electronics

MDPI

Article

Data Exfiltration Detection on Network Metadata with Autoencoders

Daan Willems [1,*], Katharina Kohls [2], Bob van der Kamp [1] and Harald Vranken [2,3]

[1] National Cyber Security Centre (NCSC), Ministry of Justice and Security, 2511 DP The Hague, The Netherlands; bob.vanderkamp@ncsc.nl
[2] Institute for Computing and Information Sciences, Faculty of Science, Radboud University, 6525 AJ Nijmegen, The Netherlands; kkohls@cs.ru.nl (K.K.); harald.vranken@ou.nl (H.V.)
[3] Faculty of Science, Department of Computer Science, Open Universiteit in The Netherlands, 6419 AT Heerlen, The Netherlands
* Correspondence: daan.willems@ncsc.nl

Abstract: We designed a Network Exfiltration Detection System (NEDS) to detect data exfiltration as occurring in ransomware attacks. The NEDS operates on aggregated metadata, which is more privacy-friendly and allows analysis of large volumes of high-speed network traffic. The NEDS aggregates metadata from multiple, sequential sessions between pairs of hosts in a network, which captures exfiltration by both stateful and stateless protocols. The aggregated metadata include averages per session of both packet count, request entropy, duration, and payload size, as well as the average time between sequential sessions and the amount of aggregated sessions. The NEDS applies a number of autoencoder models with unsupervised learning to detect anomalies, where each autoencoder model targets different protocols. We trained the autoencoder models with real-life data collected at network sensors in the National Detection Network as operated by the National Cyber Security Centre in the Netherlands, and configured the detection threshold by varying the false positive rate. We evaluated the detection performance by injecting exfiltration over different channels, including DNS tunnels and uploads to FTP servers, web servers, and cloud storage. Our experimental results show that aggregation significantly increases detection performance of exfiltration that happens over longer time, most notably, DNS tunnels. Our NEDS can be applied to detect exfiltration either in near-real-time data analysis with limited false positive rates, or in captured data to aid in post-incident analysis.

Keywords: network intrusion detection; data exfiltration; autoencoder; anomaly detection

Citation: Willems, D.; Kohls, K.; van der Kamp, B.; Vranken, H. Data Exfiltration Detection on Network Metadata with Autoencoders. *Electronics* **2023**, *12*, 2584. https://doi.org/10.3390/electronics12122584

Academic Editor: Paulo Ferreira

Received: 27 April 2023
Revised: 25 May 2023
Accepted: 2 June 2023
Published: 8 June 2023

1. Introduction

Ransomware is an ongoing and increasing problem worldwide. Additionally, in the Netherlands, the use of ransomware is considered a national security risk, since it threatens the continuity of vital processes, the leaking and/or publication of confidential or sensitive information, and the deterioration of cyberspace integrity [1]. Next to the encryption of files, ransomware attacks are used increasingly with double or even triple extortion, when cybercriminals steal data and threaten to publish these if the affected organisation or its customers, suppliers, or partners do not pay a ransom [1]. Due to the strong financial incentive, cybercriminals are motivated to continuously improve their methods and techniques.

In this paper, we focus on the detection of data exfiltration as occurring in ransomware attacks, where data are stolen covertly [2]. With good cybersecurity practices becoming more prevalent, the threat of data exfiltration is becoming even greater than the threat of data encryption, since organisations with effective backup systems may recover from encrypted data rather quickly while the theft of data can hardly be undone and it is difficult to impossible to prevent stolen data from being spread or sold on the internet. Data

exfiltration enables cybercriminals to exercise more leverage on their victims by threatening to publish or sell stolen data to interested parties. Hence, there is a pressing need to quickly detect and stop data exfiltration by live analysis, and to identify by post-incident analysis what data have been stolen and through which ways.

To increase cybersecurity, including resilience against ransomware threats, the National Cyber Security Centre (NCSC) in the Netherlands has created the National Detection Network (NDN), a platform for information sharing and automated detection. The NDN receives intelligence from diverse sources and uses this to inform participating members on cyber threats. The NDN also includes a large array of sensors that monitor network traffic at central government organisations. Due to the large quantity of monitored traffic data, automated filtering and analysis is required. The NCSC currently relies on threat intelligence acquired from vendors as Indicators of Compromise (IOCs) for automated detection of suspicious traffic. Although it is highly reliable, a limitation is that IOCs are limited to known threats, while zero-day attacks may go unnoticed.

In this paper, we propose a detection system for data exfiltration that relies on anomaly detection to detect known attacks and potentially also zero-day attacks. We leverage that the NDN provides a unique source of real-life network data that are readily available, and particularly that the NDN provides aggregated network data in the form of session metadata. The contributions of this paper are as follows:

- We design and test a novel Network Exfiltration Detection System (NEDS) that can detect instances of data exfiltration as occurring in ransomware attacks. The NEDS analyses network traffic and can detect anomalies in this traffic without relying on specific threat intelligence. The NEDS is composed of an ensemble of autoencoders, where each autoencoder is targeted at one or multiple network protocols;
- A key novelty of our NEDS is that it operates on aggregated network metadata from multiple, sequential sessions. This allows to detect data exfiltration that happens over a longer period of time, by either stateless or stateful protocols. The usage of aggregated metadata also allows to deal efficiently with large amounts of network sessions. Hence, the NEDS can be applied in practical settings for either near real-time analysis of live data or in post incident analysis of captured data;
- We train the NEDS using unsupervised learning with real-life data from the NDN sensor platform (NSP) in the Netherlands. We evaluate the detection performance of the NEDS for data exfiltration over different channels, including DNS tunnels and uploads to FTP servers, web servers, and cloud storage. Our experimental results demonstrate that the usage of aggregated metadata significantly increases detection performance of exfiltration with limited false positive rates.

In the remainder of this paper, we first give background information on data exfiltration and autoencoders in Section 2. We discuss related work in Section 3. The design and architecture of the NEDS are presented in Section 4. Our experimental setup to evaluate the detection performance of the NEDS is described in Section 5. We present and discuss our experimental results in Section 6. Section 7 concludes the paper, in which we also give directions for future research.

2. Background

In this section, we provide background information on different types of data exfiltration and give a basic introduction to autoencoders.

2.1. Data Exfiltration

Data exfiltration is a security breach where data of an individual or organisation is copied or moved without authorization. It typically includes transferring chunks of data over a Command & Control (C2) channel or an alternative channel. The MITRE ATTA&CK framework (https://attack.mitre.org (accessed on 1 November 2021)), an established knowledge base of adversary tactics and techniques based on real-world observations, lists nine exfiltration techniques for getting data out of a target network: automated exfiltration

(T1020), data transfer size limits (T1030), exfiltration over alternative protocol (T1048), exfiltration over C2 channel (T1041), exfiltration over other network medium (T1011), exfiltration over physical medium (T1029), exfiltration over web service (T1567), scheduled transfer (T1029), and transfer data to cloud account (T1537). Typically, multiple of these techniques are applied together during ransomware attacks. In this paper, we consider all of these techniques, except exfiltration over other network medium (T1011) and physical medium (T1029). We consider exfiltration through different channels, including DNS tunnels and uploads to FTP servers, web servers, and cloud storage.

In DNS tunneling, an attacker registers a domain name and configures a server that is under their control as the authoritative name server for this domain name. After installing malware on a victim's host, the malware can send DNS requests for resolving the domain name and embed additional information in the DNS packets. The DNS protocol ensures these packets end up at the authoritative name server that is under control of the attacker, where the attacker can retrieve the embedded information. Data can be embedded in multiple record types in DNS packets. DNS tunnels are rarely blocked by firewalls, as most networks require DNS, which increases the chance that data exfiltration would be successful. Depending on the type of tunnel and rate of transfer, DNS tunnels can be very hard to detect [3].

2.2. Autoencoders

An autoencoder is a specific type of artificial neural network that is typically composed of two parts, as shown in Figure 1: an encoder, which encodes the input x into an internal representation $z = e(x)$, and a decoder, which tries to reconstruct the original input from the internal representation and outputs $x' = d(z)$. The internal representation z typically is smaller than the original input x. The output x' should resemble the input x as closely as possible. During training, the autoencoder tries to minimize the loss $||x - x'||^2$.

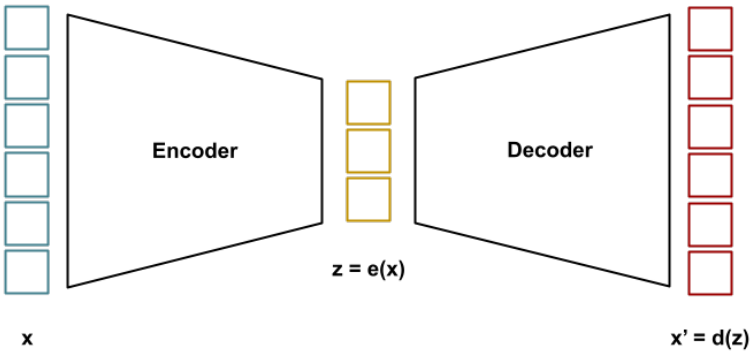

Figure 1. Autoencoder architecture.

An autoencoder with unsupervised learning can be applied in anomaly detection. The autoencoder is trained first using a training dataset that does not contain anomalies. The trained autoencoder can next be fed with a test dataset and detect anomalies by considering that the autoencoder will perform worse at reconstructing inputs that are different from the inputs that it has been trained on [4].

3. Related Work

There is a vast amount of scientific literature on data exfiltration, which is closely related to the broader fields of detecting malicious or anomalous network traffic by means of data-driven and behavior-driven approaches and traffic classification [5].

In this section, we discuss prior work in these fields, with a focus on data exfiltration. In Section 3.1, we briefly outline how network data have been used, and in Section 3.2—

what methods have been applied. In Section 3.3, we detail how our work relates to prior work.

3.1. Traffic Inspection

A common method for detecting malicious or anomalous traffic is by inspecting the traffic contents. Examples are Deep Packet Inspection (DPI) [6], tracking file system access operations on data piped from database to files [7], or signature-based detection [8]. However, detecting data exfiltration through content inspection can be challenging due to high computing cost, and is even impossible when exfiltrated traffic is encrypted or engineered to look similar to normal network traffic [5].

Fawcett [9] addressed the detection of encrypted data exfiltration by considering the Shannon entropy of outgoing network traffic. The presence of encrypted traffic on what are normally unencrypted channels can be a strong indicator of data exfiltration. However, computing entropy over network traffic incurs a computing cost, and a minimum payload size of 256 bytes is required to confidently identify encrypted traffic using entropy. He et al. [10] recorded and counted communication events to model normal behavior and to determine whether traffic is encrypted.

Some studies considered aggregated data for detecting data exfiltration. Nadler et al. [11] and Haghighat et al. [12] used a sliding window approach for detecting DNS tunnels. Mirsky et al. [13] proposed an efficient method to accumulate packets and keep track of averages. Kemp et al. [14] used netflow data, while Najafabadi et al. [15] and Nadler et al. [11] aggregated flows over a longer time to expose low-throughput traffic.

3.2. Methods

Researchers have explored numerous methods to detect anomalies in network traffic or data exfiltration. These methods range from automated methods applying wavelet analysis [16] to graph theory [17], basic machine learning [14,18], and deep learning [12]. Clustering [13,19–23] and autoencoders [4,13,24–28] are popular methods, since they can detect patterns in data using unsupervised learning and hence do not require labelled training data.

Clustering has been widely applied for network anomaly detection [19,20]. With clustering, anomalous data typically appear as outliers that have higher distance to centroids of clusters or form smaller clusters. Münz et al. [21] applied K-means clustering using the numbers of packets, bytes, and source-destination ports as features. Liu et al. [20] combined K-means clustering with Principal Component Analysis (PCA) for dimensionality reduction. Pagliari et al. [22] used bi-clustering combined with Support Vector Machine (SVM). Mao et al. [29] proposed a clustering-based feature selection mechanism, and also Radhakrishnan et al. [23] and Mirsky et al. [13] applied clustering as a pre-processing step to reduce the number of dimensions and to eliminate redundant features.

Autoencoders generally can handle high-dimensional data better than traditional clustering algorithms such as K-means. Autoencoders can also be applied to compress input data before feeding them into a clustering algorithm. Nixon et al. [24] evaluated the use of autoencoders for real-time anomaly detection in network traffic. Xu et al. [25] built a five-layer architecture with autoencoders and a data pre-processing step to remove outliers. Chen et al. [26] proposed an architecture with an ensemble layer of autoencoders in which each autoencoder receives a part of the dataset during training while randomly dropping connections in each autoencoder. Mirsky et al. [13] also applied an autoencoder ensemble in which each autoencoder uses a different set of related features. Chen et al. [4] built a convolutional autoencoder for network traffic anomaly detection that uses a two-dimensional data representation which requires less training time. Nguyen et al. [27] considered variational autoencoders that are more robust against noisy data and improve the explainability of results. Wu et al. [28] experimented with multiple types of autoencoders, including basic, variational, deep, and sparse autoencoder to detect DNS tunnels, and obtained best results with basic autoencoders.

3.3. Rationale

In this paper, we combine multiple concepts from related work as discussed above, such as using aggregated data and an ensemble of autoencoders, but we made notable modifications and extensions.

We apply the similar concept of data aggregation as introduced by Mirsky et al. [13], but we apply this on the level of sessions rather than packets. We apply aggregation for a similar goal, as introduced by Nadler et al. [11], but we aggregate all network traffic rather than DNS traffic only. We adopt the approach of only looking at outgoing data and include the concept of entropy into our feature list to take unexpected encrypted content into account, in a similar way as proposed by Fawcett [9], but we also consider additional features of the aggregated metadata. During aggregation, we apply a similar decay function from damped incremental statistics as the one introduced by Mirsky et al. [13], but we only have to keep track of a single value per feature, resulting in a linear time complexity.

Instead of a fairly complex setup, we use a basic autoencoder based on the work by Wu et al. [28]. We apply an ensemble layer of basic autoencoders, as introduced by Mirsky et al. [13] and Chen et al. [4], but instead of clustering parts of the dataset or features per autoencoder, we assign each autoencoder to a different application-level network protocol with the goal of preventing protocols with larger dimensions from overshadowing smaller protocols.

4. NEDS Design

In this section, we present the design of our NEDS.

4.1. Architecture

The architecture of our NEDS consists of three components: data collection, data aggregation, and anomaly detection. As shown in Figure 2, the data collector receives metadata from the sensors in the NPS that are based on the RSA NetWitness platform (www.netwitness.com (accessed on 1 August 2021)). Next, the data aggregator combines metadata on sessions, which enables detection of low-throughput communication. We store the aggregated data to create a dataset that can be used to train the autoencoders in the anomaly detector. Storing the data allows us to easily reuse data, for instance when updating or extending the anomaly detector. After training, the trained autoencoders can be used to evaluate live or previously captured network data. When running the anomaly detector, it tries to reconstruct the input data, which will not succeed if the data differ significantly from the data that they were trained on, which can reveal data exfiltration.

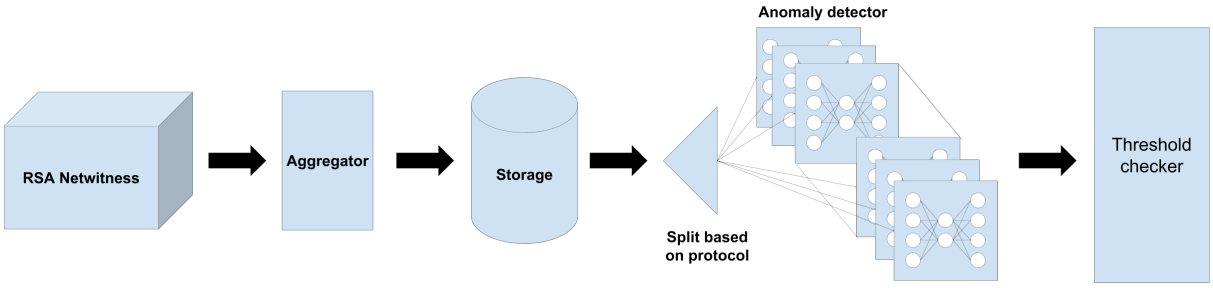

Figure 2. NEDS architecture.

4.2. Session Metadata

The NSP sensors store both raw packet captures in PCAP-files and session metadata. A network session is defined as one or two related stream(s) of traffic with a requester and, usually, a responder. For instance: a DNS session consists of a single DNS request and/or DNS response; a UDP session consists of packets assembled from the same source and destination IP address/port pairs within preconfigured limits, such as the session size or a timeout period since the last packet in the session; a TCP session consists of packets

assembled from the same source and destination IP address/port pairs and TCP flags within preconfigured limits.

The RSA Netwitness platform provides rich session metadata that contain fields such as:

- Hosts: the two hosts that are communicating in the session.
- Session size: the total amount of bytes transmitted both ways.
- Lifetime: the total number of seconds the connection was alive.
- Timestamp: the time at which the connection was initiated.
- Payload: the total amount of bytes successfully transmitted both ways.
- Total transmitted bytes of request and response: the total amount of bytes that were transmitted, including dropped packets.
- Direction of traffic: indicates whether traffic is inbound, lateral, or outbound.
- Packet count of session: total packet count of session.
- Transport level protocol: TCP or UDP.
- Service: DNS, HTTP, etc.
- Destination organisation: indicates whether the destination is linked to an organisation.
- Entropy of request and response payload: entropy of transmitted bytes.

Although metadata in the session format do not contain the content of packets as in PCAP-files, it offers several advantages. Storing metadata requires far less storage space, and therefore can be stored for longer periods. The metadata provide aggregated data that can be used more easily for analysis of large volumes of high-speed network traffic, and it is also more privacy-friendly. Our NEDS, therefore, operates on session metadata. Since we focus on the detection of data exfiltration, we only consider traffic that is part of the outbound connection and we discard incoming and lateral traffic.

4.3. Aggregated Session Metadata

We introduce data aggregation for detecting data exfiltration effectively. Exfiltration may occur in high-throughput communication, which can be identified by unexpected sessions with spikes in traffic volume, but it can also occur in low-throughput communication that is spread over multiple sessions. Session metadata already aggregates data at the level of individual sessions, which is effective for detecting communication over stateful protocols such as TCP, but not for protocols such as UDP or DNS that result in short sessions. Furthermore, sessions that grow too long or too big will be split. Hence, data exfiltration patterns may not appear in individual sessions, but instead are spread over multiple sessions, which is, for instance, the case with DNS tunnelling. We therefore aggregate session metadata from multiple, sequential sessions.

We aggregate session metadata based on aggregation keys, which identifies the two hosts that are communicating over multiple, sequential sessions. An aggregation key combines the source IP address, destination IP address, and protocol. Figure 3 shows an example with four hosts. Session 1 and 2 are between the same two hosts and are aggregated; session 3 and 4 are between different hosts and hence are not aggregated.

Our aggregated metadata contains averages of features that relate to sessions under the same aggregation key. We considered the following six features:

1. **Average packet count per session:** This feature represents the average number of packets per session. It aggregates the *packet count* metadata field. If a malicious actor is actively exfiltrating data, this feature value may possibly be higher than normal. However, this can also be due to a non-malicious actor uploading information to an external host. For some protocols where the number of packets is usually small and consistent, such as DNS, this feature can highlight anomalous behaviour.
2. **Average request entropy per session:** This feature represents the average entropy of the payloads in outgoing packets per session. It aggregates the *entropy.req* metadata field. Since our main objective is to detect exfiltration, we only consider outgoing traffic and we ignore response entropy and total entropy. Entropy roughly resembles the amount of information the payload contains. A high request entropy may indicate

exfiltration, as more data are being moved. A high request entropy can also be an indicator of encrypted traffic, which an attacker may use when tunnelling over an usually unencrypted protocol [10].

3. **Average session duration:** This feature represents the average duration (i.e., lifetime) of a session in seconds. It aggregates the *lifetime* metadata field. The duration of a session combined with other features can be indicative of anomalous behaviour. For instance, sessions with DNS tunnelling have a much longer duration compared with normal DNS traffic.
4. **Average session payload size:** This feature represents the average total session payload size in bytes. It aggregates the *payload* metadata field. Note that the average session payload size might be small even in long sessions due to packet drops. A high payload size may indicate that a lot of data is being transported, and hence this feature can potentially detect naïve data exfiltration. However, the feature will not reveal a smart adversary who uploads data in low quantities over longer time periods.
5. **Average time between sequential sessions:** This feature represents the average time between sessions in seconds. It aggregates the differences between *timestamp* metadata fields. It is potentially useful for identifying automated behaviour, since a user will likely not generate many hundreds of DNS or HTTP requests per second.
6. **Weight:** This feature represents the total amount of sessions aggregated under the key. This feature is used for computing the averages of the features above. It also indicates the difference between a single large session and multiple smaller sessions. A single session which contains a large payload will have a weight of 1, while 100 sessions transporting a total payload that is 100 times as big will have an identical average payload size but a greater weight value.

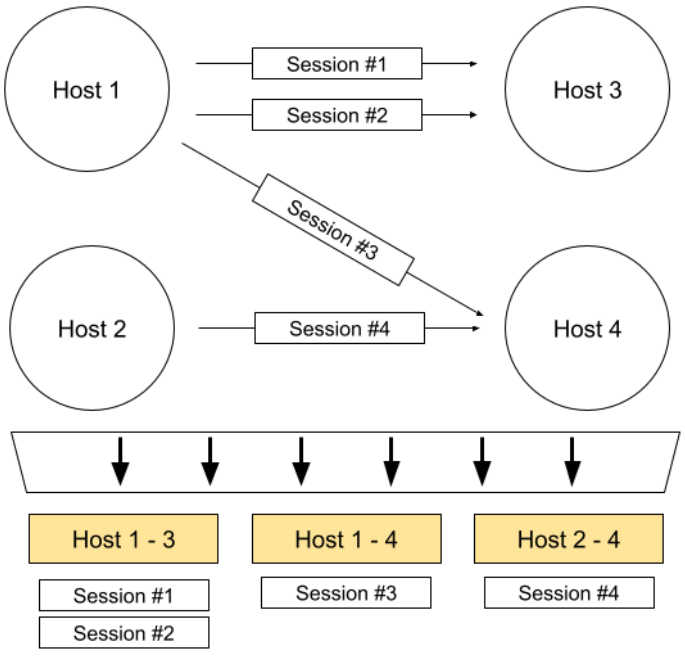

Figure 3. Data aggregation example.

Our NEDS retrieves the relevant metadata fields through the REST API of the sensor environment at regular time intervals, using the following query:

select entropy.req, sessionid, lifetime, time, service, ip.src.hash, ip.dst.hash, ipv6.src.hash, ipv6.dst.hash, packets, payload.req
where direction = 'outbound' **and** time = "-"

We implemented aggregation using the decay function from the damped incremental statistics framework as described by Mirsky et al. [13], with a few simplifications as listed

in Algorithm 1. We query the sensor platform to retrieve session metadata for the outbound traffic in a certain time interval (which is stored in *sessions*). For each of the retrieved sessions, we derive the key k from the source and destination IP addresses and the protocol. We aggregate the packet count PC, the duration D, the payload size P, and the entropy EQ of sessions that have the same key. We aggregate data by accumulating metadata of sessions that have the same key, using the damped window model in which older values are exponentially decreased over time. We apply decay function $2^{-\lambda \cdot dif}$, where λ is the decay factor that determines how quickly values decay and dif is the time difference between the current session and the previous session. We store the updated aggregated metadata $aggregate[k]$, as well as the average time between sequential sessions ($average\_time[k]$) and the total amount of session ($weight[k]$) under key k, in the storage system.

The time complexity of Algorithm 1 is $\mathcal{O}(1)$. We retrieve session metadata from the sensor environment by querying its REST API at regular time intervals (outer while-loop), and we process the metadata for each session sequentially (inner for-loop).

Algorithm 1 Querying and aggregation.

while $(current\_time < end\_time)$ **do**
  $sessions \leftarrow query(current\_time - interval,\ current\_time)$
  for $(session$ **in** $sessions)$ **do**
    $k \leftarrow < session.srcIP,\ session.dstIP,\ session.service >$
    $v \leftarrow < session.PC,\ session.D,\ session.P,\ session.EQ >$
    $dif \leftarrow session.timestamp - aggregate[k].timestamp$
    $aggregate[k] \leftarrow aggregate[k] \cdot 2^{-\lambda \cdot dif} + v$
    $store\ < aggregate[k],\ average\_time[k],\ weigth[k] >$
  end for
  $current\_time \leftarrow current\_time + interval$
end while

4.4. Storage

We store aggregated data in order to create a dataset that can be used to train the anomaly detector (see Figure 2). We apply a database system to handle large amounts of data without performance issues. We used Redis (redis.io) to implement our database structure because of the flat structure of our data and the requirement for fast read (and write). Since our anomaly detector applies separate autoencoders for different application-level protocols, we split the aggregated data based on the application-level protocol. We create separated databases in the Redis instance for each application-level protocol, with an additional database to store the state of the aggregation algorithm.

The time and space complexity for querying the REST API of the sensor environment and storing the retrieved session metadata is $\mathcal{O}(1)$. Our database implementation uses an in-memory, key-based storage model. For each unique key, a new unique key–value pair is created, and hence, the required space grows linearly with the amount of unique keys.

4.5. Anomaly Detector

4.5.1. Architecture

The anomaly detector is composed of an ensemble layer with multiple autoencoders and a Threshold Checker, as illustrated by the example in Figure 4. In the ensemble layer, a separate autoencoder is used for each application-level protocol (or multiple similar protocols). The anomaly detector receives a feature vector x of dimension $k = 6$ with aggregated session metadata, as described in Section 4.3. The feature vector is fed only to the autoencoder that matches the protocol as indicated by the aggregation key.

Each autoencoder is trained with the aggregated network data for the corresponding protocol. Depending on the use case, the user can configure how many autoencoders, and for which protocols, are included in the anomaly detector. Not all protocols appear on every sensor, primarily due to the way they are deployed in the network. We select

the protocols that are actually present, and assign each of them to an autoencoder. In case of protocols for which not enough traffic is present to reliably train an autoencoder, we combine these protocols.

As shown by Mirsky et al. [13], the complexity of executing a single autoencoder is $\mathcal{O}(k^2)$. As the number of the autoencoders can be configured by the user, the complexity of the anomaly detector scales linearly with this amount.

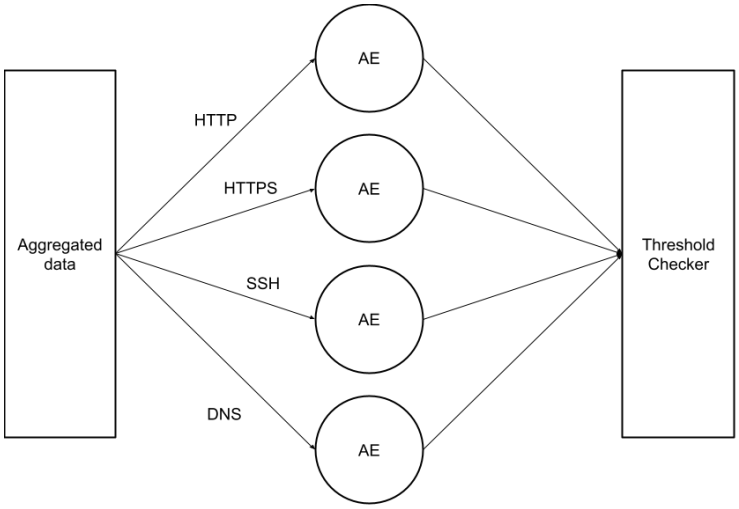

Figure 4. Anomaly detector architecture.

4.5.2. Autoencoder Model

We apply the autoencoder model in the anomaly detector instead of other machine-learning or deep-learning models for multiple reasons. Since we only have unlabelled data available for training, we rely on unsupervised learning. An autoencoder model is in principle well-suited to detect anomalies. The key idea behind this is that an autoencoder model that has been trained with normal traffic will do well at reconstructing normal traffic but poorly at reconstructing anomalies. We apply multiple autoencoders in parallel, instead of one central autoencoder, since this has several benefits. Each autoencoder targets a separate protocol, which allows us to use a relatively simple autoencoder model. The anomaly detector can be scaled easily by adding autoencoders and the autoencoders can be trained and run in parallel. This is hard to realize with other anomaly detection techniques such as clustering. Furthermore, using compact autoencoders facilitates explainability and diagnosis of decisions made by the anomaly detector.

We use a relatively simple autoencoder model, implemented with the PyTorch library. All autoencoders consist of: an encoder, containing a linear layer with six neurons and a ReLu layer; a bottleneck layer of two neurons; and a decoder, containing a linear layer with six neurons and a sigmoid layer. The difference between the input x and the reconstructed output y of the autoencoder model (i.e., the reconstruction error) is computed by the Mean Squared Error function [13]:

$$RMSE(x, y) = \sqrt{\frac{\Sigma_{i=1}^{k}(x_i - y_i)^2}{k}} \quad (1)$$

During training, we used the Adam optimizer with a learning rate of 0.001, a weight decay of 0.000001, and a batch size of 32. Furthermore, we shuffled the input order each epoch. We did not use dropout. We experimented with changing the number of neurons in the bottleneck layer. Although this drastically changes the reconstruction error, it only has a minor impact on the actual classification performance of the anomaly detector.

4.5.3. Threshold Checker

The reconstruction error of the autoencoder is compared to a threshold in the Threshold Checker. We configured a separate threshold for each autoencoder. If the error is higher than the threshold, the session is classified as an anomaly. In order to find an appropriate threshold, we first determine a desired false positive rate. The training dataset is used to find a threshold value that results in the specified max amount of false positives by applying a quantile function on the output of the model.

4.6. Normalization

Using separate autoencoders per application-level protocol does not only create a scalable architecture, but also simplifies normalization. When using only a single autoencoder, all feature values would be normalized within the same range, independent of the protocol. Since the traffic characteristics are heavily dependent on the protocol, different protocols will carry a different weight within the model. For example, an average HTTPS session will transport much more data than an average DNS session. After normalization, this will cause all DNS session payload values to lie close together, which decreases the distance between anomalous and benign sessions. To solve this, we normalize all sessions per protocol and train separate autoencoders.

It is important to normalize the data appropriately. If the data are not normalized, it is likely that certain features will have larger ranges than other features, causing them to overshadow the other features during training [13]. In our case, it is important to normalize in such a way that outliers are not destroyed or mitigated, as those are exactly the data points we wish to detect. We use min-max normalization in order to keep the distribution of the data intact. In min-max normalization, all values are mapped to a range of 0–1 based on their minimum and maximum observed values. If x represents the value, than the normalized representation x' is computed as follows [30]:

$$x' = \frac{x - min(X)}{max(X) - min(X)} \tag{2}$$

The min and max values are collected during aggregation. When the querying and aggregation are completed for constructing the training dataset, the entire dataset is normalized. While the anomaly detector is running and new data are arriving, the min and max values are no longer updated and data are normalized as they come in.

4.7. Deployment

Our NEDS can either be installed at a network sensor, or on a separate server which can communicate with multiple sensors. The latter option allows simple scaling, using the REST API to retrieve data from multiple sensors, and easily selecting particular sensors. A potential upside of placing the NEDS at a sensor is that the data do not have to be retrieved over HTTP/S, and can instead by retrieved directly from the sensor database.

The NPS includes a diverse range of sensors with different configurations at numerous organisations. Organisations may have more than one location where traffic exits their network, and organisations are free to place sensors wherever they prefer. It is, therefore, well possible that the sensors give only a partial view of the outgoing internet traffic for an organisation. It can be a limiting factor for training the anomaly detector if the gathered data do not represent all of the traffic that takes place. For our experiments, we selected a sensor that captures a wide range of internet traffic.

Furthermore, many organisations employ a firewall-and-proxy setup, where the sensor can be placed in front or behind, as shown in Figure 5. Placing it on the inside provides network information on specific workstations. Alternatively, placing it on the outside is more effective for detecting occurrences of Indicators of Compromise, but traffic will appear to be originating from the proxy which obscures workstation information.

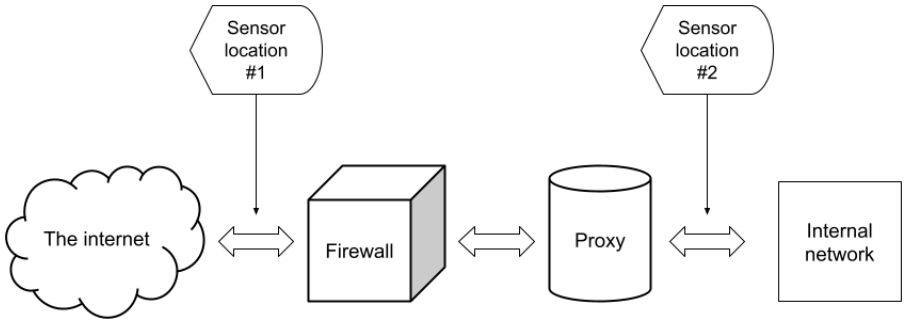

Figure 5. Network topology.

5. Experimental Setup

This section describes the setup used in our experiments. We discuss the infrastructure, the training dataset, the test dataset used for evaluation, and the detection criteria.

5.1. Infrastructure

We installed the NEDS on a separate server and connected it to one of the sensors in the NPS that monitors real-life traffic. The sensor is located in front of a proxy, on the inside of a network (see Figure 5). This means that we can observe the actual source and destination IP addresses embedded in the packets, which yields better results when aggregating.

As explained in Section 4.3, we query the REST API of the sensor at regular intervals. In principle, we want to keep this time interval small, in order to allow near-time operation of the trained anomaly detector. In our experiment, we used an interval of 2 min.

The REST API offers a size parameter to limit the maximum amount of sessions that are returned for a query. In principle, selecting a smaller size value implies that a smaller time interval between queries is required. We experimented with different values of the size parameter. We found that for a size smaller than 100, the interval between queries becomes too short and the sensor is overloaded. Additionally, for a size greater than 100,000, the sensor is overloaded and timeouts. We observed that changing the size between 100 and 100,000 had very little effect on the overall retrieval speed. We settled on using a size of 10,000.

We noticed that in the network that is monitored by the sensor, the traffic differs considerably depending on both the time of the day and the day of the week. For example, we observed much more traffic during office hours than at night, with peaks in the morning and dips around lunch time, and much less traffic on Wednesdays. When retrieving at most 10,000 sessions per interval of 2 min, we cannot capture all traffic in busy periods. Hence, in fact, we sample the traffic: each sample contains a sequence of at most 10,000 consecutive sessions, while additional sessions that occur in the 2 min interval are missed. Nevertheless, by observing traffic over a longer period, we still gain a representative view of the traffic.

As explained in Section 4.3 (Algorithm 1), we apply a damped window model for aggregating metadata with decay factor λ. We use a decay factor of 0.01, which results in a time window for aggregation of about 1 min [13].

5.2. Training Dataset

We gathered the training dataset over a period of 24 h with a query interval of 2 min, from 8:00 a.m. (UTC+2) on Thursday 14 April to 8:00 a.m. on Friday 15 April 2022. In total, 627,797 sessions were collected. Figure 6 shows the distribution of the feature values in the captured traffic. The Service figure shows that the vast majority are DNS sessions, while the other ones are mainly HTTPS sessions and some sessions with SSH, HTTP, and other protocols. The Weight figure shows that the majority of the sessions are rather short, and hence not aggregated under a key, which implies that the communication between most pairs of hosts is infrequent or spread over time (implying that aggregation decays

to 0). Additionally, the Duration and Packet count figures indicate that the majority of the sessions are rather short.

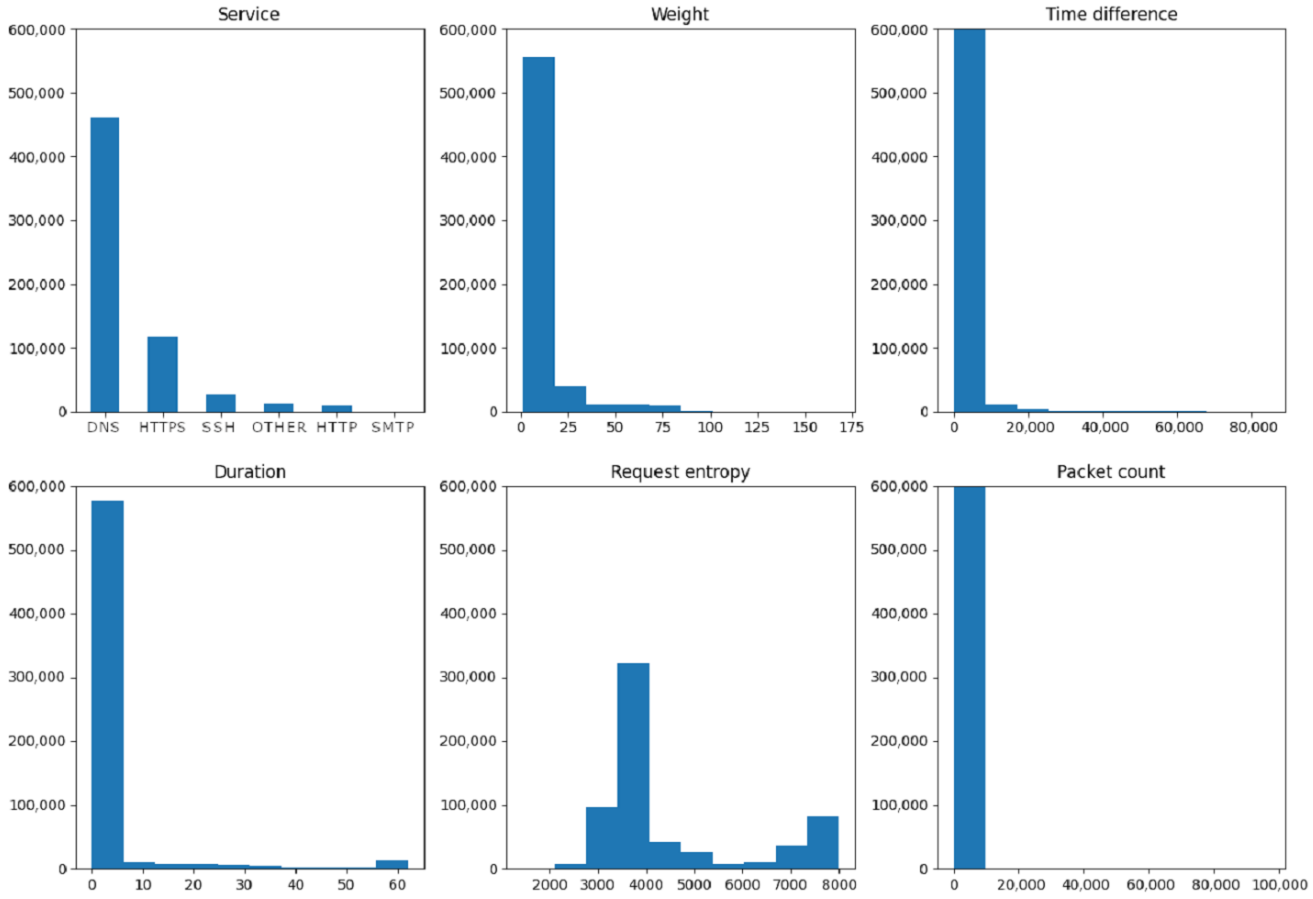

Figure 6. Feature distribution in the training dataset.

Figure 7 shows the feature distribution for DNS, HTTPS, HTTP, SSH, SMTP and other protocols. The Weight figures shows that SSH sessions are aggregated far more than other sessions. This is most likely due to the average length of an open SSH connection causing it to be split into multiple sessions. HTTPS and SSH traffic appears to have a higher average entropy, which is due to the fact that payloads are encrypted. Another interesting observation is that a considerable amount of the HTTP sessions, and HTTPS sessions to a lesser extent, have a long duration.

5.3. Test Dataset

We presume that the traffic as observed by the sensor does not contain anomalies. Since the NDN did not report suspicious activity, we can safely consider the traffic to be normal traffic, although there is a slight chance that anomalies were not detected. We, therefore, created a test dataset with normal traffic from querying the sensor data, and anomalous traffic that we injected ourselves.

5.3.1. Normal Traffic

We gathered the normal traffic in the test set in an identical way as the training set, but over a different and shorter time period. Data in the training set must not be reused in the test set, and therefore, we captured traffic in different time periods. Network traffic trends to change over longer time, to the extend that a model trained on data from more than a month ago may perform considerably worse and has to be retrained with fresh data. We queried normal traffic for the test set, again by using a 2 min time interval between

queries, one week after the training dataset from 9:30 to 11:30 a.m. on Thursday 21 April 2022. In order to limit the size of the test set, we reduced the period to 2 h. We selected the morning hours, when most traffic is generated. We collected metadata of 52,320 sessions.

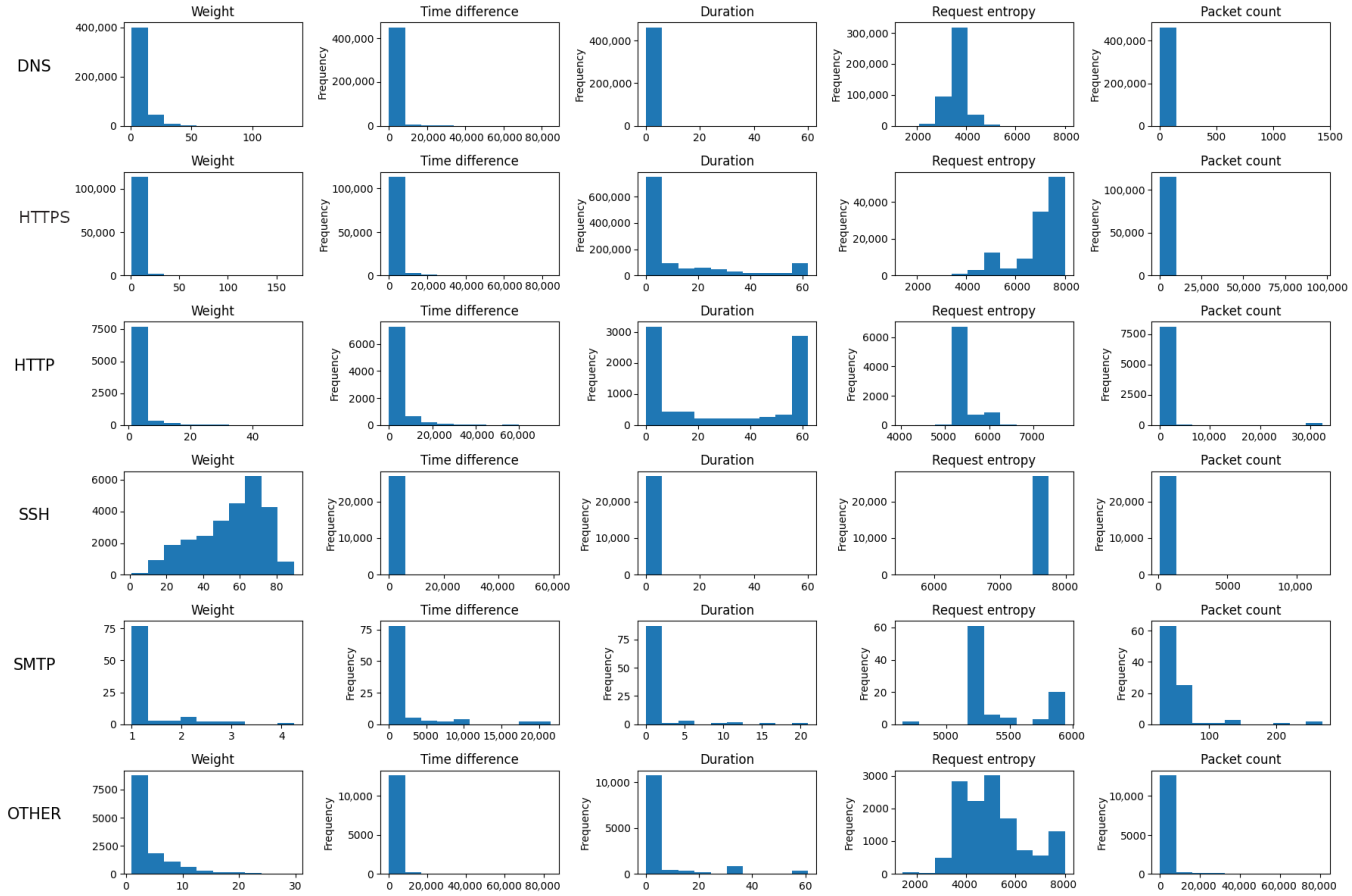

Figure 7. Feature distribution per protocol.

5.3.2. Anomalous Traffic

We added anomalous traffic to the test set. We considered data exfiltration over different channels: DNS tunnels and uploads to an FTP server, a web server, and to cloud storage.

- **DNS tunnel exfiltration**: We exfiltrated data through DNS tunnels by using the following malware: bondupdater, cobaltstrike, denis, dnspionage, ismdoor, pisloader-01, pisloader-02, and udpos. The malware creates DNS tunnels that are used as C2 channels or to exfiltrate data. Malware such as ismdoor and iodine tunnels multiple packets, while tunnels created by, for instance, pisloader-01 and pisloader-02 are very small.
- **FTP exfiltration**: We exfiltrated data by uploading files to an FTP server. We set up an FTP server on a virtual private server. The files are uploaded from another host.
- **HTTP exfiltration**: We exfiltrated data over HTTP with GET/POST requests. We set up a simple upload server on a virtual private server. The files are uploaded from another host.
- **Cloud exfiltration**: We exfiltrated data by uploading files to cloud storage, using the Rclone utility (https://rclone.org (accessed on 1 December 2021)). We setup Rclone to exfiltrate data to Google Drive.

We exfiltrated four types of files: a file containing a credit card number and files with random data of 1 kB, 100 kB, and 10 MB in size.

We performed the exfiltration and recorded the traffic generated in a PCAP-file with Wireshark (on the host from which the exfiltration is performed). We removed packets from the PCAP-file that were not related to the exfiltration. We next uploaded the PCAP-file to a separate sensor, which generates the corresponding session metadata. The name of the PCAP-file is stored as a key in a field of the metadata. We used this key to query the sensor through the REST API. By using a separate sensor, we prevented anomalies from mixing into the normal data in the training dataset and test dataset.

Table 1 shows the number of sessions that are included in the test dataset for each of the data exfiltration channels. Table 2 summarizes the numbers of sessions that are included in the training dataset and the test dataset.

Table 1. Numbers of sessions included in test dataset per exfiltration channel.

Channel	Type	Number of Sessions
DNS tunnel	bondupdater	30
	cobaltstrike	285
	denis	31
	dnspionage	8
	ismdoor	725
	pisloader-01	13
	pisloader-02	78
	udpos	127
FTP exfiltration	credit card file	4
	1 kB file	4
	100 kB file	4
	10 MB file	24
HTTP exfiltration	credit card file	1
	1 kB file	1
	100 kB file	1
	10 MB file	1
Google Drive exfiltration	credit card file	1
	1 kB file	1
	100 kB file	1
	10 MB file	3

Table 2. Numbers of sessions included in the training dataset and test dataset.

Dataset	Traffic Type	Number of Sessions
Training	Normal	627,797
Test	Normal	52,320
	Data exfiltration	1343

5.4. Evaluation Criteria

The goal of the NEDS is twofold: We want to detect anomalies, in the form of data exfiltration, as well as possible. At the same time, the number of false positives should be minimal in order to make the system usable in practice. The first goal can be evaluated by means of the true positive rate (TPR), also called sensitivity or recall, which measures what number of all anomalies is detected. The second goal can be evaluated by means of the true negative rate (TNR), also called specificity or selectivity, which measures what amount of normal traffic is classified correctly [31].

$$\text{TPR} = \text{sensitivity} = \frac{\text{true positives}}{\text{true positives} \ + \ \text{false negatives}} \tag{3}$$

$$\text{TNR} = \text{specificity} = \frac{\text{true negatives}}{\text{true negatives} \ + \ \text{false positives}} \tag{4}$$

An additional constraint is that the data are heavily unbalanced, since the amount of normal traffic is many times larger than the amount of anomalous traffic. Hence, the negative class is expected to be much larger than the positive class, which means that the amount of false positives may overshadow the amount of true positives. The precision, which measures the amount of true positives among all positive detections, is therefore not useful in our case.

The detection performance of the NEDS is largely dependent on the thresholds of the autoencoders in the anomaly detector. There is no objectively best value for these thresholds, as it differs per use case. To obtain a fair representation of the detection performance, we configure the anomaly detector with different thresholds resulting in a false positives rate of 0.001, 0.010, and 0.050 during training.

As mentioned, we consider all traffic in the training dataset to be normal, and hence, we consider positive detections during training as false positives. It may, however, be the case, albeit very rarely, that such positive detection indeed would be an actual anomaly detection. However, due to the large size of the dataset, it is impossible to investigate all false positives. In addition, the normal traffic may contain traffic that is non-malicious data exfiltration, such as automated backups. In this case, data are being exfiltrated, but in a controlled and intended manner. This may lead to false positives.

6. Experimental Results

In this section, we describe our experiments and discuss the results. We trained the autoencoders in the anomaly detector using the training dataset and evaluated the detection performance using the test dataset. In the experiments, we measured the detection performance for different thresholds of the anomaly detector, by allowing false positive rates of 0.001, 0.010, and 0.050 during training. We also evaluated the effect of using non-aggregated versus aggregated session metadata.

6.1. Results

To evaluate the effect of aggregation, we first trained and evaluated the anomaly detector with non-aggregated data, and next with aggregated session metadata. Table 3 shows results with non-aggregated data, and Table 4 with aggregated data, for data exfiltration over DNS, HTTPS, and HTTP, for the different detection thresholds.

For exfiltration over DNS tunnels, we see that the use of aggregated session metadata improves the TPR considerably. Without aggregation, the TPR is below 0.1878, while with aggregation, the TPR is above 0.6974. Additionally, increasing the threshold FPR leads to higher TPR. Even at a low threshold FPR of 0001, the TPR is 0.6974.

For exfiltration over HTTPS and HTTP, the usage of aggregated data has only marginal effect when compared with usage of non-aggregated data. Additionally, increasing the threshold FPR has hardly any effect. The only improvement is for HTTPS using aggregated data for a threshold FPR of 0.050. The TPR for HTTP is 0.3000 in all cases, which is higher than for HTTPS in nearly all cases.

Table 3. Results with non-aggregated session metadata for exfiltration over DNS, HTTPS, and HTTP.

Threshold FPR	DNS		HTTPS		HTTP	
	TPR	TNR	TPR	TNR	TPR	TNR
0.001	0.0016	0.9990	0.0000	0.9990	0.3000	0.9983
0.010	0.0368	0.9955	0.1667	0.9899	0.3000	0.9896
0.050	0.1878	0.9500	0.1667	0.9502	0.3000	0.9844

Table 4. Results with aggregated session metadata for exfiltration over DNS, HTTPS, and HTTP.

Threshold FPR	DNS		HTTPS		HTTP	
	TPR	TNR	TPR	TNR	TPR	TNR
0.001	0.6974	0.9990	0.1667	0.9990	0.3000	0.9983
0.010	0.8386	0.9900	0.1667	0.9899	0.3000	0.9896
0.050	0.9087	0.9499	0.3333	0.9495	0.3000	0.9499

Table 5 shows the results for different exfiltration channels, for both non-aggregated and aggregated data at different detection thresholds. Each row in the table shows the fraction of sessions that has been detected out of all exfiltration sessions for the particular channel. The use of aggregated data improves the detection considerably for most DNS tunnels. The exceptions are bondupdater, dnspionage, and pisloader-01. For exfiltration over FTP, the use of aggregated data leads to higher detection for larger file size. Exfiltration over HTTP is detected in all cases, except for the very small credit card file, with both non-aggregated and aggregated data. Exfiltration to Google Drive is only detected for the 10 MB file.

For DNS tunnels, increasing the threshold does not necessarily lead to more detection. The detection improves for 5 of the 8 DNS tunnels. For FTP, HTTP and Google Drive, increasing the threshold hardly improves detection, except for the 10 MB file. Even at small FPR of 0.001, exfiltration can be detected.

Table 5. Results for different data exfiltration channels.

Exfiltration Channel		Non-Aggregated Data			Aggregated Data		
Threshold FPR		**0.001**	**0.010**	**0.050**	**0.001**	**0.010**	**0.050**
DNS	bondupdater	0.000	0.000	0.033	0.000	0.000	0.000
	cobaltstrike	0.000	0.000	0.067	0.519	0.828	0.888
	denis	0.000	0.742	0.742	1.000	1.000	1.000
	dnspionage	0.000	0.250	0.375	0.000	0.000	0.000
	ismdoor	0.004	0.029	0.240	0.859	0.877	0.935
	pisloader-01	0.000	0.077	1.000	0.000	0.000	0.000
	pisloader-02	0.000	0.013	0.282	0.000	0.013	0.705
	udpos	0.000	0.000	0.055	0.000	0.701	0.843
FTP	credit card	0.500	0.500	0.500	0.500	0.500	0.500
	1 kB	0.500	0.500	0.500	0.500	0.500	0.500
	100 kB	0.500	0.500	0.500	1.000	1.000	1.000
	10 MB	0.083	0.125	0.125	0.125	0.458	1.000
HTTP	credit card	0.000	0.000	0.000	0.000	0.000	0.000
	1 kB	1.000	1.000	1.000	1.000	1.000	1.000
	100 kB	1.000	1.000	1.000	1.000	1.000	1.000
	10 MB	1.000	1.000	1.000	1.000	1.000	1.000
Drive	credit card	0.000	0.000	0.000	0.000	0.000	0.000
	1 kB	0.000	0.000	0.000	0.000	0.000	0.000
	100 kB	0.000	0.000	0.000	0.000	0.000	0.000
	10 MB	0.000	0.333	0.333	0.333	0.333	0.667

6.2. Discussion

We observe that aggregation considerably improves detection of exfiltration that is spread over multiple sessions. This is most notable for DNS tunnels, where exfiltration typically occurs over a longer time period. Results on others channels show that there is a slight improvement when using aggregated data, which tends to increase with larger file sizes of exfiltrated data. On these channels, exfiltration often occurs in a single session, for which the use of aggregated data has no additional benefit. When file sizes become

larger, the data transfer is spread over multiple sessions, and then aggregation starts to pay off.

Our work builds upon prior work by Mirsky et al. [13] and Chen et al. [4], in which anomaly detection also is addressed by using an ensemble of autoencoders. A direct comparison of the detection performance is hindered by the usage of different datasets in these works. We evaluated our NEDS using real-life data from the NDN in which we injected data exfiltration. Mirsky et al. evaluated their detection system on a dataset captured from a video surveillance network in which they injected attacks that affect the availability and integrity of the video uplinks. Chen et al. evaluated their detection system on the NSL-KDD dataset. The main difference between our work and prior work is that we target our NEDS on practical application within the NDN, with an architecture that is relatively simple and easily scalable. We leverage that the NDN provides real-life, aggregated session metadata. When compared with prior work, our anomaly detector is less complex, we apply data aggregation on the level of sessions rather than packets, and instead of clustering parts of the dataset or features per autoencoder, we assign a different application-level network protocol per autoencoder.

Using (aggregated) session metadata comes with the benefit that it is a more condensed version of network data, compared with packet data, and it is also more privacy-friendly. However, we lose the ability to inspect the content of packets.

In our experiments, we performed data exfiltration ourselves. We attempted to make this as realistic as possible, and for DNS tunnelling, we even used existing malware. However, data exfiltration may still look different in real life. Furthermore, we generated data exfiltration on a single host in an environment that is not connected to the network at which we observed the normal traffic. Hence, it is possible that the data exfiltration traffic would look slightly different when generated directly in this network.

In the experiments, we actually demonstrated that we can detect known data exfiltrations. In theory, our anomaly detector may also be able to detect still unknown zero-day attacks, since the autoencoders flag any traffic that differs sufficiently from normal traffic as an anomaly. However, it is by definition impossible to demonstrate this in practice.

We manually inspected how autoencoders reconstruct the session metadata. We observed that the autoencoders are very effective. The session metadata of anomalies that look similar to normal traffic are reconstructed well and hence remain below the threshold. This may be improved by applying machine-learning concepts such as regularisation to force the autoencoders to keep the complexity low while training, thereby increasing the distance between normal and anomalous traffic.

Autoencoders have a significant practical benefit. They allow us to scale the NEDS as we can easily add autoencoders and distribute incoming traffic over multiple servers for running autoencoders in parallel. This is hard to realize with other anomaly detection techniques such as clustering, as they often require a large amount of data to execute.

When considering actionability, i.e., the ability to respond to anomalies flagged by our NEDS, the time difference between an anomaly occurring and its detection is important. When detecting data exfiltration, the attackers can be assumed to be far along, since they already had access to hosts on which they initiated the exfiltration, and potentially may already have exfiltrated a large amount of data. Therefore, a fast response time is vital. In our current implementation, aggregated data have to be retrieved in session format from the sensor. Hence, first the sensor has to compute the session data from the packet capture, and after that, the NEDS has to query and aggregate the data and run it through the anomaly detector, which takes some time. Since fast reaction is crucial to stopping data exfiltration before it is completed, the NEDS is most useful when running on live data and taking automated action, such as closing ports or network connections to prevent further data exfiltration.

An alternative useful scenario is to use the NEDS to find anomalies in historic data, in order to search for breaches that have not been detected or to find breaches that are

still ongoing. It could also be potentially useful to determine what data exactly have been moved. This is a more useful scenario in practice for application of the current NEDS.

Another part of actionability is the information that the NEDS provides when it finds an anomaly. The current implementation does not provide context to its finding. It would, however, be interesting to determine which feature or features triggered the detection. This type of information is highly valuable to an analyst when analysing and diagnosing an anomaly.

7. Conclusions

We designed and implemented an intrusion detection system to effectively detect data exfiltration in a real-world setting using only aggregated network session metadata. We evaluated our NEDS against a realistic dataset that contains metadata on real-life normal data and custom generated data exfiltration.

Our experimental results show that aggregation of session metadata is an effective way to capture traffic between two unique hosts that happens over longer periods of time, using either stateful or stateless protocols. Aggregation considerably increases detection results for data exfiltration that is spread over multiple sessions, which, particularly, is the case in DNS tunnelling.

The anomaly detector in the NEDS contains a set of autoencoders that can be trained and run in parallel. This provides that the NEDS can be easily extended. The autoencoders are trained using unsupervised learning, and hence, no labelled dataset is required. The detection threshold is configured during training to reduce the false positive rate, which avoids time-consuming analysis of false positives.

The NEDS can be applied to analyse network traffic in near real-time, however, a delay is introduced since the sensors first have to capture network packets and compute session metadata, which is next queried and aggregated by the NEDS before running it through the anomaly detector. A more practical use case of the NEDS is to aid in post-incident analysis of captured data.

In our future work, we intend to explore how the detection accuracy of the anomaly detector can be improved, for instance, by considering other features and combining non-aggregated and aggregated session metadata. In particular, we intend to further study aspects of actionability, such as automated reacting upon anomaly detection, adding support for the diagnosis of anomalies, and adding an automated system to trigger retraining. The NEDS is currently focusing on detection of data exfiltration. We also may explore extending the NEDS for detecting other types of anomalies.

Author Contributions: Conceptualization, methodology, software, and validation, D.W.; writing, D.W. and H.V.; supervision, B.v.d.K. and K.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are not available due to their sensitive nature. Data may contain private or confidential information about state activity and/or employees.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AE	Autoencoder
API	Application Programming Interface
C2	Command & Control
DNS	Domain Name System
DPI	Deep Packet Inspection

FPR	False Positive Rate
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IOC	Indicator of Compromise
NCSC	National Cyber Security Centre
NDN	National Detection Network
NEDS	Network Exfiltration Detection System
NSP	NDN Sensor Platform
PCA	Principal Component Analysis
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SVM	Support Vector Machine
TNR	True Negative Rate
TPR	True Positive Rate

References

1. National Coordinator for Security and Counterterrorism (NCTV); Ministry of Justice and Security. *Cyber Security Assessment Netherlands 2021*; National Coordinator for Security and Counterterrorism: The Hague, The Netherlands, 2021.
2. Caviglione, L.; Choraś, M.; Corona, I.; Janicki, A.; Mazurczyk, W.; Pawlicki, M.; Wasielewska, K. Tight Arms Race: Overview of Current Malware Threats and Trends in Their Detection. *IEEE Access* **2021**, *9*, 5371–5396. [CrossRef]
3. Wang, Y.; Zhou, A.; Liao, S.; Zheng, R.; Hu, R.; Zhang, L. A comprehensive survey on DNS tunnel detection. *Comput. Netw.* **2021**, *197*, 108322. [CrossRef]
4. Chen, Z.; Yeo, C.K.; Lee, B.S.; Lau, C.T. Autoencoder-based network anomaly detection. In Proceedings of the 2018 Wireless Telecommunications Symposium (WTS), Phoenix, AZ, USA, 17–20 April 2018; pp. 1–5. [CrossRef]
5. Sabir, B.; Ullah, F.; Babar, M.A.; Gaire, R. Machine Learning for Detecting Data Exfiltration: A Review. *ACM Comput. Surv.* **2021**, *54*, 50. [CrossRef]
6. Deri, L.; Fusco, F. Using Deep Packet Inspection in CyberTraffic Analysis. In Proceedings of the 2021 IEEE International Conference on Cyber Security and Resilience (CSR), Virtual, 26–28 July 2021; pp. 89–94. [CrossRef]
7. Fadolalkarim, D.; Bertino, E. A-PANDDE: Advanced Provenance-based ANomaly Detection of Data Exfiltration. *Comput. Secur.* **2019**, *84*, 276–287. [CrossRef]
8. Liu, Y.; Corbett, C.; Chiang, K.; Archibald, R.; Mukherjee, B.; Ghosal, D. SIDD: A Framework for Detecting Sensitive Data Exfiltration by an Insider Attack. In Proceedings of the 2009 42nd Hawaii International Conference on System Sciences, Waikoloa, HI, USA, 5–8 January 2009; pp. 1–10. [CrossRef]
9. Fawcett, T.W. EXFILD: A Tool for the Detection of Data Exfiltration Using Entropy and Encryption Characteristics of Network Traffic. Master's Thesis, University of Delaware, Newark, DE, USA, 2010.
10. He, G.; Zhang, T.; Ma, Y.; Xu, B. A Novel Method to Detect Encrypted Data Exfiltration. In Proceedings of the 2014 Second International Conference on Advanced Cloud and Big Data, Huangshan, China, 20–22 November 2014; pp. 240–246. [CrossRef]
11. Nadler, A.; Aminov, A.; Shabtai, A. Detection of malicious and low throughput data exfiltration over the DNS protocol. *Comput. Secur.* **2019**, *80*, 36–53. [CrossRef]
12. Haghighat, M.H.; Foroushani, Z.A.; Li, J. SAWANT: Smart Window Based Anomaly Detection Using Netflow Traffic. In Proceedings of the 2019 IEEE 19th International Conference on Communication Technology (ICCT), Xi'an, China, 16–19 October 2019; pp. 1396–1402. [CrossRef]
13. Mirsky, Y.; Doitshman, T.; Elovici, Y.; Shabtai, A. Kitsune: An Ensemble of Autoencoders for Online Network Intrusion Detection. In Proceedings of the Network and Distributed System Security (NDSS) Symposium, San Diego, CA, USA, 18–21 February 2018.
14. Kemp, C.; Calvert, C.; Khoshgoftaar, T. Utilizing Netflow Data to Detect Slow Read Attacks. In Proceedings of the 2018 IEEE International Conference on Information Reuse and Integration (IRI), Salt Lake City, UT, USA, 6–9 July 2018; pp. 108–116. [CrossRef]
15. Najafabadi, M.M.; Khoshgoftaar, T.M.; Calvert, C.; Kemp, C. Detection of SSH Brute Force Attacks Using Aggregated Netflow Data. In Proceedings of the 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015; pp. 283–288. [CrossRef]
16. Lu, W.; Ghorbani, A.A. Network Anomaly Detection Based on Wavelet Analysis. *EURASIP J. Adv. Signal Process.* **2008**, *2009*, 837601. [CrossRef]
17. Tsikerdekis, M.; Waldron, S.; Emanuelson, A. Network Anomaly Detection Using Exponential Random Graph Models and Autoregressive Moving Average. *IEEE Access* **2021**, *9*, 134530–134542. [CrossRef]
18. Ahmed, J.; Habibi Gharakheili, H.; Raza, Q.; Russell, C.; Sivaraman, V. Monitoring Enterprise DNS Queries for Detecting Data Exfiltration From Internal Hosts. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 265–279. [CrossRef]

19. Liu, D.; Lung, C.H.; Lambadaris, I.; Seddigh, N. Network traffic anomaly detection using clustering techniques and performance comparison. In Proceedings of the 2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), Regina, SK, Canada, 5–8 May 2013; pp. 1–4. [CrossRef]
20. Liu, Y.; Xue, H.; Wei, G.; Wu, L.; Wang, Y. A Comparative Study on Network Traffic Clustering. In *Network and System Security*; Liu, J.K., Huang, X., Eds.; Springer International Publishing: Cham, Switzerland, 2019; Volume 11928, pp. 443–455. [CrossRef]
21. Münz, G.; Li, S.; Carle, G. Traffic Anomaly Detection Using K-Means Clustering. In Proceedings of the GI/ITG-Workshop MMBnet 2007, Leistungs-, Zuverlässigkeits- und Verlässlichkeitsbewertung von Kommunikationsnetzen und Verteilten Systemen, Hamburg, Germany, 13–14 September 2007; Volume 7, pp. 116–123.
22. Pagliari, R.; Ghosh, A.; Gottlieb, Y.M.; Chadha, R.; Vashist, A.; Hadynski, G. Insider attack detection using weak indicators over network flow data. In Proceedings of the MILCOM 2015—2015 IEEE Military Communications Conference, Tampa, FL, USA, 26–26 October 2015; pp. 1–6. [CrossRef]
23. Radhakrishnan, C.; Karthick, K.; Asokan, R. Ensemble Learning based Network Anomaly Detection using Clustered Generalization of the Features. In Proceedings of the 2020 2nd International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 18–19 December 2020; pp. 157–162. [CrossRef]
24. Nixon, C.; Sedky, M.; Hassan, M. Autoencoders: A Low Cost Anomaly Detection Method for Computer Network Data Streams. In Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing, Virtual, 26–28 August 2020; pp. 58–62. [CrossRef]
25. Xu, W.; Jang-Jaccard, J.; Singh, A.; Wei, Y.; Sabrina, F. Improving Performance of Autoencoder-Based Network Anomaly Detection on NSL-KDD Dataset. *IEEE Access* **2021**, *9*, 140136–140146. [CrossRef]
26. Chen, J.; Sathe, S.; Aggarwal, C.; Turaga, D. Outlier Detection with Autoencoder Ensembles. In Proceedings of the 2017 SIAM International Conference on Data Mining (SDM), Houston, TX, USA, 27–29 April 2017; pp. 90–98. [CrossRef]
27. Nguyen, Q.P.; Lim, K.W.; Divakaran, D.M.; Low, K.H.; Chan, M.C. GEE: A Gradient-based Explainable Variational Autoencoder for Network Anomaly Detection. In Proceedings of the 2019 IEEE Conference on Communications and Network Security (CNS), Washington, DC, USA, 10–12 June 2019; pp. 91–99. [CrossRef]
28. Wu, K.; Zhang, Y.; Yin, T. TDAE: Autoencoder-based Automatic Feature Learning Method for the Detection of DNS tunnel. In Proceedings of the ICC 2020—2020 IEEE International Conference on Communications (ICC), Dublin, Ireland, 7–11 June 2020; pp. 1–7. [CrossRef]
29. Mao, J.; Hu, Y.; Jiang, D.; Wei, T.; Shen, F. CBFS: A Clustering-Based Feature Selection Mechanism for Network Anomaly Detection. *IEEE Access* **2020**, *8*, 116216–116225. [CrossRef]
30. Jayalakshmi, T.; Santhakumaran, A. Statistical Normalization and Back Propagation for Classification. *Int. J. Comput. Theory Eng.* **2011**, *3*, 89–93. [CrossRef]
31. Ting, K. *Encyclopedia of Machine Learning*; Springer: Boston, MA, USA, 2011. [CrossRef]