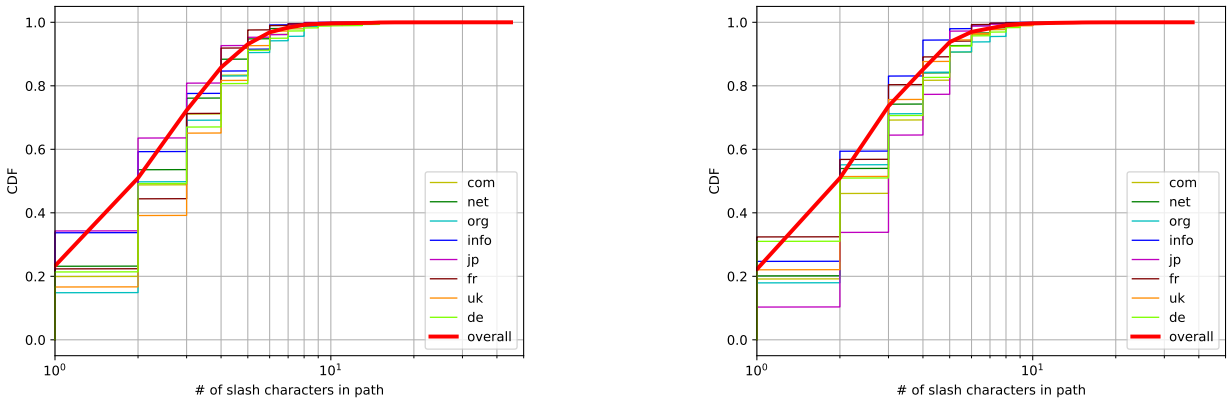
[22] Google's search business might not be as water-tight as people think it is, https://www.businessinsider.com.au/the-number-of-people-using-search-engines-is-in-decline-2015-9

[23] Search Engine Optimization (SEO) Starter Guide, https://support.google.com/webmasters/answer/7451184

[24] 15 SEO Best Practices for Structuring URLs, https://moz.com/blog/15-seo-best-practices-for-structuring-urls

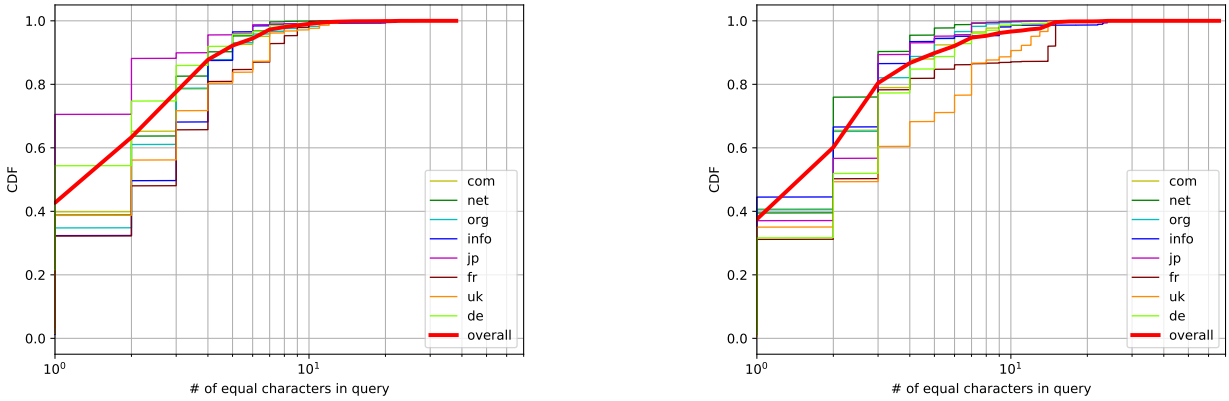
[25] RFC 1738, https://tools.ietf.org/html/rfc1738

[26] G. A. Miller, "WordNet: A Lexical Database for English," Commun. ACM, vol. 38, no. 11, pp. 39–41, Nov. 1995.



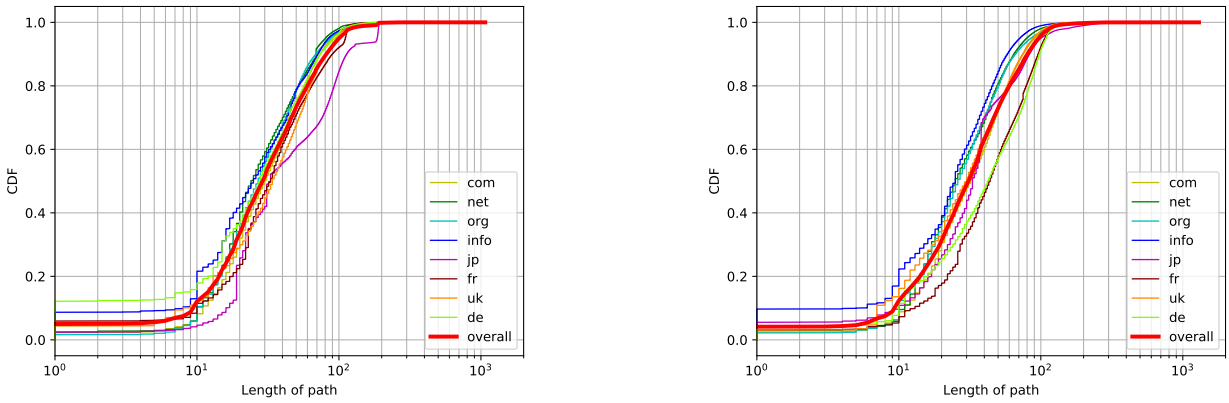
(a) Protector's name dataset. **(b)** Attacker's name dataset.

FIGURE 16: CDF of **Number of slash characters in the path**.



(a) Protector's name dataset. **(b)** Attacker's name dataset.

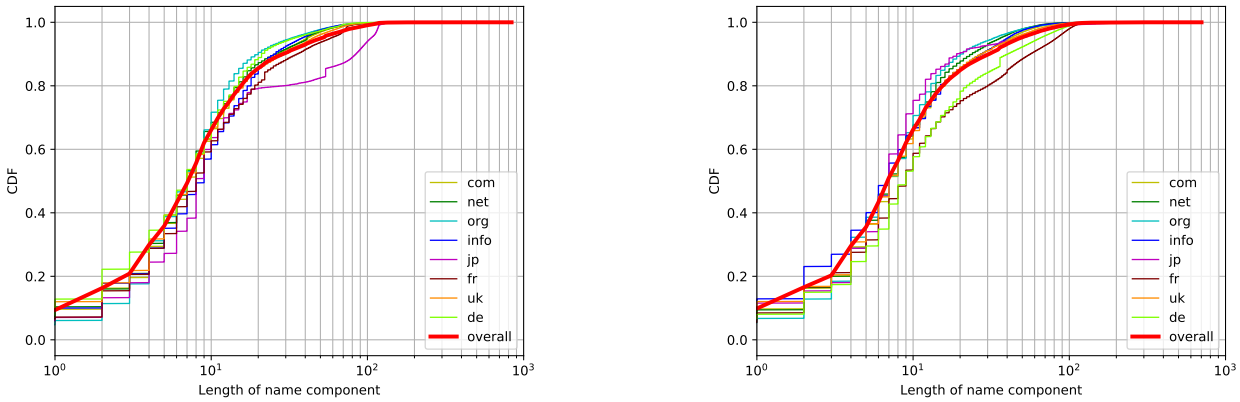
FIGURE 17: CDF of **Number of equals characters in the query**.



(a) Protector's name dataset. **(b)** Attacker's name dataset.

FIGURE 18: CDF of **the length of the path**.

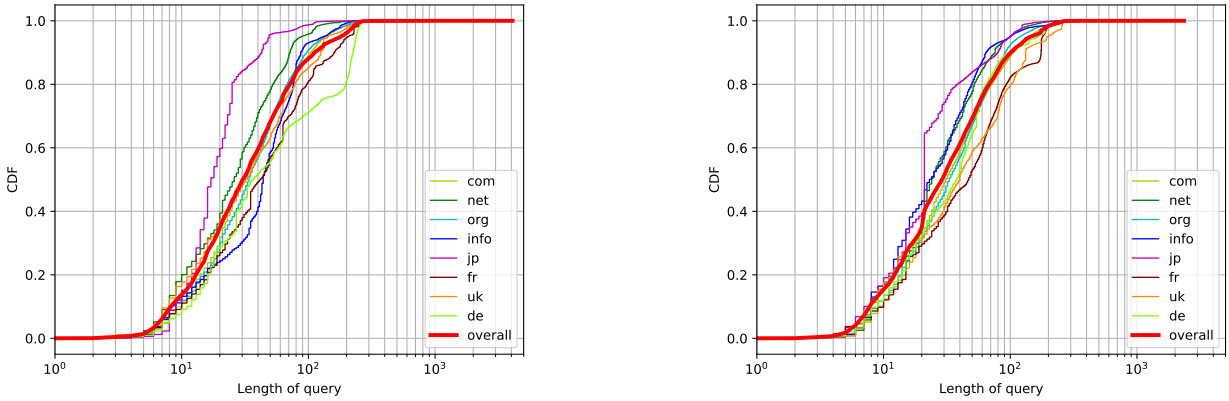
[27] F. Pedregosa, et al. "Scikit-learn: Machine Learning in Python," Machine Learning Research, vol. 12, pp. 2825–2830, 2011.



(a) Protector's name dataset.

(b) Attacker's name dataset.

FIGURE 19: CDF of **the length of the name component**.



(a) Protector's name dataset.

(b) Attacker's name dataset.

FIGURE 20: CDF of **the length of the query**.

[28] ITU-T, https://www.itu.int/en/ITU-T/publications/Pages/latest.aspx

[29] ndn-cxx (tlv.hpp), https://github.com/named-data/ndn-cxx/blob/master/src/encoding/tlv.hpp

[30] C. D. Manning and H. Schütze, "Foundations of Statistical Natural Language Processing," The MIT Press, 1999.

[31] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," in ACM SIGKDD 2009.

[32] A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical Feature Based Phishing URL Detection Using Online Learning," in AISec 2010.

[33] M. Khonji, Y. Iraqi, and A. Jones, "Lexical URL Analysis for Discriminating Phishing and Legitimate Websites," in CEAS 2011.

[34] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, "ANDaNA: Anonymous Named Data Networking Application," in NDSS 2012.

[35] R. Tourani, S. Misra, J. Kliewer, S. Ortegel, and T. Mick, "Catch Me If You Can: A Practical Framework to Evade Censorship in Information-Centric Networks," in ACM ICN 2015.

[36] J. Kurihara, K. Yokota, and A. Tagami, "A Consumer-Driven Access Control Approach to Censorship Circumvention in Content-Centric Networking," in ACM ICN 2016.

[37] T. V. Leijenhorst, K. Chin, and D. Lowe, "On the Viability and Performance of DNS Tunneling," in ICITA 2008.

[38] DNScat, http://tadek.pietraszek.org/projects/DNScat/

[39] A. Merlo, G. Papaleo, S. Veneziano, and M. Aiello, "A Comparative Performance Evaluation of DNS Tunneling Tools," in CISIS 2011.

[40] K. Born, and D. Gustafson, "Detecting DNS Tunnels Using Character Frequency Analysis," https://arxiv.org/pdf/1004.4358v1.pdf

[41] C. Qi, X. Chen, C. Xu, J. Shi, and P. Liu, "A Bigram Based Real Time DNS Tunnel Detection Approach," in ITQM 2013.

[42] G. Farnham, "Detecting DNS Tunneling," https://www.sans.org/reading-room/whitepapers/dns/detecting-dns-tunneling-34152

[43] W. Ellens, P. Żuraniewski, A. Sperotto, H. Schotanus, M. Mandjes, and E. Meeuwissen, "Flow-Based Detection of DNS Tunnels," in AIMS 2013.

[44] RFC 3917, https://tools.ietf.org/html/rfc3917

[45] A. M. Kara, H. Binsalleeh, M. Mannan, A. Youssef, and M. Debbabi, "Detection of Malicious Payload Distribution Channels in DNS," in IEEE ICC 2014

[46] M. Aiello, M. Mongelli, and G. Papaleo, "DNS Tunneling Detection through Statistical Fingerprints of Protocol Messages and Machine Learning," Int. J. Commun. Syst., vol. 28, no. 14, pp. 1987–2002, Jul. 2014.

[47] Morto Worm Sets a (DNS) Record, https://www.symantec.com/connect/blogs/morto-worm-sets-dns-record

[48] K. Xu, P. Butler, S. Saha, and D. Yao, "DNS for Massive-Scale Command and Control," IEEE TDSC, vol. 10, no. 3, pp. 143–153, 2013.

[49] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest Flooding Attack and Countermeasures in Named Data Networking," in IFIP Networking 2013.

[50] C. Ghali, G. Tsudik, and E. Uzun, "Needle in a Haystack: Mitigating Content Poisoning in Named-Data Networking," in SENT 2014.

[51] M. Baugher, B. Davie, A. Narayanan, and D. Oran, "Self-Verifying Names for Read-Only Named Data," in IEEE INFOCOM WORKSHOP (NOMEN) 2012.
[52] C. Ghali, G. Tsudik, and C. A. Wood, "Network Names in Content-Centric Networking," in ACM ICN 2016.

DAISHI KONDO received his B.S. degree in engineering from Osaka University, Osaka, Japan, in 2013 and his M.A.S degree in interdisciplinary information studies from the University of Tokyo, Tokyo, Japan, in 2015. He is currently pursuing a Ph.D. degree in computer science at the University of Lorraine, LORIA (CNRS UMR 7503), Inria Nancy - Grand Est, Nancy, France.

His research interests include Information Centric Networking (ICN), network security, privacy, and Peer-to-Peer (P2P) networking.

THOMAS SILVERSTON is an associate professor at Shibaura Institute of Technology (SIT), Japan. He received his B.Sc., M.Sc. and Ph.D. degrees in Computer Science from University Pierre and Marie Curie - Paris 6, France (currently known as Sorbonne University). Afterwards, Thomas was a post-doctoral researcher at the University of Tokyo (JSPS fellow) and became an associate professor at the University of Lorraine and LORIA/CNRS, Nancy, France, in 2011. He then held several positions in Japan as an invited researcher at the University of Tokyo (2014) and a researcher at NICT, the National Institute of Information and Communications Technology (2016), before joining SIT Research Lab at SIT in 2018. His research focuses on Future Internet and new technologies for future communication networks: Information-centric Networking, Multimedia, Cybersecurity, Internet of Things and Peer-to-Peer.

VASSILIS VASSILIADES obtained a B.Sc. in Computer Science from the University of Cyprus in 2007, a M.Sc. in Intelligent Systems Engineering from the University of Birmingham, U.K. in 2008, and a Ph.D. in Computer Science from the University of Cyprus in 2015. He worked as a post-doctoral researcher and research engineer at Inria Nancy - Grand Est (France) between 2015 and 2018. Currently, he is an associate research fellow at the University of Cyprus. His research interests lie in the area of artificial intelligence and focus on reinforcement learning, neural networks and evolutionary computation.

HIDEKI TODE holds his B. E., M. E., and Ph. D. degrees in communications engineering from Osaka University since 1988, 1990, and 1997, respectively. He was appointed an assistant professor with the Department of Communications Engineering, Osaka University on December 1, 1991 while carrying out his doctoral research. In 1998 and 1999, he was transferred and promoted to a lecturer and associate professor with the Department of Information Systems Engineering, respectively, and in 2002, he moved to the Department of Information Networking, Graduate School of Information Science and Technology, Osaka University. He has been a professor with the Department of Computer Science and Intelligent Systems, Graduate School of Engineering, Osaka Prefecture University since 2008. His current research interests include architectures and controls for optical networks, wireless multi-hop networks, the future Internet, and contents distribution networks. Dr. Tode is a member of the IEEE and a senior member of IEICE Japan.

TOHRU ASAMI received his B.E. degree and M.E. degree in electrical engineering from Kyoto University in 1974 and 1976 respectively, and Ph.D. from the University of Tokyo in 2005. In 1976, he joined a telco, KDD (KDDI). Since that time, he has been working in several research and development areas such as expert systems for telecom network, directory service systems for early academic networks, ADSL field experiment for Internet services, etc. He was C.E.O. of KDDI R&D Labs. Inc. from 2001 to 2005. In 2006, he moved to the University of Tokyo as a professor of Dept. of Information and Communication Engineering, Graduate School of Information Science and Technology. Since 2017, he has been President of Advanced Telecommunications Research Institute International, Japan. He is a member of the IEEE and IEICE (The Institute of Electronics, Information and Communication Engineers, Japan). From 2003 to 2005, he was a vice chairman of the board of directors of Information System Society in IEICE (IEICE-ISS).