The Named Data Networking Flow Filter: Towards Improved Security over Information Leakage Attacks

Daishi Kondo[a,*], Vassilis Vassiliades[b], Thomas Silverston[c], Hideki Tode[a], Tohru Asami[d]

[a]*Osaka Prefecture University, 1-1 Gakuen-cho, Naka-ku, Sakai City, Osaka 599-8531, Japan*
[b]*Research Centre on Interactive Media, Smart Systems and Emerging Technologies, Dimarcheio Lefkosias, Plateia Eleftherias, Nicosia 1500, Cyprus*
[c]*Shibaura Institute of Technology, 307 Fukasaku, Minuma-ku, Saitama City, Saitama 337-8570, Japan*
[d]*Advanced Telecommunications Research Institute International, 2-2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0288, Japan*

Abstract

Named Data Networking (NDN) has the potential to create a more secure future Internet. It is therefore crucial to investigate its vulnerabilities in order to make it safer against information leakage attacks. In NDN, malware inside an enterprise can encode confidential information into Interest names and send it to the attacker. One of the countermeasures is to inspect a name in the Interest using a name filter and identify it as legitimate or anomalous. Although the name filter can dramatically decrease the information leakage throughput per Interest, it has a serious disadvantage: it does not consider a *flow* of Interests. This means that the malware can not only cause information leakage, but even improve the speed of the attack by aggressively producing massive flows of malicious Interests. This paper investigates such NDN flow attacks. Our contribution is twofold. First, we present a scheme that converts an HTTP flow into the corresponding NDN flow, as to date there is no publicly available dataset of the latter. Second, we propose an NDN flow filter based on support vector machines to classify the *short-term* activity of NDN consumers as legitimate or anomalous. In order to obtain legitimate and anomalous flows, we use a preprocessing anomaly detection step where we mark consumers based on their *long-term* activity. Our results clearly show that the flow filter improves the performance of the name filter by two orders of magnitude. Thus, we expect that our approach will drastically reduce the impact of this security attack in NDN.

Keywords: Firewall, Flow filter, Information leakage attack, Named data networking.

1. Introduction

We live in an information-centric world. Information has more value than ever before, and companies, as well as governments, rely on it to make better strategic decisions that increase their success, profitability and the quality of life. It is, therefore, no wonder that information is the target of corporate or government espionage, which includes various security attacks [1].

Information leakage through a targeted attack is one of the most serious security threats against a company [2]. A typical scenario for leaking information out of a company's network is the following. First, the attacker sends to an employee of the targeted company an email with attached malware that looks like a legitimate company email. By opening the malicious email, the employee infects his/her computer with the malware. This creates a communication channel with the attacker which controls the malware and ultimately steals confidential information. As this security threat results from accessing suspicious media, one of the countermeasures would be to provide employees with cybersecurity education [3]. However, it is impossible to completely eliminate all human errors.

Named Data Networking (NDN) [4], proposed as one of the implementations of Information Centric Networking (ICN) [5], adopts a security-by-design approach to construct a more secure network architecture than the current Internet. It is designed so as to allow users to focus on the content (named data), rather than having to reference numerically addressed hosts (as in the Internet Protocol (IP)), in order to retrieve the data. Thus, a packet in NDN can be either an *Interest* (i.e., a request) or *Data* (i.e., a response).

NDN eliminates conventional Internet security threats, such as spoofing and tampering by signature attached in the NDN packet. However, it is not completely immune to an information leakage attack through an Interest [6, 7, 8], and by hacking the Interest packets, an attacker can steal confidential information. Therefore, it is crucial to investigate several countermeasures against such an attack in order to make NDN safer before deploying it on a large

*Corresponding author
Email addresses: daishi.kondo@cs.osakafu-u.ac.jp (Daishi Kondo), v.vassiliades@rise.org.cy (Vassilis Vassiliades), thomas@shibaura-it.ac.jp (Thomas Silverston), tode@cs.osakafu-u.ac.jp (Hideki Tode), asami@atr.jp (Tohru Asami)

scale.

A first step in preventing the attack was the introduction of a *name filter* [6, 7, 8], which was derived from content names based on URLs collected by a web crawler. The name filter inspects a name in the Interest, which is similar to an HTTP GET request message on the Internet, and identifies the name as legitimate or not. Indeed, the name filter can drastically choke the information leakage throughput per Interest. However, there are two possible counter-attacks against the filter that the malware can use aggressively in massive flows to increase the speed of information leakage.

One counter-attack would be to circumvent the filter by sending numerous Interests within a short period of time, each having a low throughput. Another counter-attack would be to exploit Interest packets with an explicit, long payload in the name (like an HTTP POST request message on the Internet). This cannot be effectively inspected by the name filter since the payload creation is unrestricted and depends on the consumer. Therefore, in this work we focus on the problem of how to deal with NDN flow attacks, i.e., malware that produce massive flows of such malicious Interests.

Our contributions are the following: **(i) a scheme to convert an HTTP flow into the NDN flow**, and **(ii) an NDN flow filter to classify a flow as legitimate or anomalous**. In NDN research, an NDN flow has not yet been standardized [9], and moreover, it is quite difficult to obtain a large NDN flow dataset by using a current NDN testbed since now the testbed environment is quite small and does not reflect traffic patterns of real users properly. Therefore, we first introduce the concept of an NDN flow and show our proposed conversion scheme from an HTTP flow into the NDN flow. We then generate the corresponding NDN flow dataset derived from the Waikato HTTP flow dataset [10].

To build the NDN flow filter against the information leakage attack, we investigate properties of malware generating malicious NDN flows that cause information leakage. Since neither legitimate nor malicious NDN flow datasets are available, we assume that the malicious NDN flows tend to be categorized as outliers to achieve higher information leakage throughput. Under this assumption, we classify the generated NDN flows into inliers (legitimate) or outliers (anomalous) from the *long-term* activity of NDN consumers using an isolation forest [11]. Using the inliers and outliers, we build an NDN flow filter based on a support vector machine (SVM) [12] to classify whether a *short-term* activity, not the *long-term* one, is legitimate or anomalous.

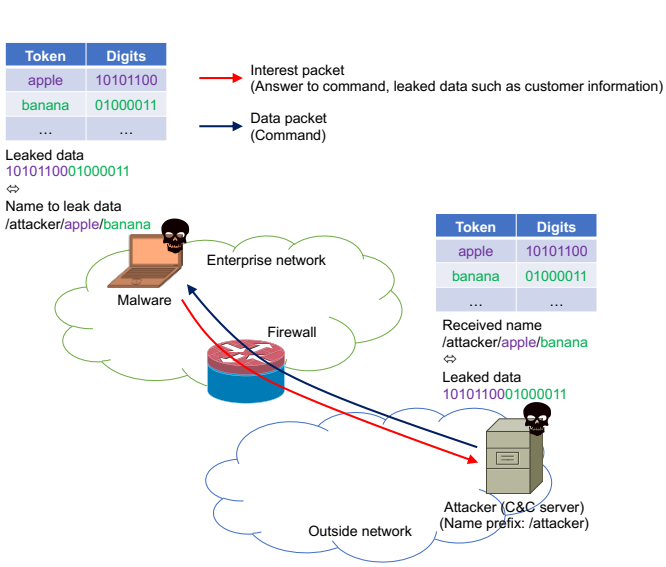
Our experiments show that the flow filter can effectively address the possible counter-attacks against the name filter. In the case that by mimicking a legitimate consumer, clever malware bypasses not only the name but also the flow filter while maximizing information leakage throughput, we demonstrate that the throughput is $1.32 \cdot 10^{-2}$ to $6.47 \cdot 10^{-2}$ times the one choked by the name filter alone, which shows a reduction of 2 orders of magnitude. Therefore, our approach will dramatically mitigate the impact of the information leakage attack in NDN.

The remainder of this paper is organized as follows. Section 2 describes an overview of NDN, information leakage attack models, an NDN name filter and its counter-attacks, and the related works as the background of this paper. Section 3 proposes our scheme to generate the corresponding NDN flow dataset from an HTTP flow dataset. Using the proposed conversion scheme, Section 4 obtains the NDN flow dataset from an HTTP flow dataset. Section 5 investigates several properties of malware generating malicious NDN flows. Section 6 introduces our NDN flow filter against information leakage through an Interest, while Section 7 presents our experimental evaluation. Section 8 discusses our findings and finally, Section 9 concludes this paper.

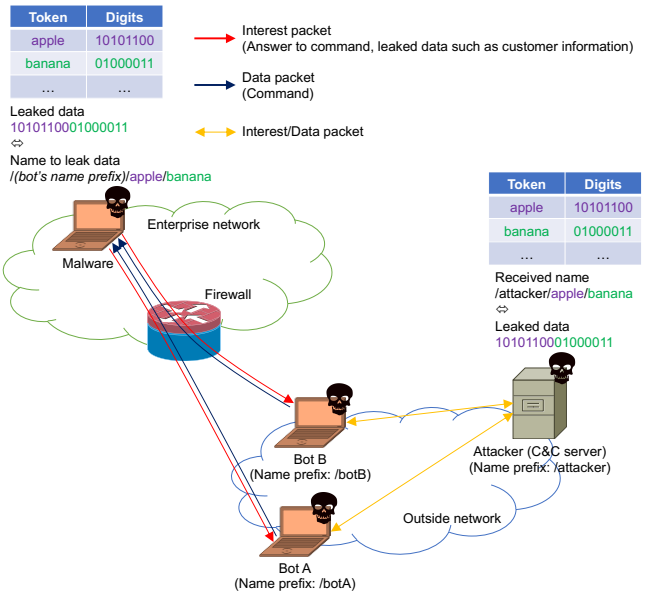
2. Background

2.1. Overview of NDN

NDN [4] adopts a "pull"-based network architecture, where each node consists of three elements: a *Content Store (CS)*, *Pending Interest Table (PIT)*, and *Forwarding Information Base (FIB)*. The CS caches Data packets with the caching replacement policy. The PIT keeps all pending Interests and their arrival interfaces in the entries, and each entry is removed when the matching Data is received or a timeout occurs. The FIB maintains routing information to forward Interests. Unlike the Internet, NDN relies on name routing, not IP one. In order to distribute content, a producer must advertise the name prefix (e.g., /DOCTOR/pub) to the NDN network. When consumers retrieve content, they specify the desired content name including the producer's name prefix (e.g., /DOCTOR/pub/video) in the Interest, and send it to the next NDN node. After receiving the Interest, the NDN node first checks the CS to know whether the corresponding Data is cached. If this is the case, the NDN node responds to the Interest with the cached Data. Otherwise, the PIT is searched to see whether information about the Interest has already been registered in the PIT. If it has been in the PIT, the NDN node adds the incoming interface of the Interest to the PIT and does not forward the Interest to the next NDN node. If not, the NDN node records the name and the incoming interface of the Interest in the PIT, and following the FIB, the NDN node forwards the Interest to the next NDN node. This process is repeated until the Interest finds the corresponding Data, and the Data can be returned by the producer or the intermediate NDN node having the cached Data. When the Data is sent to the consumer, the Data follows the reverse path of the Interest, which can be configured by the PITs of the relayed NDN nodes, and finally, the consumer receives the Data.



(a) 1-to-1 attack model.



(b) 1-to-N attack model.

Figure 1: 1-to-1 and 1-to-N attack model (i.e., malware to attacker and to N bots).

2.2. Information Leakage Attack Models

In what follows, we assume that an employee's PC in the enterprise network has been infected by malware of the attacker who is in the outside network (i.e., the employee is not involved in the attack), and the malware and attacker share the same table for steganography (which includes each token and the corresponding digits). There are two models of information leakage attack through an Interest [6, 7, 8] (Fig. 1):

- *1-to-1 attack model (i.e., malware to attacker):* Assuming that the malware knows the attacker's name prefix (e.g., /attacker) and the name prefix is routable from the malware, the malware directly communicates with the attacker, which should be the command-and-control server (C&C server) (Fig. 1a). When the malware sends out the leaked data, the leaked data is converted into the Interest name using the table for steganography. Here, the data is converted into the tokens "apple" and "banana." After the malware sends the Interest including the name to leak data to the attacker, the attacker decrypts the name using the shared table and obtains the leaked data. Then, the attacker can send the reply Data packet including, for example, the command for the malware.

- *1-to-N attack model (i.e., malware to N bots):* Assuming that the malware knows the bots' name prefixes (e.g., /botA, /botB) and they are routable from the malware, the malware indirectly communicates with the attacker via several bots (Fig. 1b). Like the 1-to-1 attack model, the 1-to-N attack model can also perform information leakage through an Interest.

The above attack process looks like accessing outside content (in fact, leaking data), so it is important to develop countermeasures against it.

2.3. NDN Name Filter and Its Counter-attacks

Information leakage through an Interest can be mitigated by constructing a name filter using URL-based NDN names [6, 7, 8]. The name filter checks a name in the Interest (like an HTTP GET request message on the Internet) and judges it as legitimate or not. When used against steganography-embedded Interest names (Section 2.2), this filter can choke information leakage throughput per Interest by up to 32.1 bytes/Interest and force malware to send 137 times more Interests in order to leak information [8].

Although this filter is efficient at reducing the throughput per Interest, two possible counter-attacks exist. The first would be to send a lot of Interests within a short period of time, each of which does not achieve high throughput per Interest and circumvents the filter. The second counter-attack would be to exploit an Interest with an explicit payload in the name (like an HTTP POST request messages on the Internet), which the name filter cannot effectively inspect, since the payload is unrestricted and depends on the consumer.

2.4. Related Work

One way to perform information leakage is to utilize *DNS tunneling* [13]. DNS tunneling is a threat where an

outside attacker and the malware inside an enterprise network bypass a firewall and perform tunneling of data and commands, by exploiting domain names in DNS queries and the corresponding DNS responses. An information leakage attack through an Interest in NDN (Section 2.2), imitates one through DNS tunneling in the Internet. Leijenhorst et al. [14] show that DNS tunneling can achieve up to 110 KB/s in throughput with DNScat [15], which is one of the DNS tunneling applications, but it adds huge traffic overhead. Merlo et al. [16] 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. [17] and Qi et al. [18] analyze character frequencies of domain names and detect DNS tunneling. Differently, Farnham [19] investigates not only such a payload analysis but also a traffic analysis such as analyzing count and frequency of queries. Ellens et al. [20] also perform the traffic analysis using a flow defined in RFC 3917 [21] and report that appropriate metrics to detect the tunneling are, for example, bytes per flow or the number of flows over time. Kara et al. [22] focus on DNS TXT record and detect DNS tunneling activities with this record. Aiello et al. [23] 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 at detecting attacks generated by specific tools (such as DNScat) or malware (such as Morto worm [24]), which means that the threat of the attack cannot be eliminated completely. Xu et al. [25] conclude that DNS-based botnet C&C channel, which is based on DNS tunneling, is "feasible, powerful, and difficult to detect and block".

These proposed remedies against DNS tunneling cannot be reused to detect an information leakage attack through an Interest in NDN. This is because, for instance, patterns of DNS traffic, which resolves a domain name, are different from ones of NDN traffic, which retrieves content. Thus, there is a need to propose new countermeasures against the information leakage attack through an Interest. In particular, this paper proposes an NDN flow filter to mitigate the drawbacks of the NDN name filter.

In this paper, we assume that our proposed flow filter can be implemented in an enterprise network firewall as shown in Fig. 1. Goergen et al. [26] propose a firewall in Content Centric Networking (CCN) [27], whose architecture is quite similar to NDN. This is one of the first papers that describe a CCN firewall, in which, however, the authors do not mention any information leakage attack through an Interest. To the best of our knowledge, such attacks have only been investigated by Kondo et al. [6, 7, 8].

3. Conversion of HTTP to NDN Flow

By assuming that applications over the current HTTP are run over NDN, in this section we propose how to generate the corresponding NDN flow dataset from an HTTP flow dataset.

In the Internet, a flow can be defined by 5-tuples (i.e., source/destination IP address, source/destination port number, and protocol type). Although many applications can generate the Internet flows, we focus on HTTP flows (Fig. 2) since HTTP is a name-centric protocol that is widely used to retrieve content named by URL [28]. Indeed, as HTTP has high affinity with NDN, some research efforts are derived from HTTP/NDN [29, 30]. Yuan et al. [31] defined NDN flow using a name, and each flow was maintained by the expiration time and a list of incoming faces. The IRTF ICN Research Group also discussed name-based flow [32]. Fig. 3 shows the name-based NDN flow. Hereafter, we discuss the NDN flow by assuming that the firewall shown in Fig. 3 is a monitoring point.

Fig. 4 illustrates the proposed conversion scheme from an HTTP flow using 5-tuples into an NDN flow. For the NDN flow, we follow a name-based flow idea discussed in [31] and [32], and utilize a name prefix to make the NDN flow. Note that [31] and [32] do not show a scheme to convert an HTTP flow into an NDN flow, which is one of the proposals in this paper. In the HTTP flow, the client (192.168.1.1) first obtains the HTML file (file.html) from the server whose IP address is 192.168.1.2, and parses the file in order to access the URLs in the HTML file. Then, the client generates two flows to 192.168.1.2 and 192.168.1.3, where each request message obtains its corresponding response message. When we convert the HTTP flow to NDN flow, we associate the destination IP address with the name prefix. In performing the conversion, we have to pay attention to the maximum size of a Data packet in NDN. Basically, in NDN one Interest retrieves one Data packet. When content size is larger than the maximum size of a Data packet (8800 bytes by default [33]), the content should be segmented into several Data packets, and multiple Interest packets corresponding to the Data packets can be generated. A manifest [34] is proposed, which includes a list of Data segment names (e.g., /b/video.mpg/0, /b/video.mpg/1) and meta-information of the Data packets. We assume that when a consumer accesses content, the consumer first obtains the manifest of the content, and the manifest can be collected by one Interest. To distinguish between an Interest to retrieve a manifest and an Interest specifying the segment name described in the manifest, we name them ***manifest Interest*** and ***sequenced Interest***, respectively. In Fig. 4, the consumer first retrieves the manifest of the HTML file (file.html) from the producer whose name prefix is /a. Sending the sequenced Interests listed in the manifest, the consumer then obtains and parses file.html, and continues to perform manifest-based content collection for image.jpg and video.mpg. These series of Interest packets make up the Interest flow to obtain the target object as well as the corresponding Data flow. We call the flow to obtain each object as an ***NDN subflow***. Thus, a set of NDN subflows for each producer constitutes an NDN flow.

When an Interest appends a payload into the name

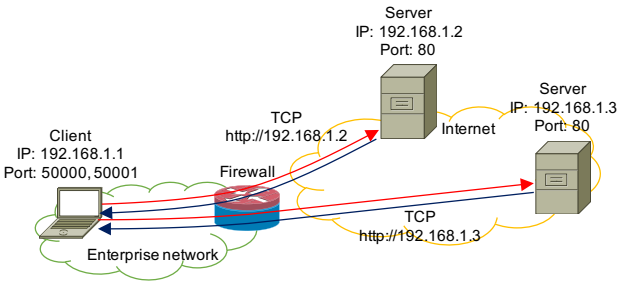


Figure 2: HTTP flow.

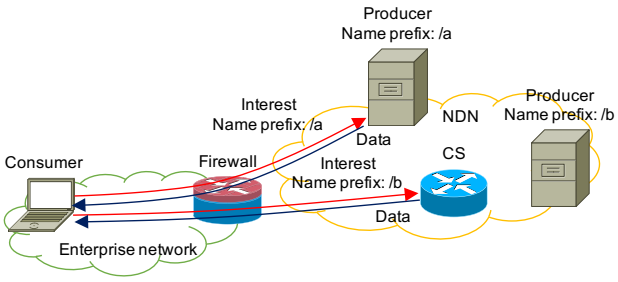


Figure 3: NDN flow.

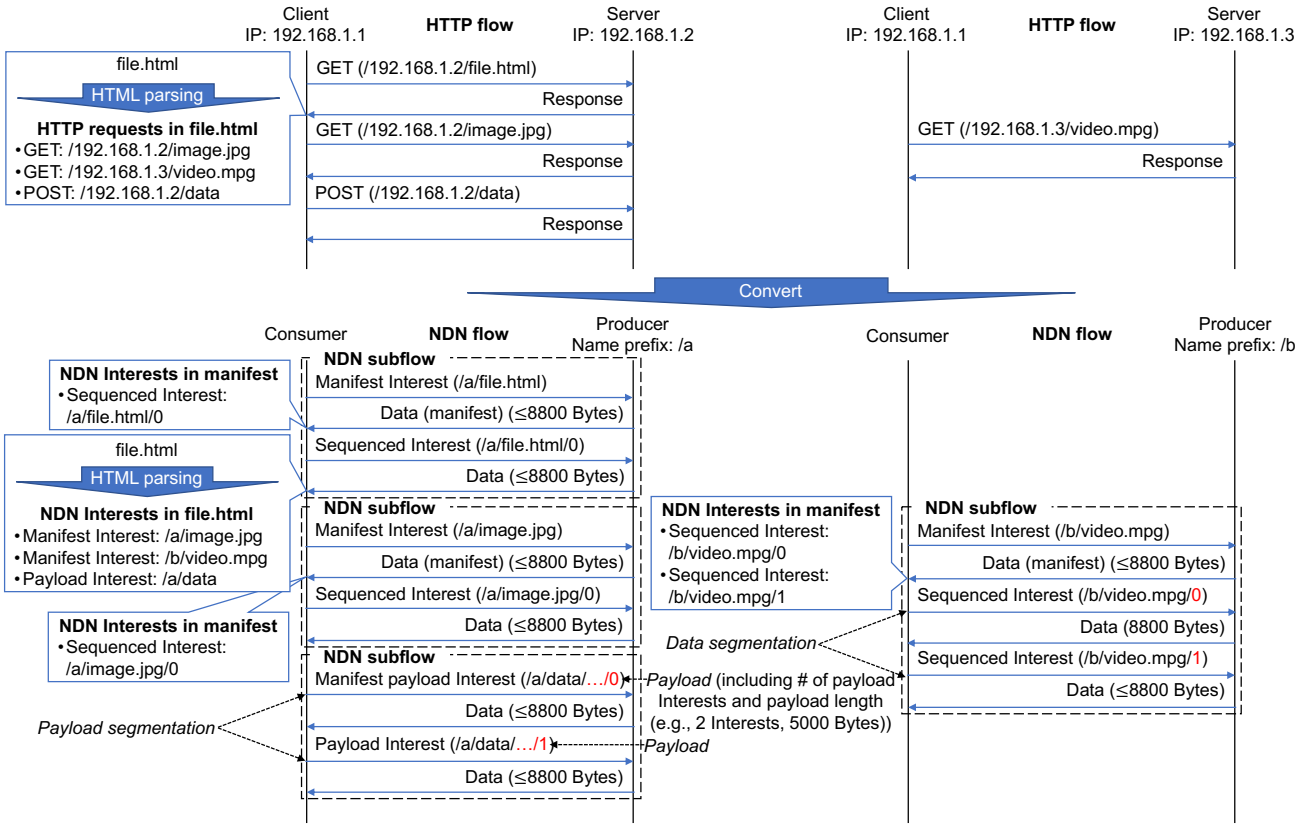


Figure 4: Conversion of HTTP to NDN flow.

like /a/data (e.g., a Signed Interest [35] for the current implementation, an Interest name carrying consumer-side information [36], an Interest for content search with keywords specified by a consumer [37, 38], etc.), the length of the content name may become longer than the maximum Data packet size. In this case, the Interest has to be divided into the multiple Interests whose name lengths leave some space for content in the Data, since the Data packet has to include the corresponding name. We call an Interest with a payload in the name a ***payload Interest***. To let the producer know how many successive Interests will be received and how long payload these Interests will have, the first payload Interest should insert the number of successive Interests and the payload length in its payload. We name the first payload Interest as a ***manifest payload Interest***. To the best of our knowledge, a protocol to split an Interest that includes a large payload into several successive Interests, considering a maximum packet size of a Data, has not been discussed in detail in NDN research.

Table 1 presents an analogy between HTTP and NDN flow features monitored at the firewall in Fig. 3. According to the assumption about tracking each consumer, a source and a destination address in the HTTP flow should be a MAC/IP address of the consumer and a name prefix in the NDN Interest flow, respectively. When the consumer utilizes a signed Interest, the Interest includes the signature which has to be verified by the consumer public key, so that the signature in the signed Interest should also correspond to the source address in the HTTP flow. Likewise,

Table 1: Analogy between HTTP and NDN flow features

HTTP flow	NDN flow at firewall	
	Interest flow	Data flow
Source address	MAC/IP†address (and signature††)	Name prefix and signature
Destination address	Name prefix	MAC/IP†address
Requested URL	Name	Name
GET request message	Manifest Interest	N/A
POST request message	Manifest payload Interest and payload Interests	N/A
GET/POST response message	N/A	Data
HTTP response timeout	Expiration time	N/A
Segmentation (done by TCP)	Manifest payload Interest	Manifest
Session time (derived from TCP)	# of all Interest packets	# of all Data packets

† In the case of NDN over IP.
†† Signed Interest (optional).

the name prefix and producer signature of the Data in the Data flow should be the source address, and the MAC/IP address of the consumer in the Data flow should correspond to the destination address. Basically, a requested URL, GET/POST request message, GET/POST response message, and HTTP response timeout in the HTTP flow correspond to a content name, an Interest packet, a Data packet, and expiration time of an Interest in the NDN flow, respectively. As for the GET request message, the message is a trigger to retrieve content and corresponds to a manifest Interest in NDN. The POST request message corresponds to a manifest payload Interest and the subsequent payload Interests. NDN does not have a transport layer; hence, segmentation done by TCP has to be realized by an NDN application. Thus, a manifest payload Interest and a manifest, which are operated by the NDN application, will compensate for this lack. In the HTTP flow, session time can be measured between SYN and the last packet of the HTTP flow. In contrast, in NDN a consumer does not make a session for a specific producer. The above NDN flow generation algorithm divides the HTTP session into several NDN subflows, each of which corresponds to getting or posting a specific object. The duration of an Interest subflow correlates with the number of Interest packets; thus, it is approximately linear to the time spent by the corresponding TCP session for the GET/POST request. The sum of these Interest packets can be seen to have a linear relation to the whole TCP session of the HTTP.

As for an HTTP flow, using different source port numbers, one client can create multiple HTTP flows to the same server at the same time. But in NDN, there is neither a source nor a destination port number such that HTTP flows can be aggregated into one NDN flow, as shown in Fig. 5. Our NDN flow definition can better reflect the load of a specific producer. In order to decide how many HTTP flows should be aggregated, we have to define a threshold for the packet time interval T_{thr} between HTTP flows. If the time interval is less than T_{thr}, then the flows can be combined. Appendix A presents one example of how to set T_{thr} using an NDN flow dataset described in Table 6. Though this parameter setup does not affect evaluations of a risk of information leakage attacks at all, investigation of T_{thr} is an interesting research direction in terms of flow management, which we keep as a future work.

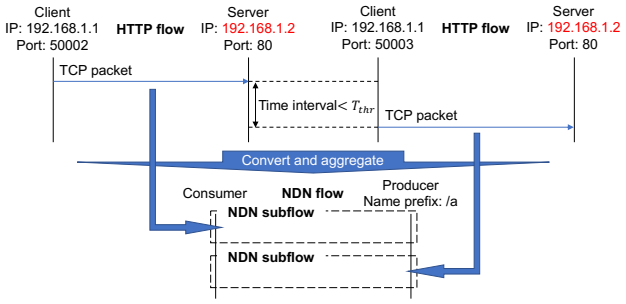


Figure 5: HTTP flow aggregation into NDN flow.

Note that NDN installs CS (i.e., cache), which can distribute content instead of the content producer, so that the NDN flow is not an end-to-end flow like the HTTP flow (Fig. 2) but a hop-by-hop flow, which might be terminated at an intermediate NDN node's CS (Fig. 3). Indeed, HTTP can also utilize cache such as a cache server (e.g., proxy) and browser cache. Even if the cache server is used, the destination IP address in the HTTP flow is that of the cache server such that an end-to-end flow can be still generated. **When we can obtain an HTTP flow dataset monitored at the firewall to which every consumer is directly connected via the underlay protocol such as an Ethernet or IP network**, we can generate the corresponding NDN flow dataset using the HTTP flow dataset. Moreover, the firewall can distinguish each consumer with a MAC/IP address as shown in Table 1.

Ongoing researches in NDN have focused on several functions such as congestion control [39], which should be

done by TCP in the Internet, but their standardization has yet to be achieved. Thus, in this study, we assume that NDN installs TCP-like functions.

4. Dataset Creation

Here, following Section 3, we create the corresponding NDN flow dataset from existing traffic traces.

4.1. HTTP Flow Dataset

4.1.1. Why Waikato Traces?

We utilized Waikato University traffic traces in Waikato VIII (duration: Apr/7/2011–Nov/5/2011) [10]. In these traces, the IP addresses were anonymized by Crypto-PAn AES encryption, which is a prefix-preserving anonymization method. The encryption key was changed once a week, and therefore we can track each consumer for one week at the most. We focus on three traces[1], each of which is a one-day trace. These traces were captured from

- Wednesday, June 8, 12:00:01 (local time) to Thursday, June 9, 12:00:00 (local time),
- Monday, June 13, 12:00:01 (local time) to Tuesday, June 14, 12:00:00 (local time),
- and Wednesday, July 13, 12:00:01 (local time) to Thursday, July 14, 12:00:00 (local time).

There are two reasons to exploit the traces (i.e., advantages).

✓ *Less HTTPS Traffic, More HTTP Traffic:* Recently, HTTPS traffic volume has drastically increased such that the application payload is encrypted, so that the request message information, such as GET or POST, cannot be extracted at all. However, the traces we exploited were created in 2011; in fact, for outgoing flow from the university, the ratios of HTTPS traffic flow (destination port number: 443) to the whole Web traffic flow (HTTP (destination port number: 80 or 8080) and HTTPS traffic flow) are 15.03%, 10.17%, and 8.91%, respectively, in the three traces used. Thus, in order to create an NDN flow dataset, we can obtain much more information from these traces than the latest trace. According to [40], Facebook and YouTube started to support HTTPS in April 2013 and January 2014, respectively, which means that the traffic about such popular content providers in the latest trace cannot be analyzed. Thus, the latest trace does not always describe a content access model of user well.

✓ *Availability of All Traffic Generated by Users:* The capture point of the traces was located between Waikato University's network and the Internet. This means that all the traffic generated by users in the university passed through the capture point. On the other hand, even if ISP traffic is captured at one point, it is possible for some traffic not to go through the point. In summary, the university traffic can imitate a hop-by-hop flow between a consumer and a capture point such as an enterprise network firewall.

However, there is a limitation of the traces.

× *Unavailability of detailed application payload information:* Usually, in an open dataset, an application payload is removed completely for the purpose of user privacy. In the Waikato HTTP traces, each application payload is truncated to the first 4 bytes, which makes it difficult to analyze the traces deeply.

Even in such a situation, we can understand the kind of request message such as GET or POST from the first 4 bytes. Note that even though the dataset provided by Waikato University does not include requested URLs, this does not affect the modeling of some metrics in NDN such as the number of Interest packets within one subflow, which depends on the response message size. This is another reason to utilize the traces (the other open datasets do not provide such information).

4.1.2. HTTP Flow Extraction

First, using pkt2flow [41], we extracted the HTTP flows from the traces based on 5-tuples. In order to make a flow filter, we specified the IP address including the IP prefix of Waikato University, 80 or 8080, and TCP as a source IP address, a destination port number, and a protocol type, respectively. Then, we set 30 min (default value in pkt2flow) as the flow timeout and ignored the flows that did not include the TCP SYN packet. For the HTTP request messages in the extracted flows, we focused on only GET and POST since we converted these two messages into NDN subflows. In cases when other request messages such as HEAD existed in the HTTP flows, we neglected the flows that included these request messages.

Table 2 shows a summary of our HTTP flow datasets. With regard to the number of Waikato IP addresses, it might not be equal to the number of clients in Waikato University because of network address translation. However, we assume that the number of Waikato IP addresses corresponds to the number of clients.

Table 2: Summary of HTTP flow datasets

	Jun/8–9	Jun/13–14	Jul/13–14
# of Waikato IP addresses	767	760	908
# of HTTP flows by only GETs	1,125,679	1,125,362	1,813,429
# of HTTP flows by only POSTs	246,233	322,384	229,393
# of HTTP flows by GETs/POSTs	28,550	31,553	10,932

[1] We assumed that the HTTP traffic using HTTP pipelining was negligible and the traces consisted of HTTP traffic without HTTP pipelining. This assumption is based on the fact shown in Appendix B.

4.2. NDN Flow Dataset

Table 3 shows a summary of our NDN flow datasets after the conversion of the HTTP flow datasets described in Section 4.1. Since each HTTP flow can consist of multiple GETs and/or POSTs, the number of converted NDN subflows is larger than that of HTTP flows.

Table 3: Summary of NDN flow datasets

	Jun/8–9	Jun/13–14	Jul/13–14
# of Waikato consumers	767	760	908
# of NDN subflows by manifest Interest	3,167,397	3,183,382	2,841,950
# of NDN subflows by manifest payload Interest	359,083	443,360	277,641

5. Properties of Malware Generating Malicious NDN Flows

In this section, we investigate properties of malware generating malicious NDN flows. We define malicious NDN flows as ones causing information leakage and assume that the firewall monitors the traffic of each consumer for a specified period T.

In order to retrieve content, a consumer first needs to obtain the content's manifest using the manifest Interest (Section 3). Since the names of successive sequenced Interests generated by each manifest do not have any letters other than their sequence numbers, the malware can exploit only the manifest Interest as a steganography-embedded Interest (Section 2.2), and this manifest Interest could be inspected by a name filter (Section 2.3). Thus, in order to obtain a high throughput, the malware has to rapidly send manifest Interests for information leakage to the attacker or the bots. Moreover, once the payload Interests are exploited, malware can ameliorate the information leakage throughput since the name filter cannot be applied to these Interests.

Within a specified period T (where the firewall monitors each consumer traffic), a naive information leakage throughput improvement method by malware will adopt the four properties listed in Table 4. When *Properties 1* and *2* are satisfied by malware, we call it ***manifest Interest-based malware***. On the other hand, when *Properties 3* and *4* are satisfied, we call the malware ***payload Interest-based malware***. When the malware shows *Properties 1, 2, 3,* and *4*, we call it ***manifest and payload Interest-based malware***. These properties will be referred to define a feature vector in Section 6 and analyzed in Section 8.

Table 4: Properties of malware generating malicious NDN flows

Property 1	The number of manifest Interests will be large.
Property 2	The number of sequenced Interests per manifest Interest will be small.
Property 3	The number of manifest payload Interests will be large.
Property 4	The size of data sent by payload Interests will be large.

6. NDN Flow Filter

6.1. Overview

In order to protect against such NDN flow attacks for information leakage, we need a filter (installed in the firewall) that determines whether an NDN flow is legitimate or malicious. Instead of manually programming such a filter, we aim to build it using machine learning.

One of the standard procedures is to use a classification algorithm that builds a model using legitimate and malicious activities, which then predicts whether an activity is legitimate or malicious. In our case, we do not have any actual legitimate or malicious data. For this reason, we followed a different methodology. First, we classified the consumers as either normal (inliers) or anomalous (outliers) by considering features extracted from their *long-term* activity[2] (24h). Then, using this labeled one-day dataset, we made the firewall respond quickly by monitoring each consumer's *short-term* activity (over the last few minutes). This is motivated by the requirement that a firewall cannot wait until the end of the day to decide whether certain activities are legitimate or not. The flow filter is trained with data coming from normal and anomalous consumers. Assuming that these activities are likely to be legitimate and malicious respectively, the flow filter can be used for the detection of malicious activities. Hereafter, we use the terms "legitimate" and "anomalous" instead of "legitimate" and "malicious."

6.2. Isolation Forest

In order to mark a consumer as legitimate or anomalous, we use an isolation forest [11] (IF), which is a state-of-the-art anomaly detection algorithm. An IF is a collection of trees, each one "isolating" data points from the rest by random, recursive partitionings of the feature space. The key idea is that an outlier is different from most of the data points in terms of feature values, thus, it is more likely to be isolated using *less* splits, i.e., being closer to the root of the tree. On the other hand, an inlier is more likely to be isolated using *more* random splits of the feature space, thus, its "path length" or depth in the tree will be greater. The IF calculates an anomaly score for each data point, based on this "path length" value averaged over all trees. These simple operations make the IF an efficient model with a linear time complexity and low memory requirements.

6.3. Methodology

In order to build the filter, given our above restrictions, we perform the following steps (see Fig. 6):

(1) we split an NDN flow dataset into l consumer datasets;

[2]This was important since the longer an activity is monitored the easier it is to determine whether it is anomalous or not. For example, an anomalous consumer could be one that causes heavy traffic over a long time span.

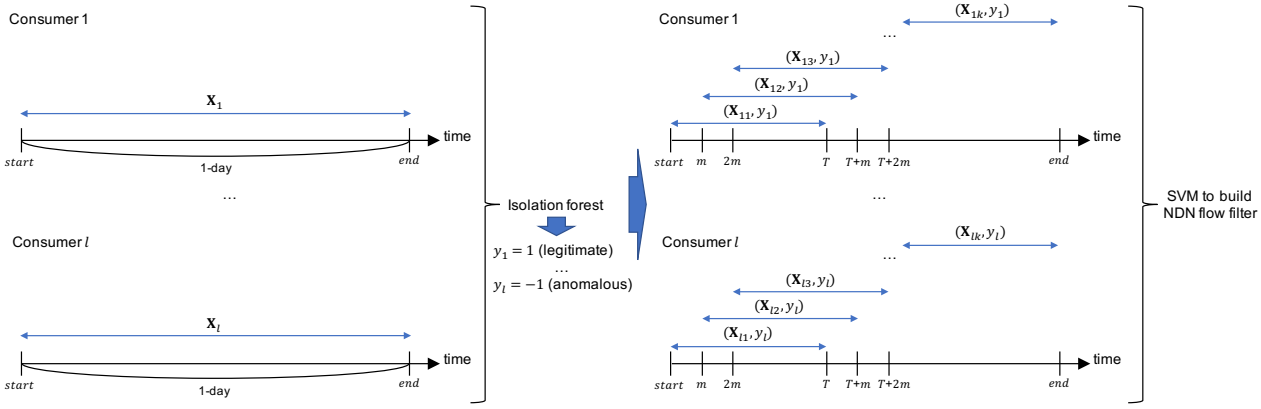


Figure 6: Dataset creation to build NDN flow filter.

(2) we extract a feature vector $\mathbf{x}_i$ (described below and Table 5) for each consumer i based on its *long-term* activity (24 h) and create a dataset $D_C = \{\mathbf{x}_1, \ldots, \mathbf{x}_l\}$;

(3) we provide D_C to an IF [11], to mark the consumers as either legitimate ($y_i = 1$) or anomalous ($y_i = -1$);

(4) we split the dataset of each consumer i into k window datasets of size T (*short-term*) shifted by an offset m, and extract a feature vector $\mathbf{x}_{ij}$ (described below and Table 5) for all $j = 1, \ldots, k$ window datasets;

(5) we create a labeled dataset for classification by associating each $\mathbf{x}_{ij}$ with its corresponding label y_i, calculated at step (3), $D_W = \{(\mathbf{x}_{ij}, y_i)\}_{i=1:l, j=1:k}$.

(6) we use D_W to train an SVM classifier [12], which is able to construct an optimal hyperplane that separates legitimate from anomalous activities.

Table 5 indicates features extracted from a flow window of the NDN consumer. The first feature distinguishes two attack models (i.e., 1-to-1 and 1-to-N attack model) shown in Section 2.2. The rest of the features are derived from Table 4.

7. Experiments

7.1. Experimental Setup

7.1.1. Legitimate and Anomalous NDN Consumer Dataset

We classified the NDN consumers into legitimate and anomalous sets using the NDN flow datasets derived from the HTTP flow datasets (Table 3) following the procedure described in Section 6. At step (3), we configured the appropriate outlier proportion, which is one of the parameters of the isolation forest[3]. For two outlier proportions, 0.005 and 0.01, we visualized the inliers and outliers using isomap [43] in two dimensions[3] (Fig. 7). With the outlier proportion set to 0.005 (Fig. 7a), the isolation forest separated the outliers more clearly from the inliers near the coordinate origin than when the outlier proportion was set to 0.01 (Fig. 7b). Thus, we set the proportion to 0.005. Table 6 summarizes the obtained NDN flow datasets generated by legitimate and anomalous consumers after labeling flows from inliers and those from outliers as legitimate and anomalous, respectively. The statistics about the NDN subflows in the datasets are given in Appendix A.

7.1.2. Window Dataset

To build the SVM-based NDN flow filter described in Section 6, as shown in Fig. 6, we need to obtain the window datasets from the NDN flow datasets given by Section 7.1.1. Table 7 shows the parameter settings for window T and offset m to extract the NDN window datasets. We increased T exponentially starting at 60 s from Case 1 to Case 4, and starting at 120 s from Case 5 to Case 7. From Case 1 to Case 4, we set each m as 0.2 multiplied by the corresponding T. From Case 5 to Case 7, we fixed m at 12 s. It is possible for consumers not to generate any traffic in a specified period T; in this case, we consider that the corresponding window dataset does not exist.

7.1.3. Training, Validation, and Test Sets

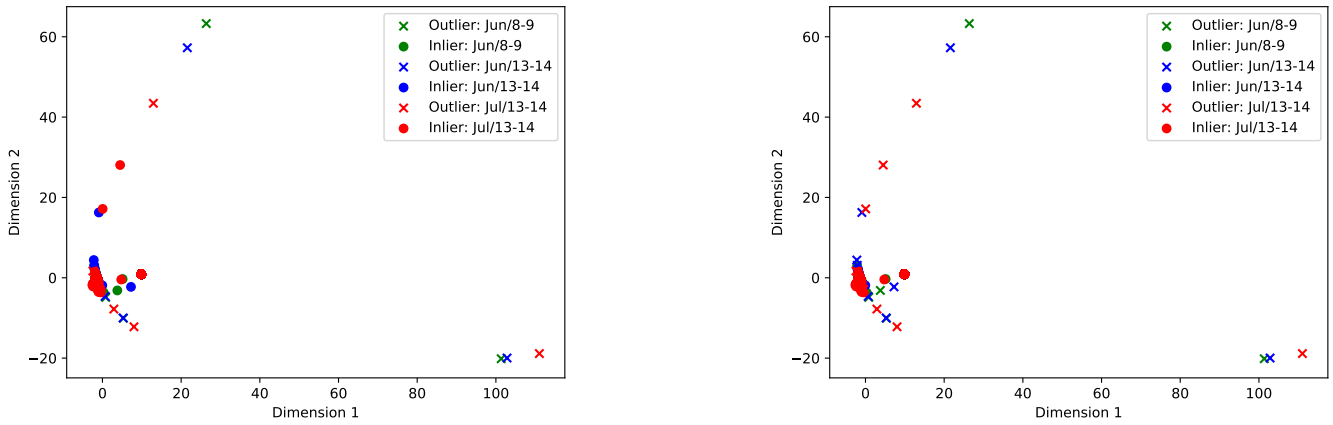
We built seven NDN flow filters (each corresponding to Cases 1 to 7 described above) based on 70% of the Jun/8–9 dataset as the training and validation sets. To do so, we performed feature scaling and a grid search on the hyperparameters[3] using five-fold cross-validation. Finally, we evaluated the performance using 30% of the Jun/8–9 dataset as the test set.

In addition, we used 100% of the Jun/13–14 and Jul/13–14 datasets as the test sets. This is done to evaluate the vulnerability of the training process, since we expect that other days will have a different distribution, which should be reflected as worse results in the classification performance of the trained filters.

[3]See Appendix D for parameter settings.

Table 5: Features extracted from flow window of NDN consumer (manifest Interest, sequenced Interest, manifest payload Interest, and payload Interest are defined in Section 3)

Notation	Feature variable
$N_{Producer}$	Total # of accessed producers
$N_{ManifestInterest}$	Total # of manifest Interests
$N_{SequencedInterest}$	Total # of sequenced Interests
$N_{ManifestPayloadInterest}$	Total # of manifest payload Interests
$S_{Payload}$	Total payload size of payload Interests
$\sigma_{N_{ManifestInterest}}$	STDEV of # of manifest Interests per producer
$\sigma_{N_{SequencedInterest}}$	STDEV of # of sequenced Interests per producer
$\sigma_{N_{ManifestPayloadInterest}}$	STDEV of # of manifest payload Interests per producer
$\sigma_{S_{Payload}}$	STDEV of payload size of payload Interests per producer



(a) 0.005 outlier proportion.

(b) 0.01 outlier proportion.

Figure 7: Visualization by isomap.

Table 6: Summary of legitimate and anomalous NDN flow datasets

	Jun/8–9		Jun/13–14		Jul/13–14	
	Legitimate	Anomalous	Legitimate	Anomalous	Legitimate	Anomalous
# of Waikato consumers	763	4	756	4	903	5
# of NDN subflows by manifest Interest	1,055,045	2,112,352	911,194	2,272,188	871,669	1,970,281
# of NDN subflows by manifest payload Interest	48,916	310,167	36,061	407,299	44,533	233,108

Table 7: Parameter settings for window T and offset m

	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
T [sec]	60	120	240	480	120	240	480
m [sec]	12	24	48	96	12	12	12

7.2. Results

7.2.1. Performance of NDN Flow Filter against Anomalous NDN Flows

Table 8 shows the performance of the seven filters in terms of precision (P), recall (R), and F_1 score (see Appendix C) on all test sets. The results show that the legitimate class can easily be classified from the *short-term* windows of all trained filters (Cases 1 to 7), since all metrics (P, R, F_1) have a high value (≥ 0.94) on all test sets. A perfect score of 1.0 is not achieved since it is possible for a legitimate consumer to generate anomalous flows (which would be classified as anomalous).

On the other hand, the results of the anomalous class on the test set of Jun/8–9 (the day from which the training and validation sets were generated) show the highest P for Case 7 (0.99), the highest R for Case 7 (0.80), and the highest F_1 for Case 7 (0.89), and indicate that all metrics (P, R, F_1) increase as the window size increases (from Cases 1 to 4 and Cases 5 to 7). It is worth noticing that

Table 8: Performance of NDN flow filter (based on Jun/8–9 dataset) in terms of precision (P), recall (R), and F_1 score on test sets

		Case 1			Case 2			Case 3			Case 4			Case 5			Case 6			Case 7		
		P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1	P	R	F_1
Jun/8–9	Legitimate	0.97	0.99	0.98	0.98	1.00	0.99	0.99	1.00	0.99	0.99	1.00	0.99	0.98	1.00	0.99	0.99	1.00	0.99	0.99	1.00	1.00
	Anomalous	0.93	0.69	0.79	0.94	0.73	0.82	0.96	0.76	0.85	0.96	0.79	0.86	0.94	0.74	0.83	0.98	0.78	0.87	0.99	0.80	0.89
Jun/13–14	Legitimate	0.94	0.99	0.96	0.96	0.99	0.98	0.97	0.99	0.98	0.98	1.00	0.99	0.96	0.99	0.97	0.97	0.99	0.98	0.97	1.00	0.98
	Anomalous	0.84	0.43	0.57	0.86	0.49	0.63	0.86	0.58	0.69	0.86	0.61	0.71	0.86	0.48	0.62	0.87	0.57	0.69	0.86	0.38	0.53
Jul/13–14	Legitimate	0.94	0.99	0.96	0.95	0.99	0.97	0.96	0.99	0.98	0.97	1.00	0.98	0.95	0.99	0.97	0.96	0.99	0.98	0.97	1.00	0.98
	Anomalous	0.84	0.42	0.56	0.86	0.43	0.57	0.82	0.43	0.56	0.85	0.43	0.57	0.85	0.42	0.57	0.84	0.43	0.57	0.79	0.30	0.44

all metrics of the anomalous class have lower value than those of the legitimate class; this is expected since (1) in our dataset, there are only very few points for the anomalous class, compared to the legitimate one, and results in a suboptimal decision boundary; and (2) the anomalous consumers do not only generate anomalous flows, but create a lot of legitimate traffic as well.

The test sets of days Jun/13–14 and Jul/13–14 show a different story. In particular, P ranges from 0.84 to 0.87 for Jun/13–14, and from 0.79 to 0.86 for Jul/13–14; R ranges from 0.38 to 0.61 for Jun/13–14, and from 0.30 to 0.43 for Jul/13–14; F_1 score ranges from 0.53 to 0.71 for Jun/13–14, and from 0.44 to 0.57 for Jul/13–14. The reduction in performance compared to Jun/8–9 is more pronounced with the date difference (i.e., the results are worse for Jul/13–14 which is 1 month later compared to Jun/13–14 which is 5 days later).

7.2.2. Information Leakage Throughput of Malware Mimicking Legitimate

The information leakage throughput can be calculated by the following formula: $R_{Leaked} = 8 \times (S_{LeakedData} \times N_{ManifestInterest} + S_{Payload})/T$ bps[4], where the $S_{LeakedData}$ indicates the leaked data size by a manifest Interest, and the $N_{ManifestInterest}$ and $S_{Payload}$ are given in Table 5. To evaluate the throughput, we assume that the firewall installs not only the flow filter but also the name filter to inspect a manifest Interest. We set $S_{LeakedData}$ to 32.1 bytes, i.e., the maximum throughput per manifest Interest that can be choked by the name filter against a steganography-embedded Interest [8]. Picking up a window dataset with the maximum throughput R_{Leaked} out of the whole dataset, we will get the corresponding $N_{ManifestInterest}$ and $S_{Payload}$. This dataset is categorized as legitimate by the name filter, while the feature vector with these $N_{ManifestInterest}$ and $S_{Payload}$ is detected as anomalous by the flow filter. Thus, we assume that malware is clever and tries to mimic a legitimate consumer, and then maximizes R_{Leaked} in this restriction. In the following discussion, we maximize R_{Leaked} by benchmarking one window dataset from the training and validation dataset, which is identified as legitimate by the flow filter.

To evaluate the 1-to-1 attack model, we should pick one legitimate window dataset, $N_{Producer}$ of which is 1.

[4] 1 byte = 8 bits

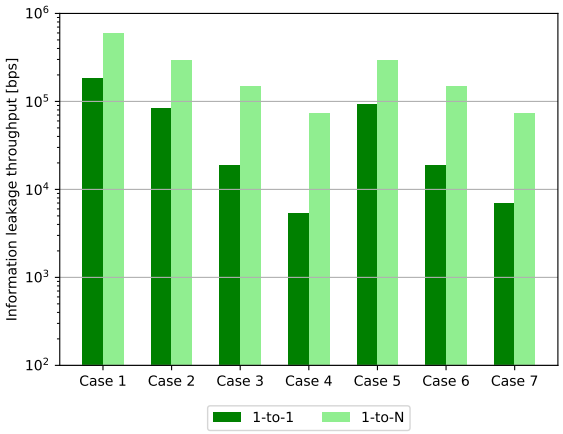


Figure 8: Maximum information leakage throughput in 1-to-1 and 1-to-N attack models.

Table 9: Number of bots to perform maximum information leakage throughput

# of bots	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
	5	12	25	40	12	25	38

Likewise, to evaluate the 1-to-N attack model, we should pick one legitimate window dataset with $N_{Producer} \geq 2$. Fig. 8 shows the information leakage throughput in the 1-to-1 and 1-to-N attack models, which is calculated using the Jun/8–9 training and validation window dataset. The number of bots required to achieve maximum information leakage throughput is between 5 and 40 (Table 9). By exploiting several bots, the 1-to-N attack model achieves higher information leakage throughput than the 1-to-1 attack model. Moreover, in the 1-to-N attack model, the Data packet that the malware receives includes the bot's signature; hence, in terms of attacker anonymity, the 1-to-N attack model would be more useful. In summary, from attacker's point of view, the 1-to-N model is preferable, but as window T increases (i.e., from Cases 1 to 4 and from Cases 5 to 7), the number of needed bots increases to maximize the throughput.

Compared to a manifest Interest using steganography, a payload Interest might leak much more information easily since the name filter cannot inspect it and the payload does not have to be encoded. Thus, banning payload In-

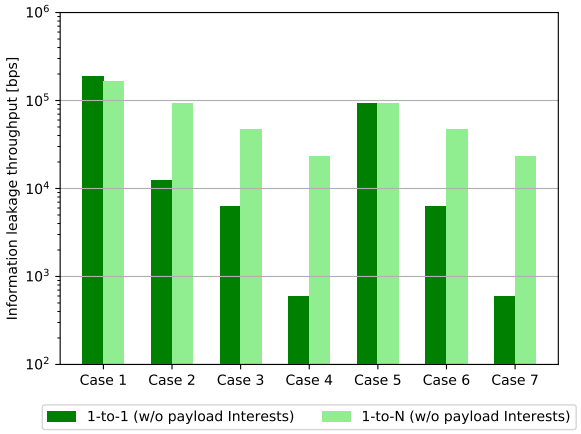


Figure 9: Maximum information leakage throughput in 1-to-1 and 1-to-N attack models under banning payload Interests.

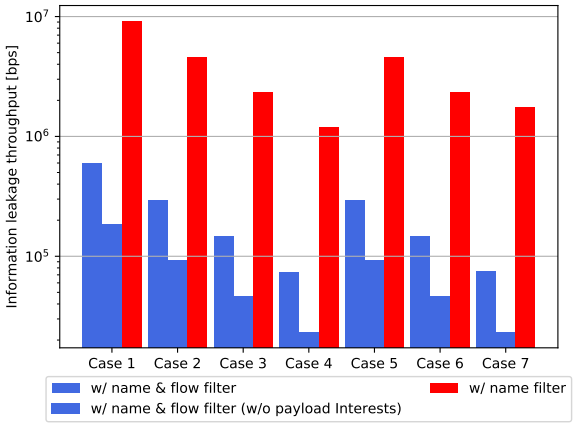


Figure 10: Maximum information leakage throughput choked by name and flow filter, by name and flow filter under banning payload Interests, and by only name filter.

Table 10: Number of bots to perform maximum information leakage throughput under banning payload Interests

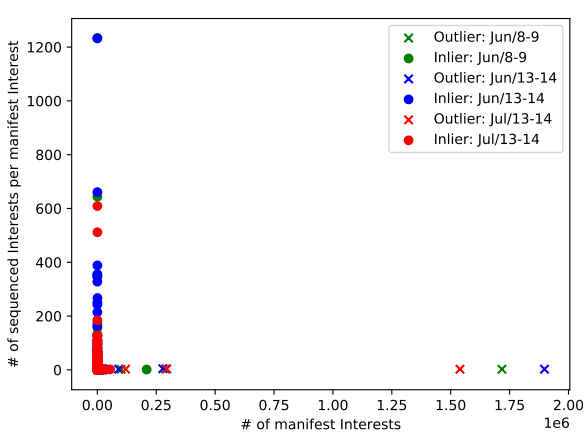
	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7
# of bots	2	3	4	51	3	4	54

terests might be a simple and effective solution against the information leakage attack, which means that only pulling content is accepted. Fig. 9 shows the information leakage throughput in the 1-to-1 and 1-to-N attack models under banning payload Interests, which is calculated using the Jun/8–9 training and validation window dataset. To obtain the maximum information leakage throughput, we pick one legitimate window dataset which maximizes R_{Leaked} while replacing $S_{Payload}$ to zero. The number of bots required to achieve maximum information leakage throughput is between 2 and 54 (Table 10). In any cases except Case 1, the 1-to-N attack model can achieve higher information leakage throughput than the 1-to-1 attack model, so the discussion is much the same as the one about Fig. 8.

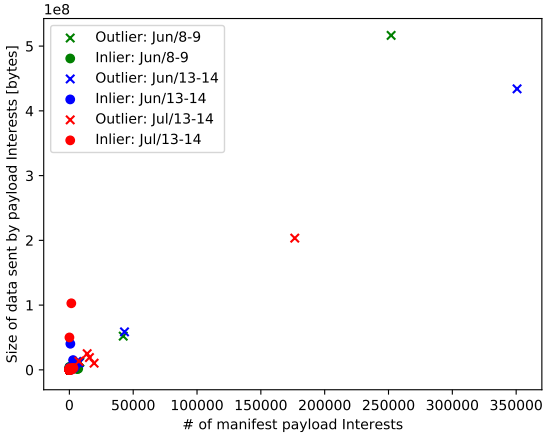
Using the Jun/8–9 window dataset, we also calculate the maximum information leakage throughput for three filters: the name and flow filter, the name and flow filter under banning payload Interests, and the only name filter [8] that is a comparison to our proposals in this paper (Fig. 10). In all cases, the throughput choked by the name and flow filter and by the name and flow filter under banning payload Interests are significantly lower than the throughput choked only by the name filter. This shows that the flow filter effectively solves the main problem of the name filter where the malware sends a lot of manifest and payload Interests in a short period of time. The throughputs after applying the name and flow filter to the traffic are $6.47 \cdot 10^{-2}$, $6.44 \cdot 10^{-2}$, $6.39 \cdot 10^{-2}$, $6.28 \cdot 10^{-2}$, $6.44 \cdot 10^{-2}$, $6.37 \cdot 10^{-2}$, and $4.24 \cdot 10^{-2}$ times those after applying only the name filter to the traffic in Cases 1 to 7, respectively, while the throughputs choked by the name and flow filter under banning payload Interests are $2.02 \cdot 10^{-2}$, $2.01 \cdot 10^{-2}$, $1.99 \cdot 10^{-2}$, $1.96 \cdot 10^{-2}$, $2.01 \cdot 10^{-2}$, $1.98 \cdot 10^{-2}$, and $1.32 \cdot 10^{-2}$ times those choked only by the name filter in Cases 1 to 7, respectively. After applying the name and flow filter to the traffic, the throughput can be reduced to 74.4 Kbps, and moreover, the throughput choked by the name and flow filter under banning payload Interests can be reduced to 23.2 Kbps.

8. Discussion

The results of Table 8 show that the filters are no longer effective 5 days (Jun/13–14) and 1 month (Jul/13–14) after the date (Jun/8–9) when the training and validation sets to build the filters were captured. It is also striking that the results of Case 7 at these later dates become worse than the ones of Case 6. This indicates that the distribution of anomalous data is different at these later dates, or in other words, the anomalous consumers have different features than the ones of the training day. It also highlights the need for training the filters periodically with up-to-date information. In particular, in this paper, we intentionally used a one-shot training process, rather than a continual one, to showcase this vulnerability. On a deployed system, the filter should be periodically replaced (e.g., every couple of days) with a new one that is trained using a dataset which contains both the past activities as well as the most recent ones. Doing so, however, might have the negative consequence of slowly accepting the malware activities as legitimate, if the outlier proportion remains fixed. In other words, more data do not necessarily imply more malware, thus, a smarter approach is required, e.g., by using additional information from a density-based clustering model to adjust the outlier proportion accordingly.



(a) Two metrics about *Properties 1* and *2*.



(b) Two metrics about *Properties 3* and *4*.

Figure 11: Four metrics of outliers and inliers.

The value of the outlier proportion is difficult to predict for arbitrary datasets. This difficulty comes from the fact that currently, to the best of our knowledge, no labeled dataset exists that would enable a machine learning model to infer information leakage through an Interest under various settings (e.g., some poorly maintained network may have many compromised machines participating in the leakage vs. a well-secured network with very few compromised machines). For this reason, we focused on a specific dataset which we assumed comes from a "well-secured network with very few compromised machines" (this is reflected in Fig. 7, as well as the fact that the outliers may satisfy the malware properties outlined in Table 4). Creating such labeled datasets is an important endeavor for future work. Having said that, it has to be noted that our current evaluation method is generic if we assume the setting "well-secured network with very few compromised machines": the proportion of 0.005 was found to better separate the outliers in this dataset, but a different outlier proportion could be found for other datasets using this procedure.

The SVM models we use in this work have two limitations. First, their non-parametric nature makes them intractable to use with huge datasets since their computational complexity scales cubicly ($O(n^3)$) with the number of data points. Second, they do not have a mechanism for remembering information that they have seen in the past. A more robust and flexible filter could be made by employing types of parametric machine learning models that integrate temporal information, such as recurrent neural networks (RNNs) [44]. Such models could in principle remember past information and output at a given time step the probability that the activity is anomalous or legitimate, rather than deterministically perform a binary classification as our SVM models.

It is worth noting that in general, the extracted outliers might not always correspond to malware. For example, one outlier might be a crawler program whose behavior is totally different from that of a real user. To assess the appropriateness of our assumption that the malicious NDN flows tend to be categorized as outliers to achieve higher information leakage throughput, we provide Figs. 11a and 11b to show four metrics of the outliers and inliers about *Properties 1, 2, 3,* and *4* that we described in Section 5. From the results, three types of malware which we defined in Section 5 can be approximated by the outliers.

In Table 7, the minimum value of T is 60 s, which means that it takes 60 s to detect the malware. Since the malware can generate and send out a lot of Interests within 60 s, it might leak all information before detected. One of the naive countermeasures is to limit the number of Interest packets sent within T. However, it is possible for this kind of activity to be legitimate (e.g., live streaming which might generate many Interests continuously). In this case, in order to avoid misdetection, acceptable name prefixes of such an application should be manually inserted into a whitelist of the firewall.

The information leakage throughput choked by the name and flow filter can be reduced to 23.2 Kbps. In other words, by mimicking legitimate user's behavior, it is possible for malware to leak data very slowly from an enterprise. In the case that the malware can continue to hide in the enterprise and the attacker does not care about the low throughput, the attacker can retrieve any kinds of a large file after waiting for a long time. Indeed, this kind of case happens on the Internet and such an attack is called Advanced Persistent Threat (APT) [45], which is a type of targeted attack. One of the features of APT is a "low and slow" approach to avoid detection. Therefore, considering not only targeted attack but also APT, besides the name and flow filter, researching more advanced countermeasures should be needed, which is an interesting future

direction.

9. Conclusion

NDN has the potential to create a more secure future Internet. Before deploying it on a large scale, it is crucial to investigate its vulnerabilities in order to make it safer against information leakage attacks. Previous work suggested the use of a name filter to decrease the information leakage throughput per Interest. A disadvantage of the name filter, however, is that it does not consider a flow of Interests. This paper assessed the risk of information leakage in NDN while taking malicious NDN flows that cause information leakage into account. To do so, we first proposed a conversion method to obtain the corresponding NDN flows derived from HTTP traces. Then, we presented properties of malware producing malicious NDN flows and proposed an SVM-based flow filter against such malicious flows, which can increase the efficiency of the name filter by two orders of magnitude. This shows that a flow filter can drastically reduce the impact of this security threat for NDN. We further outlined some limitations of this work as well as potential remedies. Future work in this area should focus on addressing these limitations and create a more robust and flexible filter.

Appendix A. Statistics of NDN Subflows

Figs. A.12 and A.13 describe cumulative distribution functions (CDFs) of the number of sequenced Interests in one NDN subflow generated by a manifest Interest and those of data size of payload Interests in one NDN subflow generated by a manifest payload Interest on the Jun/8–9, Jun/13–14, and Jul/13–14 dataset, respectively. According to Fig. A.12, the presence or absence of the outliers does not have impact on the statistic, and also the change of the datasets (i.e., date) does not affect it, since retrieving content from the producer does not depend on whether the consumer is an inlier or an outlier. The outliers have impact on the number of NDN subflows generated by a manifest Interest as shown in Table 6.

On the other hand, according to Fig. A.13, the presence or absence of the outliers greatly have impact on the statistic while the change of the datasets also affects it. For example, near the point where the data size is around 300 bytes, the CDFs including the outliers show steepness, and that might be caused by a program often sending the data whose size is fixed. Moreover, the data size depends on the consumer (e.g., uploading content from the consumer), and that has impact on the CDFs.

From Fig. A.12, around 82% of the NDN subflows include only one sequenced Interest. Assuming that the data size of a payload Interest is less than 8800 bytes, Fig. A.13 shows that most of the NDN subflows consist of only one payload Interest.

Fig. A.14 shows the CDFs of time interval of both a manifest Interest and a manifest payload Interest (Manifest Interest + Manifest payload Interest), only a manifest Interest (Manifest Interest), and only a manifest payload Interest (Manifest payload Interest) on the Jun/8–9, Jun/13–14, and Jul/13–14 dataset. Note that Fig. A.14 does not take account of the HTTP flow aggregation into the NDN flow in Fig. 5, and Fig. A.14 is introduced by obtaining the NDN flows corresponding to each of the HTTP flows and calculating the time interval in each of the NDN flows. According to Figs. A.14a, A.14b, and A.14c, regardless of the presence or absence of the outliers, the CDFs of time interval of both a manifest Interest and a manifest payload Interest are quite similar to the ones of only a manifest Interest. It is because the number of samples about time interval of a manifest Interest is around from 23 to 41 times larger than that of a manifest payload Interest. As for the CDFs of time interval of only a manifest payload Interest, which include the outliers, they indicate steepness around near the point where the time interval is around 2.25 sec, and that comes from the consumer which continuously sends a manifest payload Interest to the specific producer per around 2.25 sec. Moreover, regardless of the presence or absence of the outliers, the time interval of a manifest Interest is shorter than the one of a manifest payload Interest.

Here, we consider the HTTP flow aggregation into the NDN flow in Fig. 5. From Fig. A.14, around from 95 to 99% of the time intervals of a manifest Interest and a manifest payload Interest are less than 100 sec, and we set 100 sec to T_{thr}. Then, Fig. A.15 can be obtained by applying the HTTP flow aggregation to Fig. A.14. According to Figs. A.15a, A.15b, and A.15c, by the aggregation, the time interval becomes shorter than the one shown in Fig. A.14.

Appendix B. HTTP Traffic Using HTTP Pipelining in Waikato Traces

RFC 7230 [46] officially supports HTTP pipelining in HTTP/1.1 traffic, which allows a browser to send multiple request messages without waiting for the corresponding response messages. However, according to "HTTP Pipelining - Big in Mobile" [47], most major browsers did not support HTTP pipelining (or deactivated it by default). Only Opera for desktop and mobile, and Android for mobile employed HTTP pipelining in 2011. StatCounter [48] statistics on browser usage share in June 2011 showed that the share of Opera in desktop was 1.74% while the share of Opera and Android in mobile was 40.1%. As for the traffic volume of desktop and mobile in June 2011, 93.5% of all traffic came from desktop and 6.53% was mobile traffic. Thus, the ratio of HTTP traffic using HTTP pipelining to all the HTTP traffic in the Waikato HTTP traces we utilized should be approximately 4.25% ($= 93.5 \times 0.0174 + 6.53 \times 0.401$). Accordingly, we assumed

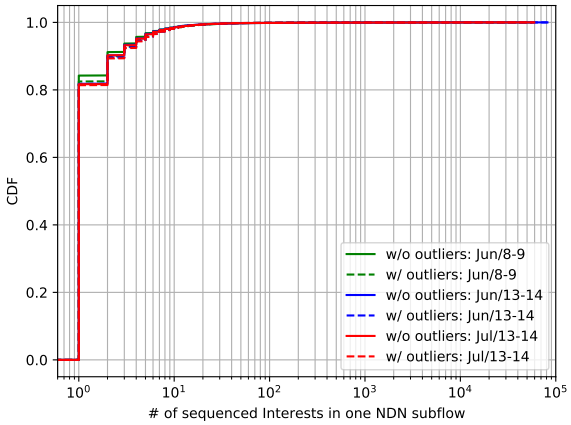


Figure A.12: CDF of number of sequenced Interests in one NDN subflow.

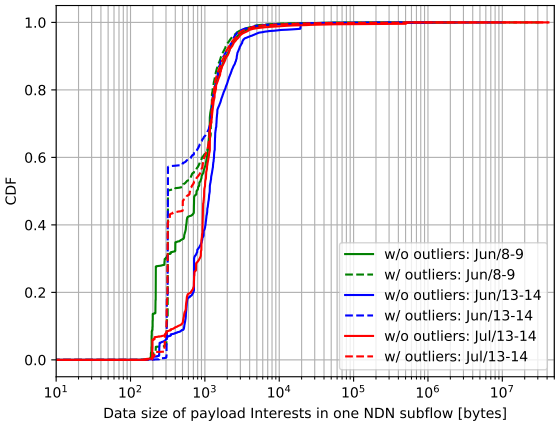
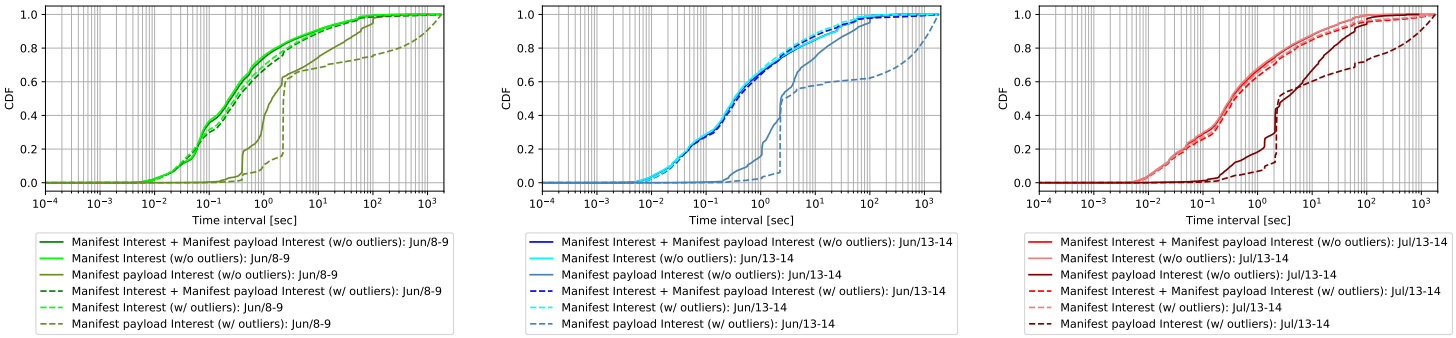


Figure A.13: CDF of data size of payload Interests in one NDN subflow [bytes].



(a) Jun/8–9. (b) Jun/13–14. (c) Jul/13–14.

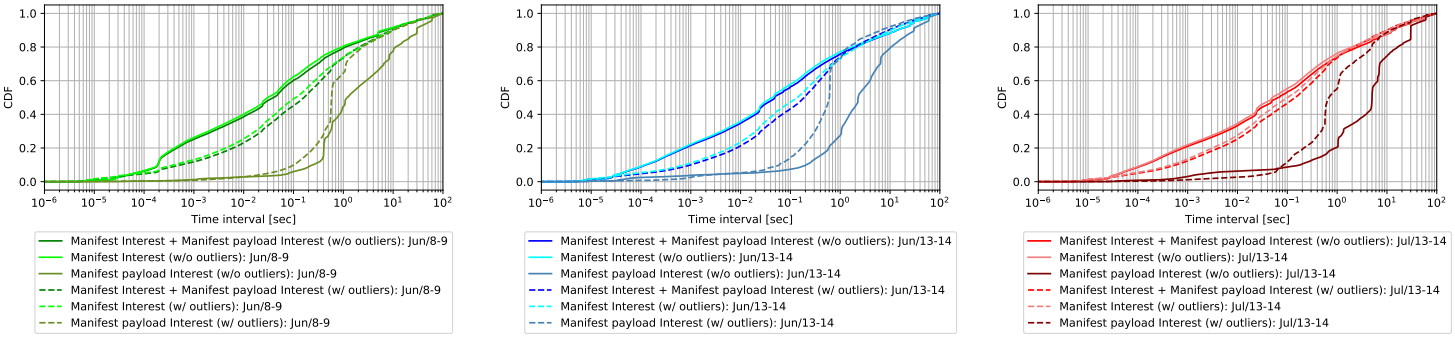
Figure A.14: CDF of time interval [sec]. In Fig. A.14b, the graph "Manifest Interest + Manifest payload Interest (w/o outliers): Jun/13–14" almost completely overlaps with the graph "Manifest Interest (w/o outliers): Jun/13–14".

that the HTTP traffic using HTTP pipelining was negligible and the traces consisted of HTTP traffic without HTTP pipelining.

Appendix C. Metrics

The metrics that we use are calculated based on the numbers of *true positives* (TP), i.e., correctly predicted values of the corresponding (legitimate or anomalous) class, *false positives* (FP), i.e., wrongly predicted values, e.g., when predicting the positive class when the actual class is the negative one ("positive" could mean either legitimate or anomalous depending on the use, and correspondingly for "negative"), and *false negatives* (FN), i.e., wrongly predicted values, e.g., when predicting the negative class when the actual class is the positive one. The metrics are: Precision (P), Recall (R) and F_1 score calculated as follows:

- $P = TP/(TP + FP)$. It is the ratio of correctly predicted observations of the corresponding class (legitimate or anomalous) to the total *predicted* observations of that class. When considering for example the precision of the legitimate class, the question this metric answers is: of all activities labeled as legitimate, how many of them are actually legitimate? The same holds for the anomalous class.
- $R = TP/(TP + FN)$. It is the ratio of correctly predicted observations of the corresponding class (legitimate or anomalous) to the total *actual* observations of that class. When considering for example the recall of the legitimate class, the question this metric answers is: of all legitimate activities, how many of them did we label as legitimate? The same holds for the anomalous class.
- $F_1 = 2 \times (R \times P)/(R + P)$. It is the weighted average of Precision and Recall, which takes both FP and FN into account.



(a) Jun/8–9. (b) Jun/13–14. (c) Jul/13–14.

Figure A.15: CDF of time interval considering HTTP flow aggregation to NDN flow [sec].

Appendix D. Parameter Settings

For all machine learning models (isolation forests, isomap, SVMs) we use scikit-learn [42] with the default parameters. The feature vectors are preprocessed using z-score normalization. For isolation forests we use two outlier proportions: $\{0.005, 0.01\}$. The number of consumers (l) is 767 for Jun/8-9, 760 for Jun/13-14 and 908 for Jul/13-14 (shown in Table 3). 70% of the Jun/8-9 dataset was used for training and validation, using 5-fold cross-validation ; the remaining 30% of the Jun/8-9 dataset, as well as the 100% of the other datasets were used as test sets. For the grid search on the hyperparameters of the SVM (based on the validation fold) we use: Kernel: [RBF], γ: [10^0, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}], C: [10^0, 10^1, 10^2, 10^3, 10^4]. T and m are set for each of the 7 cases as in Table 7.

Acknowledgments

We thank the anonymous reviewers for their constructive comments, as well as Olivier Perrin. This work is supported by the DOCTOR Project and funded by French National Research Agency (ANR-14-CE28-0001). Also, this work is partly supported by the JSPS KAKENHI grant number JP17H00734. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 739578, complemented by the Government of the Republic of Cyprus through the Directorate General for European Programmes, Coordination and Development.

[1] S. Gupta and B. B. Gupta, "Detection, Avoidance, and Attack Pattern Mechanisms in Modern Web Application Vulnerabilities: Present and Future Challenges," *IJCAC*, vol. 7, no. 3, pp. 1–43, Jul. 2017.

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

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

[4] 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.

[5] 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.

[6] D. Kondo, T. Silverston, H. Tode, T. Asami, and O. Perrin, "Name Anomaly Detection for ICN," in *IEEE LANMAN* 2016.

[7] 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.

[8] D. Kondo, T. Silverston, V. Vassiliades, H. Tode, and T. Asami, "Name Filter: A Countermeasure against Information Leakage Attacks in Named Data Networking," *IEEE Access*, vol. 6, pp. 65151–65170, Oct. 2018.

[9] Information-Centric Networking Research Group (ICNRG), `https://trac.ietf.org/trac/irtf/wiki/icnrg`

[10] WITS: Waikato VIII, `https://wand.net.nz/wits/waikato/8/`

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

[12] C. Cortes and V. Vapnik, "Support-Vector Networks," in *Machine Learning*, vol. 20, no. 3, pp. 273–297, Sep. 1995.

[13] DNS Tunneling in the Wild: Overview of OilRigs DNS Tunneling, `https://unit42.paloaltonetworks.com/dns-tunneling-in-the-wild-overview-of-oilrigs-dns-tunneling/`

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

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

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

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

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

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

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

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

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

[23] 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.
[24] Morto Worm Sets a (DNS) Record, https://www.symantec.com/connect/blogs/morto-worm-sets-dns-record
[25] 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.
[26] D. Goergen, T. Cholez, J. Francois, and T. Engel, "A Semantic Firewall for Content-Centric Networking," in *IFIP/IEEE IM Mini-Conference* 2013.
[27] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *ACM CoNEXT* 2009.
[28] L. Popa, A. Ghodsi, and I. Stoica, "HTTP As the Narrow Waist of the Future Internet," in *HotNets* 2010.
[29] H. Yuan and P. Crowley, "Experimental Evaluation of Content Distribution with NDN and HTTP," in *IEEE INFOCOM* 2013
[30] X. Marchal, M. E. Aoun, B. Mathieu, W. Mallouli, T. Cholez, G. Doyen, P. Truong, A. Ploix, and E. M. de Oca, "A Virtualized and Monitored NDN Infrastructure Featuring a NDN/HTTP Gateway," in *ACM ICN* 2016.
[31] H. Yuan and P. Crowley, "Scalable Pending Interest Table Design: From Principles to Practice," in *IEEE INFOCOM* 2014.
[32] Flow Classification in Information Centric Networking, https://tools.ietf.org/html/draft-moiseenko-icnrg-flowclass-02
[33] ndn-cxx, https://github.com/named-data/ndn-cxx/blob/master/src/encoding/tlv.hpp
[34] Fetching Content in Named Data Networking with Embedded Manifests, https://named-data.net/wp-content/uploads/2014/09/ndn-tr-25-manifest-embedding.pdf
[35] Signed Interest, http://named-data.net/doc/ndn-cxx/current/specs/signed-interest.html
[36] I. Moiseenko, M. Stapp, and D. Oran, "Communication Patterns for Web Interaction in Named Data Networking," in *ACM ICN* 2014.
[37] S. S. Adhatarao, J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "ORICE: An Architecture for Object Resolution Services in Information-Centric Environment," in *IEEE LANMAN* 2015.
[38] S. S. Adhatarao, J. Chen, M. Arumaithurai, X. Fu, and K. K. Ramakrishnan, "Prototype of an Architecture for Object Resolution Services in Information-Centric Environment," in *ACM ICN* 2015.
[39] K. Schneider, C. Yi, B. Zhang, and L. Zhang, "A Practical Congestion Control Scheme for Named Data Networking," in *ACM ICN* 2016.
[40] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The Cost of the "S" in HTTPS," in *ACM CoNEXT* 2014.
[41] pkt2flow, https://github.com/caesar0301/pkt2flow
[42] F. Pedregosa, et al. "Scikit-learn: Machine Learning in Python," *Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[43] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A Global Geometric Framework for Nonlinear Dimensionality Reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, Dec. 2000.
[44] S. Haykin, "Neural Networks and Learning Machines," *Pearson*, Upper Saddle River, NJ, USA, 2008.
[45] Advanced Persistent Threats: A Symantec Perspective, https://www.symantec.com/content/en/us/enterprise/white_papers/b-advanced_persistent_threats_WP_21215957.en-us.pdf
[46] RFC 7230, https://tools.ietf.org/html/rfc7230
[47] HTTP Pipelining - Big in Mobile, http://www.guypo.com/http-pipelining-big-in-mobile/
[48] StatCounter, http://gs.statcounter.com/