Detection of Network Covert Channels in IoT Ecosystems Using Machine Learning*

Massimo Guarascio[1,*,†], Marco Zuppelli[2,†], Nunziato Cassavia[1,†], Giuseppe Manco[1] and Luca Caviglione[2]

[1]*ICAR - Institute for High Performance Computing and Networking, Via Pietro Bucci, cubo 8/9C – 87036, Rende, Italy*
[2]*IMATI - Institute for Applied Mathematics and Information Technologies, Via de Marini, 6 - 16149, Genova, Italy*

Abstract
Steganographic techniques and covert channels are becoming exploited by a wide-range of malware to avoid detection and bypass network security tools. With the ubiquitous diffusion of IoT nodes, such offensive schemes are expected to be used to exfiltrate data or to covertly orchestrate botnets composed of resource-constrained nodes (e.g., as it happens in Mirai). Therefore, in this paper, we present a machine learning technique for the detection of network covert channels targeting the TTL field of IPv4 datagrams. Specifically, we propose to use Autoencoders to reveal anomalous traffic behaviors. The experimental evaluation performed over realistic traffic traces showcases the effectiveness of our approach.

Keywords
Covert Channel, Autoencoder, IoT Security

1. Introduction

The Internet of Things (IoT) paradigm allows to create advanced services able to interact with the physical world and to remotely operate large-scale infrastructures. As a result, the number of applications taking advantage of IoT technologies is now almost unbounded. For instance, cost-effective sensors and devices are used for entertainment and health purposes, to access and manage industrial control systems, as well as to automatize homes and buildings. Unfortunately, the tight coupling between devices and physical entities, the resource-constrained nature of many nodes, and the lack of rigorous development or configuration processes, are at the basis of countless security and privacy flaws [1].

Despite IoT nodes are often considered simple devices, they can be used to implement effective threats. As an example, the Mirai malware allows to create a large-scale botnet of devices with limited computing and connectivity resources, which has been used to launch Distributed Denial of Service (DDoS) attacks against many international organizations and sensitive targets [2].

ITASEC'22: Italian Conference on Cybersecurity, June 20–23, 2022, Rome, Italy
*Corresponding author.
†These authors contributed equally.
massimo.guarascio@icar.cnr.it (M. Guarascio); marco.zuppelli@ge.imati.cnr.it (M. Zuppelli); nunziato.cassavia@icar.cnr.it (N. Cassavia); giuseppe.manco@icar.cnr.it (G. Manco); luca.caviglione@ge.imati.cnr.it (L. Caviglione)
0000-0001-7711-9833 (M. Guarascio); 0000-0001-6932-3199 (M. Zuppelli); 0000-0001-9991-5021 (N. Cassavia); 0000-0001-9672-3833 (G. Manco); 0000-0001-6466-3354 (L. Caviglione)

In addition, IoT nodes can be enumerated to infer details on the physical deployment [3] and their traffic can be inspected to implement various side-channel-based techniques or to conduct reconnaissance campaigns [4]. Therefore, a major effort is devoted to make IoT ecosystems more secure, but this could be partially voided by the recent trend of developing malware able to remain undetected and bypass classical network security mechanisms. This new class of threats takes advantage of various information hiding and steganographic techniques to conceal malicious payloads in innocent-looking software assets, retrieve additional configuration files without being noticed, or covertly exfiltrate hidden information [5]. Among the various attack mechanisms, the adoption of network covert channels, i.e., cloaked and parasitic communication paths nested within legitimate traffic flows, is gaining momentum. Specifically, network covert channels can be exploited to establish Command & Control (C&C) communications, as well as to bypass intrusion detection/prevention systems and firewalls for stealthily exchange vast volumes of personal data [5, 4]. A recent example of attack using a network covert channel is Sunburst, which hides commands in HTTP traffic[1]. Due to the ubiquitous availability of devices always connected to the Internet, their intrinsic interaction with sensitive data, as well as several design flaws and limitations, completely assessing the security of IoT deployments requires also to consider threats endowed with network covert channels capabilities. To develop suitable mitigation techniques, machine learning approaches demonstrated to be effective for detecting a multitude of network attacks and to implement general intrusion detection mechanisms [6]. Unfortunately, countermeasures against network covert channels are poorly-generalizable, since each hiding mechanism and network protocol have specific traits and behaviors [7]. For instance, using some form of AI to reveal a channel hidden within DNS traffic [8] requires a complete different inspection mechanism and metrics compared to the case of parasitic communications targeting IPv6 conversations [9]. As a result, the literature abounds of attack-specific detection methodologies and working towards a unique framework is still an open research problem (see, e.g., [10] for a recent survey on the topic). A different case concerns the detection of timing channels, which are created by encoding information in temporal statistics of network traffic. To this aim, the secret information is usually hidden within the inter-packet time or in the throughput characterizing a specific stream or network conversation [5, 7]. Owing to the protocol-agnostic trait of timing protocols, several works using machine learning have been proposed [11] even by exploiting techniques originally introduced for image processing [12].

Therefore, in this work we address the problem of detecting network covert channels targeting the TTL field of IPv4 datagrams. In fact, the resource-constrained nature of IoT devices, including the use of "lean" TCP/IP protocol stacks to tame complexity, prevent malware to implement sophisticated covert channels or computing-intensive network steganography algorithms. To develop our detection methods, we take advantage of autoencoders, which are neural networks where the target of the network is the data itself. Autoencoders allow to reduce dimensionality and to learn efficient encoding, whereas they are a convenient choice when in absence of a labeled dataset. This is of prime importance when addressing malware exploiting network covert channels, since it often remains undetected or undocumented until major reverse engineering or forensics investigations [13]. As regards prior works considering covert channels targeting

[1]An updated list of attacks leveraging information hiding, steganography, and covert channels observed "in the wild" is available online at: https://github.com/lucacav/steg-in-the-wild [Last Accessed: June 2022].

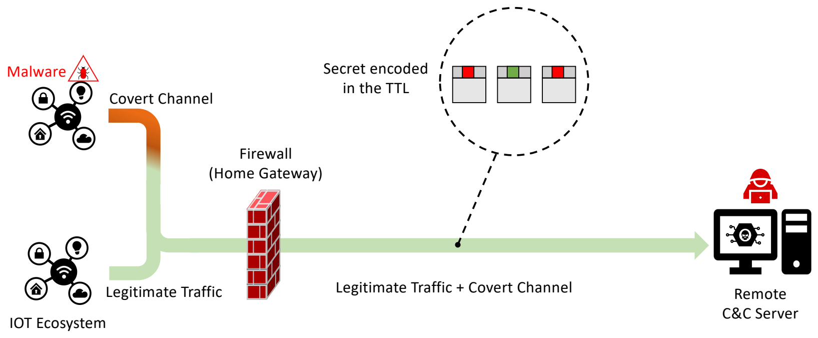


Figure 1: Attack model considering a malware sending data towards a remote command and control facility via a network covert channel created within the TTL field of IPv4 traffic.

IoT scenarios, the literature mainly focuses on timing channels, for instance to detect cloaked communications in SCADA applications [14] or in the Constrained Application Protocol [15].

Summing up, the contributions of this work are: the design of a machine-learning-capable approach for detecting covert channels targeting IoT ecosystems, and a performance evaluation campaign based on realistic traffic traces commonly used in the literature. Since countermeasures could be also deployed at the border of the network in nodes with limited capabilities (e.g., home gateways) emphasis has been put on the footprint required by the proposed approach.

The remainder of the paper is structured as follows. Section 2 provides details on the considered attack model, Section 3 introduces our approach to detect covert channels targeting the TTL of IPv4 datagrams, and Section 4 showcases numerical results. Finally, Section 5 concludes the paper and outlines possible future research directions.

2. Attack Model and Design of the Covert Channel

This section discusses the attack model taking advantage of a network covert channel. Figure 1 showcases the general reference scenario. Specifically, we consider an attacker able to take control of one or more IoT nodes, for instance by dropping a malicious payload via a phishing campaign [1]. The infected device will then create a network covert channel to exfiltrate data towards a remote C&C server or to exchange commands with the attacker, e.g., to configure a backdoor or operate a botnet. Relying upon a network covert channel allows to bypass a firewall or specific security policies enforced by a middlebox, such as a home gateway.

Even if the literature abounds of techniques for creating cloaked communication paths within network flows and real-world threats taking advantage of information hiding are multiplying [5, 7, 16], the resource-limited nature of IoT nodes poses constraints on the complexity of the covert channel. As a consequence, the embedding mechanism should be simple in order to not disclose the presence of the malware due to perceptible lags or anomalous depletion of batteries. At the same time, since IoT traffic often requires some form of Quality of Experience (e.g., to not postpone the execution of commands sent by the user), traffic alterations and the introduction of additional delays should be limited. Therefore, we consider a malware cloaking data within the TTL field of the IPv4 header [7]. In more detail, the TTL is manipulated to implement a storage network covert channel and transport arbitrary information. Due to the varying nature of the TTL and to not appear suspicious, the malware should not directly write the secret data in the field [17]. Rather, it can encode the bits 1 and 0 by increasing or decreasing the observed TTL of a suitable threshold or by using most common values as “high” and “low” signals. Finding proper TTL values is not trivial, since their difference should be ample enough to absorb fluctuations caused by alterations of the routing and to prevent decoding errors, while not reducing the stealthiness of the channel. To design the covert channel, the attacker usually investigates the targeted network to understand “clean” traffic conditions and adapt the hiding mechanism. To tune the channel, we considered the collection of IoT traffic made available in [3]. As an example, we showcase results for the 24-hour slice of data captured from September 22, 2016 at 16:00 to September 23, 2016 at 16:00, CEST[2]. Without loss of generality and to prevent burdening results, we removed IPv6, ICMP, DNS and NTP conversations, in addition to multicast/broadcast traffic. Figure 2 depicts heatmaps for the collected TTL values. As depicted in Figure 2a, the values observed for the TTL are clusterized, especially in the $32-64$ and $208-224$ ranges. This requires the attacker to encode information without using values never observed in normal conditions. Yet, traffic conditions are not static, hence, we refined our analysis by resetting the observed values each hour. Figure 2b portraits results. As shown, some values of the TTL are always present in the traffic (e.g., those around 48), whereas others have an intermittent behavior. For instance, datagrams with a TTL equal to 128 are present only for 3 hours (i.e., from 13-th to the 16-th hours). This puts constraints on the temporal location of channels using a TTL equal to 128 as well as on their duration.

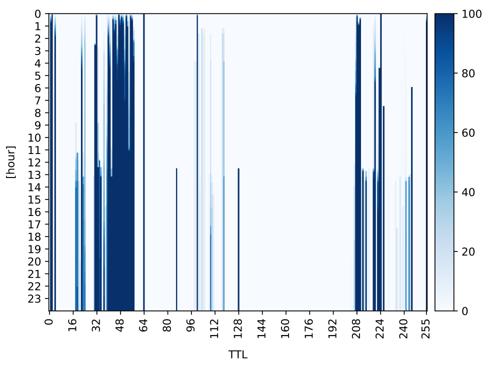
In general, channels targeting the TTL should alter datagrams in a limited manner in order to avoid macroscopic per-flow signatures [17]. Moreover, TTL values highly depend on the type of nodes, hosts and appliances exchanging traffic through the network. In fact, Android and iOS devices as well as Linux hosts generate traffic with a default TTL of 64, whereas Windows nodes use a default TTL of 128 [18]. Thus another important trade off should aim at avoiding to make the channel detectable via simple host/OS fingerprinting mechanism

3. Detecting Covert Channels With Autoencoders

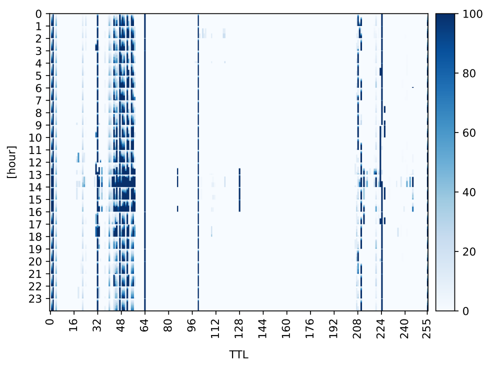
This section describes the deep learning-based approach adopted to identify the presence of covert channels within traffic flows. In our scenario, the detector takes the form of an unsupervised deep neural network. The main benefit of our approach is the possibility of the model to raise alarms also on never seen attacks: this is a frequent case when dealing with covert channels, since they are often undocumented and unknown *a priori*. Moreover, this solution allows for coping with the lack of labeled data issue, typical of our application scenario.

Specifically, our solution allows for learning a neural encoder-decoder network aiming at compressing the input data (represented by metrics computed over the traffic generated by the IoT network) within a latent space, which is then used to reconstruct the original information. Here, the main idea is that the legitimate input data should be slightly affected by the encoding/decoding procedures performed by the model, therefore the original distributions

[2]Data collected for IEEE TMC 2018, University of New South Wales, Sydney. Available online at: https://iotanalytics.unsw.edu.au/iottraces.html [Last Accessed: June 2022].



(a) Heatmap computed over the entire dataset



(b) Heatmap computed on 1-hour long slices

Figure 2: Various heatmaps computed over a 24h traffic trace.

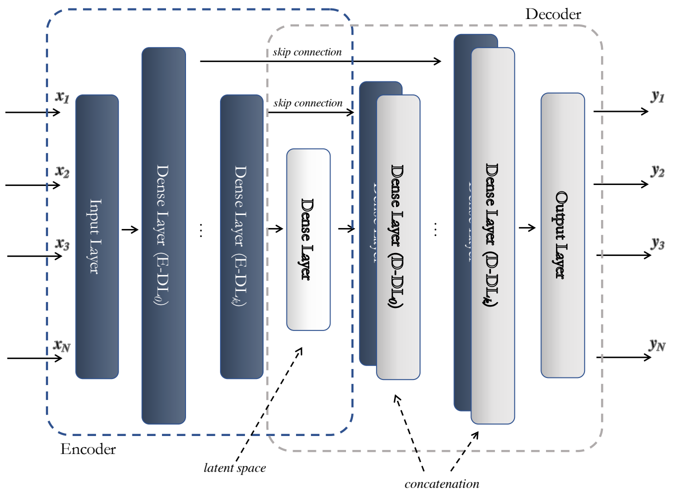


Figure 3: Neural architecture (Sparse Autoencoder) used to perform the detection of covert channels targeting the TTL of IPv4 traffic.

substantially remain unchanged after this process. By contrast, anomalous instances will exhibit deviant values that can lead to failures in the input reconstruction.

Although the idea to use the reconstruction error as an anomaly score to identify deviant behaviors is not new itself, the adoption of unsupervised techniques (and in particular of autoencoder-based solutions) for detecting covert channels is quite unexplored [6, 11, 10]. Hence, the use of autoencoders [19, 20] represents an effective approach to the unsupervised task of learning a compressed representation able to effectively summarize the main information contained in the input data. In essence, it can be thought as a neural network whose aim consists in yielding as output a duplicate (as close as possible) similar to the input data.

Figure 3 shows the considered neural architecture. As shown, the architecture is composed of two main components, named *Encoder* and *Decoder*, respectively. Let $x = \{x_1, \ldots, x_N\}$ be a set of numeric features (in our scenario, a set of statistics computed on the network traffic flow yielded in a time slot). The former subnet allows for mapping $\mathbf{z} = enc(\mathbf{x})$ the input data with a latent space (*encoding*), whereas the second one maps the features extracted by the encoder with the output $\mathbf{y} = dec(\mathbf{z})$ (*decoding*). Gradient descent is used to learn the model weights by minimizing a suitable reconstruction loss. In this paper, we adopted the *Mean Square Error*, i.e., $Loss_{MSE}(x) = \frac{1}{N} \sum_i \|\mathbf{x_i} - \mathbf{y_i}\|^2$.

Notably, the architecture shown in Figure 3 exhibits two main differences with respect to a standard encoder-decoder model: *(i) Skip Connections* are used to boost the predictive performances of the model and to reduce the number of iterations required for the learning algorithm convergence, and *(ii)* a hybrid approach including the usage of *Sparse Dense Layers* is adopted to make the autoencoder more robust to noise, especially since attacks often exhibit slight differences compared with normal behaviors. In more detail, the idea behind Skip Connections is to "help" the learning phase of the decoder by providing as input to each layer of the decoder both the previous and the correspondent encoder layer. As regard the use of Sparse Dense Layers, this allows for yielding a wider number of discriminative features that can be used to extract a more effective latent representation.

Figure 4 shows how the detection of covert channels targeting the TTL of IPv4 datagrams is performed. Without loss of generality, we assume to monitor an infinite datastream, i.e., the traffic produced by the various IoT nodes continuously feeds our detection mechanism. At pre-fixed time intervals (corresponding to a time slot in Figure 4), we compute a number of statistics to describe the behavior of the TTL fields composing the aggregate traffic flow. This operation can be performed without impacting on the overall traffic and by using limited computing resources (see, e.g., the use of the extended Berkeley Packet Filter (eBPF) [9]). In more detail, we compute metrics such as the min, average, max, different percentiles, etc., starting from TTL values gathered from the packets composing the inspected traffic aggregate. First, an autoencoder, pretrained only against legitimate data flows, is used to reproduce the statistics, then reconstruction error is computed for the current example as the MSE between x and y. Finally, if the error is lesser than a given *outlierness threshold*, the current data are labeled as "normal" and exploited to update the model, otherwise a warning is raised.

4. Performance Evaluation

In this section we present the performance of the approach based on autoencoders. Preliminary, we discuss the used dataset, then we showcase numerical results.

4.1. Dataset Preparation

To evaluate the effectiveness of our approach for detecting network covert channels targeting IoT ecosystems, we prepared an artificial dataset starting from the traffic traces made available in [3]. In more detail, we used datasets containing traffic collected from September 22, 2016 at 16:00 to September 29, 2016 at 16:00, CEST. Similarly to the example of Section 2, we removed IPv6, ICMP, DNS and NTP conversations as well as multicast/broadcast traffic. To avoid unwanted signatures/fingerprints, we also removed traffic generated by non-IoT devices, such as mobile

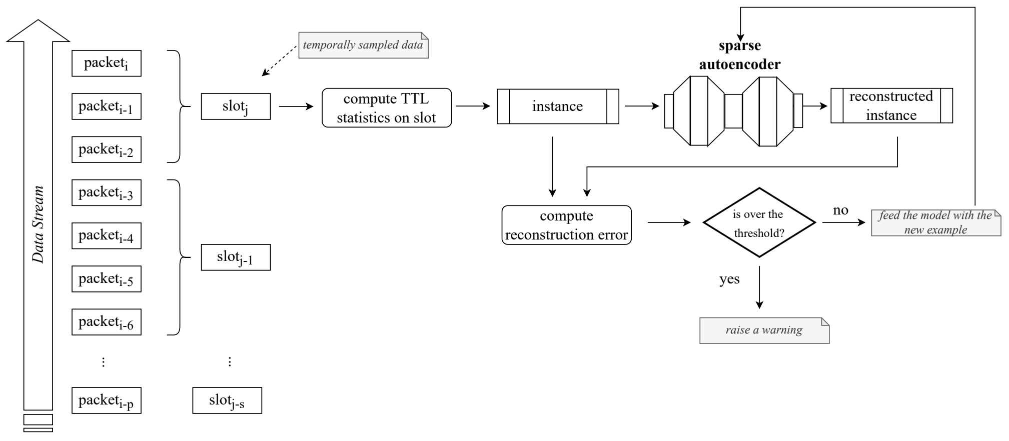


Figure 4: Detection mechanism used to reveal the presence of network covert channels.

phones and laptops. We then obtained a 1-week long dataset with an overall throughput in the $5 - 36$ kbit/s range, generated by 28 IoT endpoints, such as speakers, lights, cameras, and hubs.

To implement the considered attack template in a realistic manner, we modeled the presence of a threat tampering a single IoT device. As an example, the attacker could gain access to the assets of the victim via phishing or by exploiting some ad-hoc CVEs[3]. In our scenario, we considered a malicious software targeting the Dropcam camera, which has been used to send/exfiltrate sensitive data towards a remote C&C facility. To have a dataset containing fair amounts of "legitimate" and cloaked conversations, we assumed that the IoT device has been tampered on September 27, thus the Dropcam has been under control of the attacker for 3 days.

To create the various storage network covert channels, we used the tool available in [21], which allows to directly rewrite the traffic captures and implement realistic attack conditions. As discussed in Section 2, to not make the detection trivial, we encoded bits `1` and `0` in TTL values equal to 64 and 100, respectively. Moreover, we randomly interleaved packets containing hidden data with legitimate/unaltered packets in order to prevent long bursts of manipulated TTL values. In fact, the latter could reduce the stealthiness of the covert channel leading to a trivial detection [17]. Such a behavior can be ascribed to an attacker switching the hidden communication among two states (i.e., exfiltrate data and not manipulate traffic) to remain unnoticed via elusive mechanisms. To avoid further statistical signatures, the secret data transmitted over the covert channel has been modeled with randomly-generated strings: this represents an attacker using some obfuscation technique, e.g., encryption or scrambling [22]. Concerning the volume of data transmitted within the covert channel, we modeled each day of attack with a different template. Specifically, we considered the exfiltration of 69, 80, and 64 kbit of data. Such volumes can represent sensitive information like several username+password pairs or configuration details of a specific IoT device or smart hub. Moreover, assuming covert transmissions in the $64 - 80$ kbit range allowed to have an IoT node accounting for a variable amount of steganographically-modified traffic. In more detail, the compromised IoT node

[3]List of CVEs targeting IoT nodes/devices maintained by MITRE. Available online at: https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=iot [Last Accessed: June 2022].

manipulates the 18%, 1%, and 12% of the overall daily traffic, respectively.

4.2. Pre-processing, Parameters and Evaluation Metrics

To test our approach for revealing the presence of network covert channels within traffic aggregates, we developed a prototype in Python based on the TensorFlow[4] library. The traffic dataset presented in Section 4.1 has been processed to obtain the following information: a progressive timestamp, the number of incoming packets within a given time slot, the average and median values of observed TTLs, the values of the 10^{th}, 25^{th}, 75^{th} and 90^{th} percentile, minimum and maximum TTLs, as well as a label indicating the presence of the attack (i.e., for testing purposes). Recalling that our approach exploits a "slotted" architecture (see Figure 4), in this work we consider a time slot with a duration of 5 seconds.

The dataset has been divided in training and test sets by using a temporal split. Specifically: *(i)* the data gathered in the first 96 hours only contains legitimate traffic and has been used for the learning phase of the autoencoder, whereas *(ii)* the remaining instances compose the test set. As a result, the training and the test set have $69,116$ and $51,837$ instances, respectively. To normalize the input data feeding the model, a pre-processing phase has been performed. In essence, a *MinMax* normalization has been used to map each feature in the range $\{-1, 1\}$ in order to improve the stability of the learning process.

As discussed in Section 3, the proposed model is a neural network composed of two subnets. The *Encoder* has four fully-connected dense layers. Three layers have been instantiated with 32, 16, and 8 neurons and equipped with a ReLU (Rectified Linear Unit) activation function. The fourth layer is the latent space and can be thought as a dense layer (shared between the encoder and the decoder) including 4 neurons, and it is equipped with a ReLU activation function. The *Decoder* is composed again of three fully-connected dense layers with the same dimensions and activation function. Finally, the output layer is instantiated with the same size of the input, and equipped with a *Tanh* activation function. This choice has been made since we want to yield an output ranging in $\{-1, 1\}$. The model is trained over 16 epochs with a batch size of 16.

To assess the detection capabilities, we computed the following performance metrics. Let us define TP as the number of positive cases correctly classified, FP as the number of negative cases incorrectly classified as positive, FN as the number of positive cases incorrectly classified as negative, and TN as the number of negative cases correctly classified. Then, we considered the following metrics: the *Accuracy*, defined as the fraction of cases correctly classified, i.e., $\frac{TP+TN}{TP+FP+FN+TN}$, the *Precision* and the *Recall* to measure the accuracy in identifying attacks and avoiding false alarms, i.e., $\frac{TP}{TP+FP}$ and $\frac{TP}{TP+FN}$, respectively. We also considered the *F-Measure* to summarize the overall system performances as the harmonic mean of *Precision* and *Recall*.

Lastly, to perform experiments, we used a machine equipped with 32 Gb RAM, an Intel i7-4790K CPU @4.00GHz and an 1Tb SSD disk drive.

4.3. Numerical Results

Since the outlierness threshold can influence the detection capability of the proposed approach, we investigated its impact.

[4]TensorFlow machine learning library. Available online at: https://www.tensorflow.org/ [Last Accessed: June 2022].

Table 1
Experimental results for different outlier thresholds. Values have been selected by computing the outlier scores against the training set and by extracting the correspondent percentile values.

Threshold Values	Accuracy	Precision	Recall	F-Measure
0.040 - (90^{th} perc.)	0.872	0.729	0.991	0.840
0.055 - (95^{th} perc.)	0.901	0.791	0.964	0.869
0.126 - (99^{th} perc.)	0.910	0.937	0.786	0.855

As the autoencoder model is trained only against legitimate data (i.e., clean traffic produced by IoT nodes), we computed the outlierness degree for each slot composing the training set. We then selected as the anomaly threshold the values corresponding to the 90^{th}, 95^{th} and 99^{th} percentiles. A detailed breakdown is depicted in Figure 5.

Table 1 reports experimental results obtained by taking into account different outlierness thresholds computed over the training set. As shown, collected values exhibit an intuitive behavior, i.e., when a more restrictive threshold is selected (99^{th} percentile), the approach exhibits a good precision ($\sim$ 94%), but a percentage of slots containing a network covert channel is not correctly recognized. By contrast, a looser threshold value (90^{th} percentile) allows to improve the probability of detection ($\sim$ 99% of recall), but a higher number of false alarms are raised. This can be mitigated by considering our mechanism as a first stage of a more complex detection chain, which can trigger more resource-consuming approaches such as deep packet inspection. Yet, the best setting is the one where the 95^{th} percentile is used, since it guarantees the highest value in terms of F-Measure. This represents the best trade off between probability of detecting the presence of a covert communication and false alarm rate.

Moreover, Figure 6 portraits the distribution of the outlierness degree for a window including a marked number of compromised time slots. As it can be seen, the outlierness degree exhibits higher values than the ones reported in Figure 5. In some cases, the outlierness is one order of magnitude higher than the outlierness max value computed on the training set. This event represents the presence of a covert communications within the bulk of traffic, thus leading to a “deviation” in the output of the neural network.

Lastly, as regards the feasibility of deploying our approach in realistic settings, we point out that its resource footprint is very limited. In more detail, gathering information about the TTL usually accounts for an additional packet delay of $\sim$ 100 ns when using eBPF and 1 ms with a C implementation exploiting libpcap over commodity hardware. Instead, apart the training phase, which can be done offline, the average prediction time is 0.0132 ms. Another important aspect concerns the “stateless” nature of the approach. In fact, the used neural architecture performs the detection of covert communications by using information on the overall traffic (grouped in time slots), which prevents memory consumption due to the need of storing information with a per-flow granularity. Thus, the proposed approach should be considered suitable for being implemented in home gateways often used in production-quality IoT ecosystems.

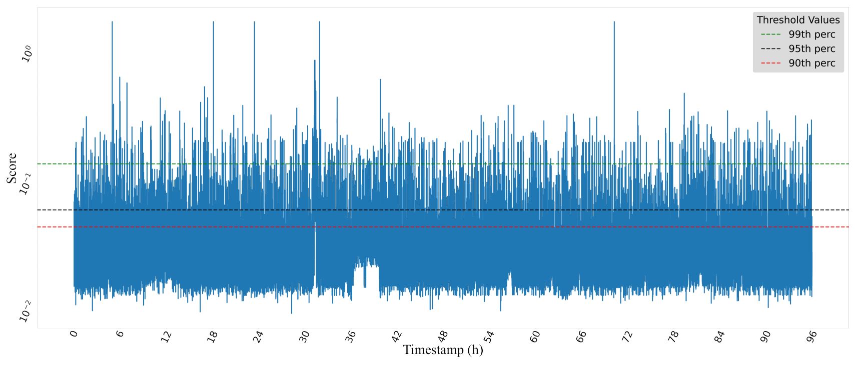


Figure 5: Training-set outlierness with different threshold values.

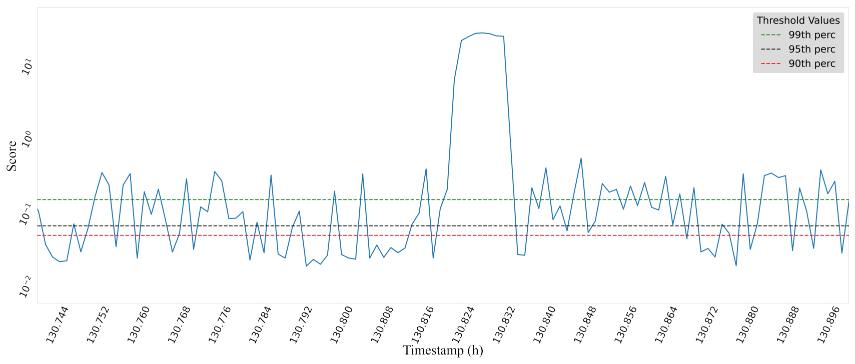


Figure 6: Test-set outlierness.

5. Conclusions and Future Work

In this paper, we presented a lightweight mechanism based on autoencoders for detecting network covert channels targeting IoT scenarios. Results indicated the effectiveness of our approach, i.e., the method can achieve the values of $\sim 91\%$ and $\sim 94\%$ for the accuracy and the precision, respectively. Although our solution addresses a specific case, it can be easily generalized to handle different network covert channels and environments, e.g., by considering an ensemble of specialized detectors combined to reveal attacks on different carriers.

Future works aim at refining the proposed framework by considering other types of network covert channels. At the same time, part of our ongoing research is devoted to develop some form of "intermediate" representations, which can be used to exploit a unique mechanism to face different threats. We are working towards general metrics that could partially compensate the tight-coupling between the used hiding methodology/protocol and the countermeasure.

Acknowledgments

This work has been partially supported by the H2020 Program within the framework of SIMARGL (Grant Agreement No. 833042), and CyberSec4Europe (Grant Agreement No. 830929).

References

[1] N. Neshenko, E. Bou-Harb, J. Crichigno, G. Kaddoum, N. Ghani, Demystifying IoT security: An exhaustive survey on IoT vulnerabilities and a first empirical look on Internet-scale IoT exploitations, IEEE Communications Surveys & Tutorials 21 (2019) 2702–2733.

[2] M. Antonakakis, T. April, M. Bailey, M. Bernhard, E. Bursztein, J. Cochran, Z. Durumeric, J. A. Halderman, L. Invernizzi, M. Kallitsis, et al., Understanding the Mirai botnet, in: 26th USENIX security symposium (USENIX Security 17), 2017, pp. 1093–1110.

[3] A. Sivanathan, H. H. Gharakheili, F. Loi, A. Radford, C. Wijenayake, A. Vishwanath, V. Sivaraman, Classifying IoT devices in smart environments using network traffic characteristics, IEEE Transactions on Mobile Computing 18 (2018) 1745–1759.

[4] W. Mazurczyk, L. Caviglione, Cyber reconnaissance techniques, Communications of the ACM 64 (2021) 86–95.

[5] W. Mazurczyk, L. Caviglione, Information hiding as a challenge for malware detection, IEEE Security Privacy 13 (2015) 89–93.

[6] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, F. Ahmad, Network intrusion detection system: A systematic study of machine learning and deep learning approaches, Transactions on Emerging Telecommunications Technologies 32 (2021) e4150.

[7] S. Zander, G. Armitage, P. Branch, A survey of covert channels and countermeasures in computer network protocols, IEEE Communications Surveys & Tutorials 9 (2007) 44–57.

[8] S. Chen, B. Lang, H. Liu, D. Li, C. Gao, DNS covert channel detection method using the LSTM model, Computers & Security 104 (2021) 102095.

[9] M. Repetto, L. Caviglione, M. Zuppelli, bccstego: A framework for investigating network covert channels, in: The 16th International Conference on Availability, Reliability and Security, 2021, pp. 1–7.

[10] M. A. Elsadig, A. Gafar, Covert channel detection: Machine learning approaches, IEEE Access (2022).

[11] O. Darwish, A. Al-Fuqaha, G. B. Brahim, I. Jenhani, A. Vasilakos, Using hierarchical statistical analysis and deep neural networks to detect covert timing channels, Applied Soft Computing 82 (2019) 105546.

[12] S. Al-Eidi, O. Darwish, Y. Chen, G. Husari, Snapcatch: automatic detection of covert timing channels using image processing and machine learning, IEEE Access 9 (2020) 177–191.

[13] W. Mazurczyk, S. Wendzel, Information hiding: challenges for forensic experts, Communications of the ACM 61 (2017) 86–94.

[14] C. Alcaraz, G. Bernieri, F. Pascucci, J. Lopez, R. Setola, Covert channels-based stealth attacks in industry 4.0, IEEE Systems Journal 13 (2019) 3980–3988.

[15] A. Velinov, A. Mileva, D. Stojanov, Power consumption analysis of the new covert channels in CoAP, International Journal On Advances in Security 12 (2019) 42–52.

[16] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, K. Wasielewska, Tight arms race: Overview of current malware threats and trends in their detection, IEEE Access 9 (2020) 5371–5396.

[17] S. Zander, G. Armitage, P. Branch, Covert channels in the IP time to live field, Swinburne University of Technology Report (2006).

[18] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, L. Qiu, OS fingerprinting and tethering detection

in mobile networks, in: Proceedings of the 2014 Conference on Internet Measurement Conference, 2014, pp. 173–180.

[19] G. Hinton, R. Salakhutdinov, Reducing the dimensionality of data with neural networks, Science 313 (2006) 504 – 507.

[20] Y. Bengio, L. Pascal, P. Dan, H. Larochelle, Greedy layer-wise training of deep networks, in: Advances in Neural Information Processing Systems, volume 19, MIT Press, 2007, pp. 153–160.

[21] M. Zuppelli, L. Caviglione, pcapstego: A tool for generating traffic traces for experimenting with network covert channels, in: The 16th International Conference on Availability, Reliability and Security, 2021, pp. 1–8.

[22] P. McLaren, G. Russell, B. Buchanan, Mining malware command and control traces, in: 2017 Computing Conference, 2017, pp. 788–794.