Journal of Intelligent Information Systems
https://doi.org/10.1007/s10844-023-00819-8

RESEARCH

Learning autoencoder ensembles for detecting malware hidden communications in IoT ecosystems

Nunziato Cassavia[1] · Luca Caviglione[2] · Massimo Guarascio[1] · Angelica Liguori[1,3] · Marco Zuppelli[2]

Received: 23 May 2023 / Revised: 12 September 2023 / Accepted: 26 September 2023

Abstract
Modern IoT ecosystems are the preferred target of threat actors wanting to incorporate resource-constrained devices within a botnet or leak sensitive information. A major research effort is then devoted to create countermeasures for mitigating attacks, for instance, hardware-level verification mechanisms or effective network intrusion detection frameworks. Unfortunately, advanced malware is often endowed with the ability of cloaking communications within network traffic, e.g., to orchestrate compromised IoT nodes or exfiltrate data without being noticed. Therefore, this paper showcases how different autoencoder-based architectures can spot the presence of malicious communications hidden in conversations, especially in the TTL of IPv4 traffic. To conduct tests, this work considers IoT traffic traces gathered in a real setting and the presence of an attacker deploying two hiding schemes (i.e., naive and "elusive" approaches). Collected results showcase the effectiveness of our method as well as the feasibility of deploying autoencoders in production-quality IoT settings.

Keywords Deep autoencoder · Ensemble method · Covert channel · Intelligent cyber attack detection system

✉ Angelica Liguori
angelica.liguori@dimes.unical.it

Nunziato Cassavia
nunziato.cassavia@icar.cnr.it

Luca Caviglione
luca.caviglione@ge.imati.cnr.it

Massimo Guarascio
massimo.guarascio@icar.cnr.it

Marco Zuppelli
marco.zuppelli@ge.imati.cnr.it

[1] Institute for High Performance Computing and Networking, Via Pietro Bucci 8-9/C, Rende 87036, Italy

[2] Institute for Applied Mathematics and Information Technologies, Via de Marini 6, Genova 16149, Italy

[3] University of Calabria, Via Pietro Bucci, Rende 87036, Italy

Published online: 03 November 2023

1 Introduction

The flexibility of Internet of Things (IoT) technologies is the main driver for their diffusion, especially in large-scale scenarios, such as the remote monitoring of critical infrastructures or the control of complex cyber-physical systems. Despite the proliferation of products targeting the mass market (e.g., for smart lighting) as well as the integration of sensors to create novel use cases (e.g., for health applications or the engagement of tourists (Balandina et al., 2015)), IoT ecosystems are frequently plagued by countless security and privacy flaws (Neshenko et al., 2019). Specifically, the resource-constrained nature of many affordable devices often requires "lean" protocols/services, which could partially fail to guarantee suitable degrees of security. Moreover, the consumer-oriented nature of several IoT ecosystems may cause erratic configuration policies, the presence of aged devices without any timely fix against new CVEs, and incorrect traffic partitioning schemes. As a consequence, threat actors intensified their activity against IoT deployments in the last years. For instance, the Mirai malware allowed to access a multitude of IoT devices to launch attacks against many international organizations (Antonakakis et al., 2017), while traffic inspection and fingerprinting have been largely used to derive habits or gather sensitive information (Sivanathan et al., 2018).

Therefore, both industry and academia are making considerable efforts to prevent and detect attacks against IoT nodes, also by deploying Artificial Intelligence (AI) and Machine Learning (ML) techniques (see, e.g., (Sahu et al., 2021)). Unfortunately, this spawned an "arm race", leading to a new-wave of malware endowed with mechanisms to remain unnoticed, bypass secure execution perimeters, or resist forensics and reverse-engineering attempts (Chakkaravarthy et al., 2019). Among the various offensive techniques, the ability to cloak network activity or implement hidden communications services are demonstrating their effectiveness. For instance, nodes of a botnet may encrypt traffic, mimic other applications and protocols, or eavesdrop well-known services (Vormayr et al., 2017). To this aim, an emerging attack scheme exploits network covert channels, i.e., parasitic communication paths cloaked within legitimate traffic flows (Mazurczyk and Caviglione, 2015). In more detail, they can make it difficult to recognize Command & Control (C&C) flows or the exfiltration of data within the bulk of traffic. Covert channels can be also used to bypass traffic blockages or firewalls, for instance, to orchestrate compromised devices. Alas, each method for hiding the presence of a malicious communication requires a tight coupling with the abused protocol, thus making the design and deployments of countermeasures poorly generalizable (Zander et al., 2007). Fortunately, the various cloaking mechanisms could be brought back to a set of recurrent "hiding patterns", mainly based on the overwriting of a field or the manipulation of a temporal behavior, see, (Wendzel et al. (2015); Wendzel et al. (2021)) for a detailed taxonomy.

To partially balance the intrinsic asymmetry between attack and defense phases, AI/ML demonstrated to be effective in detecting network-wide threats and improving the security of IoT frameworks (Elsadig and Gafar, 2022) even if additional vulnerabilities may arise (Caviglione et al., 2023). However, the very recent surge of malware targeting IoT devices using hidden communications has been almost neglected except for the problem of identifying timing channels in SCADA applications (Alcaraz et al., 2019), which are outside the scope of our research.

Hence, this work addresses the problem of detecting network covert channels targeting IoT ecosystems, i.e., devices compromised by an attacker that remotely exchanges data while remaining unnoticed. In more detail, to identify such anomalous behavior, we propose a Deep Learning (DL) technique adopting an unsupervised incremental learning approach based on

an ensemble of neural networks. This allows for learning effective detection models when a labeled training set is not available and can be incrementally updated to make the deployment feasible on resource-constrained IoT devices. The contribution of this work is two-fold: *i*) the design of an AI-based framework to spot communications hidden in the TTL field of the header of IPv4 packets, and *ii*) an extensive comparison of different neural architectures when handling real traffic traces. Since middleboxes for protecting IoT ecosystems are often implemented in resource-constrained devices (e.g., home gateways), emphasis has been put on evaluating the time requirements of the various AI approaches.

This paper largely extends the work presented in (Cassavia et al. (2022)). Compared to such previous research, the present work has the following novel contents: *i*) it considers many combination strategies for merging the outputs of the various ensembles, even with varying sizes, *ii*) it compares the performances of different baseline architectures, *iii*) it analyzes the convergence of AI methods deployed to spot the presence of hidden network communications, *iv*) it further investigates the behavior of the proposed deep ensemble model when dealing with a scarce amount of data, *v*) it evaluates the applicability of ML-capable frameworks in realistic IoT settings, especially in terms of the time needed to process the data for detecting the hidden transmission, and finally *vi*) it extends the original attack model by introducing an "advanced" threat actor able to improve its stealthiness via a suitable encoding.

The rest of the paper is structured as follows. Section 2 reviews past research works on hidden communications in IoT environments, whereas Section 3 introduces the considered attack scenario and threats. Section 4 showcases the detection approach based on different neural architectures, and Section 5 discusses numerical results. Finally, Section 6 concludes the paper and outlines future research directions.

2 Related work

As hinted, the most recent IoT ecosystems are characterized by a complex interplay of devices, communication technologies, network protocols, and software layers. As a consequence, their attack surface is difficult to outline, and vulnerabilities range from misconfiguration to zero-day-based exploits (Noor and Hassan, 2019). Despite the wide array of offensive opportunities, network connectivity is still the preferred attack vector. For instance, a threat actor can remotely inject malicious routines, gather traffic to infer habits of users, or orchestrate compromised devices for launching Denial of Service (DoS) campaigns (Caviglione et al., 2018; Caputo et al., 2020). As an example of the magnitude of the impact of hidden communication techniques targeting IoT deployments, the work (Velinov et al., 2019) introduces thirteen covert channels exploiting the Message Queuing Telemetry Transport (MQTT) protocol at the basis of the ubiquitous publish-subscribe paradigm. The motivation for assessing such a protocol is rooted in the availability of about 50, 000 MQTT servers, which could be accessed via the Internet without any password. Among the various hiding schemes, an attacker could encode and transmit secret information by overwriting unused protocol fields or by modulating the timing/ordering of topics exchanged by IoT nodes (e.g., subsequent readings of a sensor). In this context, the most popular detection technique relies on the "compressibility" metric, i.e., the more the timestamps of messages are regular, the higher would be the compression ratio of their textual representation (Cabuk et al., 2004). Even if machine learning approaches can be deployed to detect information cloaked in MQTT conversations, the research community is still focused on creating suitable datasets to model more canonical network attacks such as packet flooding, slow-DoS, and brute-force authentications (Vaccari

et al., 2020). Moreover, IoT devices and gateways are often resource-constrained; hence preventing the abuse of a network behavior is preferable to performing detection/mitigation at run-time. For instance, if IoT nodes are devoted to sensing the environment, their update frequency can be forced to be fixed, thus voiding artificial skews in the produced network traffic carrying the measurement (Ho, 2019). Another early line of defense impairs the attacker when trying to modulate the evolution of network traffic, including the envelope of throughput. In this case, traffic normalization or the identification of well-defined "hiding patterns" revealed to be effective tools (Frolova et al., 2021; Wendzel et al., 2021).

The tight coupling of IoT technologies with industrial, health, and cyber-physical applications has also spawned a vivid research area dealing with the creation of air-gapped channels. In this case, a hidden communication path is created by abusing a physical behavior, which can be remotely sensed by the attacker to infer information. For instance, light sources (e.g., a smart light bulb) can be adjusted to encode a signal without causing visible alterations (Jung et al., 2022). Even if AI can be used to reveal anomalous patterns, the most effective defensive approach concerns the adoption of trusted hardware (e.g., to avoid inoculation of firmware capable of altering the physical behavior of an IoT node) or shielding mechanisms (Carrara and Adams, 2015). Yet, addressing this type of channel is definitely outside the scope of our work.

Concerning the use of AI/ML for detecting hidden communications, the survey in (Elsadig and Gafar (2022)) offers a relevant corpus of works focusing on emerging scenarios, such as threats targeting the IPv6 protocol, Voice over Long Term Evolution services, and IoT deployments. Unfortunately, mechanisms specifically tailored for the peculiarities of the IoT have been largely neglected in favor of more general frameworks. Even if reusing network- or protocol-agnostic techniques could work in terms of accuracy (e.g., the ability to detect traffic flows containing cloaked contents), this could be unfeasible in realistic deployments. For instance, in IoT settings, the detection often happens at the border of the network *á-la* edge, or data could not be sent or processed in a centralized manner due to encryption or privacy constraints. In this perspective, the work (Thakkar and Lohiya, 2021) investigates issues and challenges when ML/DL are used for intrusion detection in IoT scenarios. Alas, the work does not consider covert communications, but it solely addresses standard threats, such as worms, phishing attempts, DoS attacks, and diffusion of trojans. The work (Nowakowski et al., 2021) is a notable exception since it proposes to use ML and data mining techniques to exploit the hierarchical organization of frequent sets to reveal communications hidden in the traffic of IoT nodes. Even if the paper also considers information nested in the TTL field of IPv4 (along with other cloaking techniques targeting TCP and HTTP traffic), it does not provide insights on the feasibility of the approach when adopted in production-quality scenarios, privacy and scalability implications, or computational requirements. Moreover, such a framework utilizes a mechanism for handling raw-data-specific representations rather than general autoencoders to directly observe a protocol feature of interest.

Thus, to the best of the knowledge of the authors, the only works leveraging the adoption of autoencoders while considering the benefits of an incremental deep ensemble are (Cassavia et al. (2022)) and (Guarascio et al. (2022)). However, they are preliminary investigations and only focus on a very limited scenario, both in terms of neural architectures and attack models. Summing up, Table 1 reports the various techniques already discussed in the literature, including their relevance with respect to the considered IoT scenario and the detection/mitigation methodology.

Table 1 Analysis of past works dealing with hidden communications in IoT ecosystems and the proposed/used techniques to tackle the considered problem

Reference	Considered Scenario	AI based	Detection or Mitigation Strategy
Cassavia et al. (2022)	Covert channels in IoT nodes	✓	Ensemble of sparse autoencoders
Elsadig and Gafar (2022)	IoT covert communications are not addressed for detection	✓	Survey with many detection mechanisms, e.g., SVM, LSTM, and KNN
Guarascio et al. (2022)	Covert channels in IoT nodes	✓	Sparse autoencoders
Frolova et al. (2021)	General network-wide covert communications	-	Traffic manipulation and engineering to disrupt or prevent hidden transmissions
Thakkar and Lohiya (2021)	Covert channels are not considered. Attack models for IoT nodes are tojans, DoS, etc.	✓	Survey covering techniques like ML/DL, k-Nearest, Random Forest and Ensemble Learning
Nowakowski et al. (2021)	Covert channels and threats cloaking data in TTL of IPv4, as well as in various parts of TCP and HTTP traffic	✓	The Driverless AI platform is used to prevent feature engineering and spot anomalous flows from a threat-specific raw data representation
Vaccari et al. (2020)	Not focusing on hidden communications but on general attack templates	-	A dataset to perform realistic experiment is provided
Alcaraz et al. (2019)	Timing channels in SCADA IoT applications	-	By-design elimination or impairment via additional delays
Noor and Hassan (2019)	Generic hidden communications for IoT nodes via the MQTT protocol	-	Use of different state-of-the-art and attack dependent metric, including the "compressibility" (see, Cabuk et al. (2004))
Ho (2019)	Not related to hiding data in network traffic, but in data produced by sensors	-	By-design elimination or "constraints" in the timing behaviors of data managed by IoT nodes

3 Attack scenario and threat modelling

In this section, we first introduce the attack scenario and then we discuss the dataset built to model malware covertly exchanging information within an IoT ecosystem.

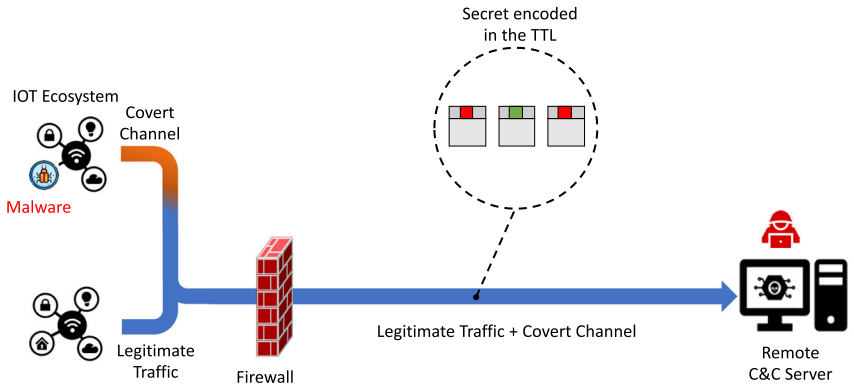
3.1 Attack scenario

In this work, we investigate the problem of revealing malware communications targeting the network traffic of IoT ecosystems also in the presence of different offensive behaviors. Without loss of generality, we assume that the attacker has already taken control of the IoT node, e.g., by using a well-documented CVE or by dropping a payload via phishing/spear-phishing (Gupta et al., 2017; Blinowski and Piotrowski, 2020). The reference scenario is depicted in Fig. 1. In more detail, Fig. 1a depicts the general attack model where a threat actor can abuse an IoT node to create a network covert channel. The hidden communication path can be used to exfiltrate information towards a remote server or to exchange commands without being spotted by an intrusion detection system or blocked by a firewall. Despite almost any protocol or traffic feature can be manipulated for hiding data, the resource-limited nature of IoT nodes poses constraints on the complexity of the covert channel (Zander et al., 2007). Hence, the hiding mechanism should be simple to not disclose the presence of the malware due to additional delays, anomalous energy consumption, or intermittent connectivity when an IoT node is remotely operated. To cope with such requirements, we consider a malware cloaking data within the TTL field of the IPv4 header (Skowron et al., 2020; Zander et al., 2007). Specifically, the malicious software manipulates the TTL of IPv4 traffic generated by the compromised IoT node to transport arbitrary information. To not appear suspicious, the malware should not directly write the secret data in the field (Zander et al., 2006). Rather, it should encode the bits `1` and `0` by increasing or decreasing the observed TTL of a suitable threshold or by exploiting the most popular values as "high" and "low" signals. The attacker should then design a proper information hiding mechanism by taking into account the "clean" traffic conditions and select accordingly how bits are encoded. To this aim, the threat actor is expected to perform a reconnaissance campaign to gather traffic information (e.g., by fingerprinting IoT devices) or deploy a packet sniffing routine for monitoring local network conversations (Mazurczyk and Caviglione, 2021; Zorawski et al., 2023). To have a general setting, we consider the two use cases depicted in Fig. 1b. In more detail:

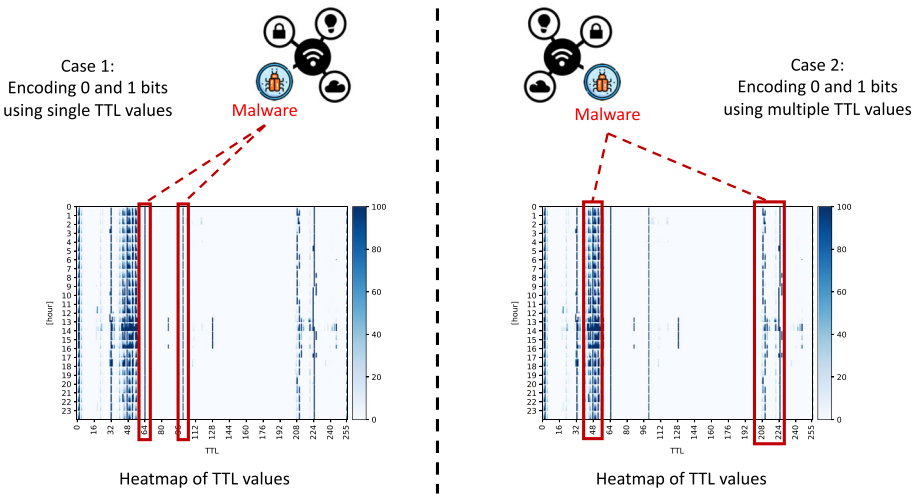
- **Case 1** (Naive Encoding Scheme): the malware encodes the bit `1` and `0` by selecting a single TTL value for each bit;
- **Case 2** (Advanced Threat Scheme): the malware encodes the bit `1` and `0` by randomly selecting a single TTL value from three alternatives for each bit.

Despite the specific reconnaissance and encoding mechanism, the final result could be organized into a heatmap. Figure 1b. depicts the "distribution" of TTL values obtained in a real IoT setup[1] during an observation window of 12 hours. As shown, the values for the TTL aggregate in two main ranges $32-64$ and $208-224$. Other values have an intermittent behavior, e.g., datagrams with a TTL equal to 128 are present only for 3 hours, thus limiting the duration of a possible covert communication. To avoid that it could be easily detected, the data hidden in the TTL should not represent an anomaly, i.e., disrupt the clean traffic conditions. Therefore, when the malware deploys a naive mechanism (see Case 1 in

[1] Heatmaps have been computed by using the 24-hour slice of data captured during Sept. 22-23, 2016, whereas for the performance evaluation, we used traces containing traffic collected during Sept. 22-29, 2016.



(a) A malware sends data towards a remote C&C facility via a network covert channel in the TTL field of IPv4 traffic.



(b) The malware encodes data using single or multiple TTL values (Case 1 and Case 2).

Fig. 1 Reference attack scenario

Fig. 1b), the bit `0` is encoded by using a TTL value equal to 64, whereas the bit `1` is encoded by using 100. Differently, a more advanced scheme tries to reduce the footprint left in the traffic, e.g., in terms of anomalous distributions or possible signatures. Hence, at each bit sent, the malware can slightly adapt its encoding scheme by selecting a different TTL value (see, Case 2 in Fig. 1b). As a result, the bit `0` is encoded by using a TTL value among 44, 56, and 57, whereas the bit `1` is encoded by using a TTL value among 210, 223, and 224.

3.2 Datasets for modelling the covert channel

To model the malware hidden communications and to quantify the performance of our approach, we built a benchmark dataset starting from real traffic traces described in

(Sivanathan et al. (2018)). To avoid burdening the data, we removed IPv6, ICMP, DNS, and NTP conversations, as well as multicast/broadcast traffic. This is not a limitation: for instance, broadcast traffic is mainly link-local and thus not suitable for creating Internet-wide covert channels. Moreover, many vendors are slowly migrating to more secure protocols, such as the DNS Security Extensions. To prevent unwanted signatures, we also removed traffic generated by non-IoT devices, e.g., smartphones and laptops. The final dataset contains the traffic flows exchanged by 28 different IoT nodes (e.g., smart speakers and home hubs) over an entire week.

A realistic attack template has been obtained by modeling the presence of a malware compromising a specific IoT device. Typically, this requires exploiting a CVE for granting access to the device or by leveraging a configuration error (e.g., weak/default credentials) (Neshenko et al., 2019). In this work, we considered a threat abusing a smart camera to smuggle sensitive data towards a remote host controlled by the attacker. Specifically, the dataset borrowed from (Sivanathan et al. (2018)) allowed to take into account a local network populated with a mixed set of devices, controllers, and appliances communicating both directly or through the Internet. To model the threat depicted in Fig. 1b, we assumed that the Dropcam IoT device present in the original dataset has been under the control of the malware for 3 days. To create the covert communication, we rewrote the flows generated by the Dropcam directly in the original traffic captures by using the pcapStego tool (Zuppelli and Caviglione, 2021), and we implemented the two encoding schemes discussed in Section 3.1. Since we want to investigate Internet-wide covert communications hidden in the TTL field of IPv4 packets, we do not consider the presence of additional middleboxes or NAT devices, which can be considered "transparent" for our threat model.

To not make the detection trivial, we always used the most frequent/observed values present in the traffic traces. Specifically, for the Naive Encoding Scheme of Case 1 and for the Advanced Threat Scheme of Case 2, we used the values depicted in the heatmaps of Fig. 1b discussed in Section 3.2.

To prevent that bursts of manipulated datagrams would reveal the presence of the channel, we randomly interleaved packets containing hidden data with legitimate/unaltered ones (Zander et al., 2006). The secret information transmitted by the malware via the covert channel has been modeled with randomly-generated strings: this represents an attacker using some obfuscation technique, such as encryption or scrambling (McLaren et al., 2017). To bear with the exfiltration of several contents (e.g., username+password pairs or configuration files), each day of attack contains a different volume of hidden information, i.e., we considered the exfiltration of 69, 80, and 64 kbit of data. As a result, for both the encoding cases, the compromised IoT node manipulates the 18%, 1%, and 12% of the overall daily traffic, respectively. We point out that, since the resulting traffic traces are a modified subset of the data provided in (Sivanathan et al. (2018)), they have not been publicly released. Yet, they are available upon request, e.g., to advance the research in the field of covert communications targeting IoT ecosystems.

4 Deep ensemble learning scheme

In this section, we illustrate the approach based on unsupervised learning of deep neural network ensembles to spot hidden malware communications in IoT traffic. The main benefit relies on the capability of the models to raise alarms also on never seen attacks: this is a frequent scenario since covert channels are often undocumented and unknown *a priori*.

Although our primary focus is revealing covert communications hidden within the TTL of IPv4 packets, our results can be extended to other similar attack scenarios. Moreover, as it will be shown, the use of Autoencoders (AEs) allows to consider different encoding mechanisms (see Case 1 and Case 2 illustrated in Section 3) with the same neural architecture. Another important aspect concerns the lack of labeled data. In fact, advanced persistent threats or malware endowed with cloaking mechanisms often remain undetected for long time frames or not recognized at all. Therefore, a fully unsupervised neural approach based on AEs is the best option, especially since it does not require labeled training data, often unavailable. We first illustrate the detection mechanism for a single model, and then we describe how it can be extended to learn effective and scalable ensemble models.

4.1 Detection through a single autoencoder

The idea behind our solution is to adopt a neural encoder-decoder architecture trained against traffic data. Basically, an autoencoder is an unsupervised (i.e., trained without any information concerning the nature of the attack/normal behavior) neural network model performing two main operations. First, it compresses the input data (i.e., a number of statistics computed over the traffic generated by the IoT network and described in Section 5.1) within a latent space. Then, it reconstructs the original information provided as input. In our setting, the model is only trained against the normal behavior. The underlying intuition is that the legitimate input data should be (almost) correctly reconstructed by the autoencoder. In other words, the encoding/decoding phases should not introduce a heavy distortion in the output. By contrast, outliers and anomalous values in the input will yield a deviant output.

The usage of the reconstruction error as a measure of outlierness to discover abnormal behaviors has already been proposed in the literature, but the adoption of unsupervised techniques (and in particular of encoder-decoder architectures) for revealing covert channels is substantially unexplored (Ahmad et al., 2021; Darwish et al., 2019; Elsadig and Gafar, 2022). As discussed in (Hinton and Salakhutdinov (2006); Bengio et al. (2006)), autoencoders are considered a valid solution to the problem of effectively summarizing the information of a given input into a low-dimensionality representation. In essence, these neural network models aim at yielding a duplicate of the input as output.

In this work, we employ the Sparse U-Net neural model shown in Fig. 2. Basically, it includes two main components, named *Encoder* and *Decoder*, respectively. Let $\mathbf{x} = \{x_1, \ldots, x_N\}$ be a set of numeric features (in our case, a number of traffic flow statistics computed for a time slot). The former sub-network is devoted to mapping the input data with a latent space (*encoding*), i.e., learning a function $\mathbf{z} = enc(\mathbf{x})$, whereas the second one yields the overall network output by reconstructing the input from the features extracted by the encoder $\mathbf{y} = dec(\mathbf{z})$ (*decoding*). Gradient descent is employed to learn the model weights by minimizing a suitable loss function. In our approach, the *Mean Square Error*, i.e., $Loss_{MSE}(\mathbf{x}) = \frac{1}{N}\sum_i \|x_i - y_i\|_2$, is used as loss.

Notably, the architecture of Fig. 2 exhibits two main differences w.r.t. 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 flows cloaking secret data often exhibit slight differences compared with normal behaviors. Both the encoder and decoder are composed of M hidden layers, therefore we adopted a symmetric architecture. The choice of skip connections simplifies the learning process of the network by providing as input to each layer

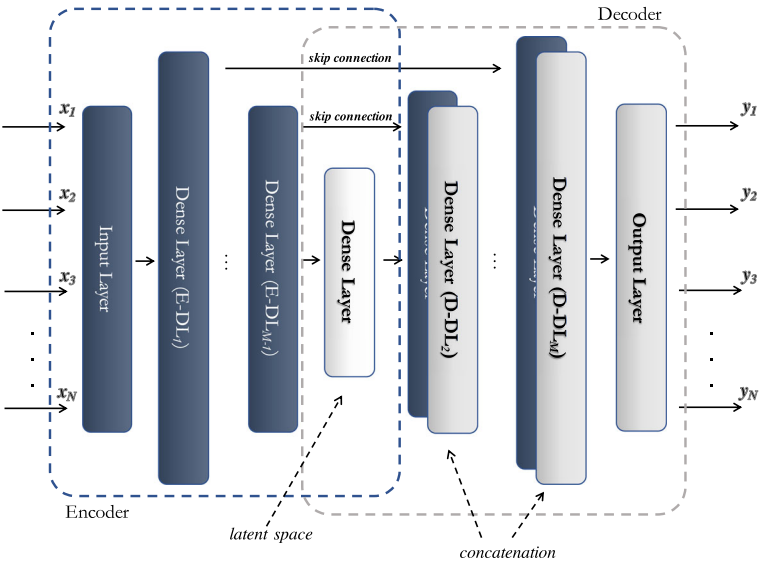


Fig. 2 The Sparse U-Net architecture for detecting covert channels in the TTL

of the decoder ($D\text{-}DL_i$), except for the shared latent space, both the previous ($D\text{-}DL_{i-1}$) and the correspondent encoder layer ($E\text{-}DL_{M-i+1}$). As regards the Sparse Layers, they are used to generate a wider number of discriminative features, which allow to extract a more representative latent space.

Figure 3 illustrates the detection process for covert channels targeting the TTL field of IPv4 datagrams. This mechanism can be co-located within the firewall depicted in Fig. 1a or implemented through a dedicated appliance. Without loss of generality, we assume to monitor an "infinite datastream", i.e., the traffic of the various IoT nodes feeds our detection mechanism in a continuous manner. At pre-fixed time intervals (i.e., a time slot in Fig. 3), we compute several statistics to describe the behavior of the TTL fields composing the aggregate traffic flow. Specifically, we compute metrics such as the minimum, average, maximum,

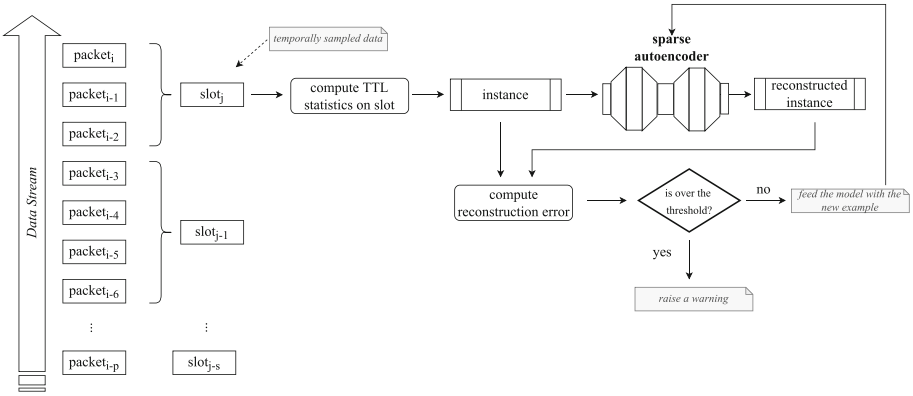


Fig. 3 Detection mechanism for revealing the presence of network covert channels

or different percentiles, starting from TTL values gathered in a time slot. Preliminary, an autoencoder, pre-trained only against legitimate data flows, is used to reproduce the statistics, then the reconstruction error is calculated for the current example as the MSE between $\mathbf{x}$ and $\mathbf{y}$. If the error is smaller than a given *outlierness threshold*, the current data are labeled as "normal" and update the model, otherwise a warning is raised.

In the context of AE-based anomaly detectors, different techniques are commonly employed to compute anomaly scores. As mentioned above, these models primarily operate by learning to reconstruct input data and then quantifying the dissimilarity between the original input and the reconstructed output. Common methods for computing anomaly scores include Reconstruction Error, Probability Density Estimation, Clustering-based Approaches, and Ensemble Techniques. Additionally, an essential aspect of anomaly detection is determining the threshold for anomaly scores, which significantly impacts model performance. Several thresholding strategies are available, including Fixed Thresholding, Statistical Methods, Quantile-Based Thresholding, Density Estimation, Precision-Recall Curve Analysis, Domain-Specific Knowledge Incorporation, and Dynamic Thresholding. Although more sophisticated techniques may be used, we mainly focused on demonstrating the robustness of our approach by evaluating its performance across different threshold values, highlighting its effectiveness under various scenarios.

As it will be detailed in Section 5.3, the collection of TTL values can be done by using limited computing resources to prevent a decay in the overall traffic performance. A viable approach may require to instrument gateways with a lightweight and non-invasive software layer (Repetto et al., 2021).

4.2 Learning and combining different detectors

A main limitation of the described approach relies on the necessity to learn the neural network model against the whole training set (that could be unfeasible in IoT networks with tight computational resources). Moreover, in real scenarios, the limited resources of the device where the detector is deployed and the presence of concept drifts in the observed behaviors (Folino et al., 2019) can affect the predictive performances of the autoencoder. To mitigate such issues, we devised an incremental learning scheme based on an ensemble of encoder-decoder architectures shown in Fig. 4. Basically, we consider the case where only a limited number of training examples $\mathcal{D}$ can be gathered and stored in a data chunk (denoted as D_i in the figure). Our ensemble solution relies on building up a series of k base Deep Neural Network detectors (denoted as $M_i, M_{i-1}, \ldots, M_{i-k}$) sharing the same neural architecture described above. These autoencoders are trained from disjoint *data chunks* (denoted as $D_i, D_{i-1}, \ldots, D_{i-k}$, respectively), which are fed with data instances gathered in different temporal intervals. Specifically, the incremental learning process adopted in our solution is loosely inspired by the work (Folino et al., 2021) proposing an ensemble-based deep learning approach trained on disjoint data chunks. Differently from this work where each model is trained independently, in our solution the model M_i is trained (i.e., fine-tuned) from the weights of the model M_{i-1} and the sample D_i. The same procedure is also applied to all the models composing the ensemble. In this way the detection model will be able to gradually adapt to normal concept drifts that can occur, for instance, due to the deployment of new devices in the network that can modify the network statistics. This approach can also reduce the risk of *catastrophic forgetting* (Parisi et al., 2019) that affects DL models (and also

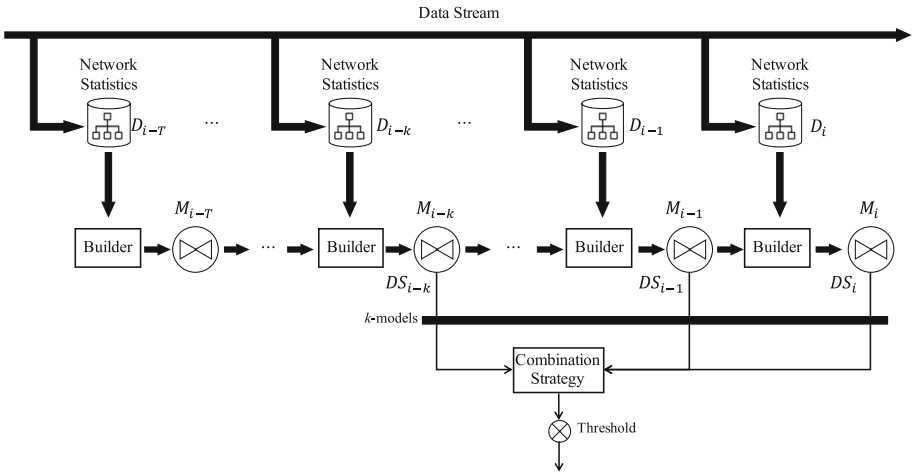


Fig. 4 Incremental Deep Ensemble model approach

different types of shallow architectures) when learned incrementally. Specifically, catastrophic forgetting refers to the phenomenon where a neural network trained on a new task or dataset tends to forget or overwrite knowledge it acquired during previous training on different tasks or datasets. In the literature, different solutions have been proposed to address this issue effectively, e.g., in (Faber et al. (2023)), the authors blend a lifelong change point detection with a suitable model update strategy to learn robust anomaly detectors, whereas (Li et al., 2022) use an experience replay mechanism based on self-imitation learning. While not the primary focus of our work, we attempted to address the issue of catastrophic forgetting by combining multiple models trained on data collected from distinct time windows. This approach allows us to mitigate the risk by preserving information about various normal behaviors. Then, the final anomaly score is computed by adopting a non-trainable combination strategy. By denoting the reconstruction error as Deviance Score (DS) (see, Fig. 4), we then consider the median, max, and average value of the k reconstruction errors $\{DS_i, \ldots, DS_{i-k}\}$ yielded by the base models.

5 Experimental investigation

In this section, we present a series of experiments for assessing the detection capability of our approach against the scenarios and threat models of Section 3. Tests were carried out to: *i*) evaluate and compare different combination schemes and ensemble sizes, *ii*) compare our neural model with other unsupervised deep architectures, *iii*) analyze the behavior of our ensemble learning approach when a limited portion of training data is available, and finally *iv*) evaluate the convergence rate of the proposed model and the data processing time in the deployment stage.

In Section 5.1, we supply more insights on the used datasets, parameters, and neural architectures, whereas in Section 5.2 we discuss how we compare different base models, ensemble parameters and robustness to lack of data. Finally, in Section 5.3, we present the results of the convergence analysis and the efficiency of the proposal.

5.1 Datasets, parameters and evaluation metrics

5.1.1 Datasets and evaluation protocol

To perform the experiments, we prepared the datasets starting from the main features of the traffic illustrated in Section 3.2. In particular, for each time slot we extracted the following fields: a progressive timestamp, the number of incoming packets, 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). As our approach employs a "slotted" mechanism (see Fig. 3), we considered a time slot with a duration of 5 seconds. In the following, with the term "instance" we refer to a tuple composed of the extracted fields and the label observed within the chosen time slot. Data have been partitioned in training and test sets using a temporal split. The data gathered in the first 96 hours only contain legitimate traffic and has been used for the learning phase of the ensemble. The remaining 72 hours contain the traffic used to generate the two test sets for the attack strategies of Case 1 and Case 2 (see Fig. 1b). As a result, the training set is composed of 69, 116 legitimate instances, whereas the test sets include 51, 837 instances. From now on, we will refer to the test sets as Testset Case 1 and Testset Case 2, respectively.

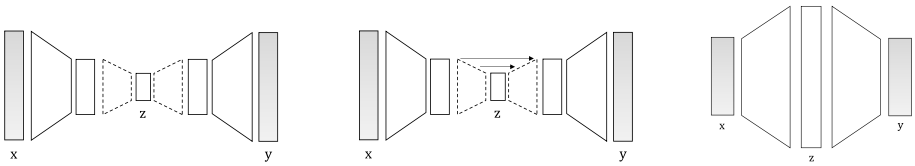
Concerning the class distribution of the two test sets, the number of legitimate and malicious instances between the Testset Case 1 and Testset Case 2 slightly differ. In more detail, the number of legitimate instances for Testset Case 1 is 34, 306, while it is equal to 34, 272 for Testset Case 2. The malicious instances slightly differ as well, and turn out to be 17, 531 and 17, 565 for Testset Case 1 and Testset Case 2, respectively. Such a behavior can be ascribed to the the attack strategies used in Case 1 and Case 2, which require to modify the traffic with different proportions. Finally, input data are further pre-processed through a normalization procedure: a *MinMax* normalization has been adopted to map each feature in the range $\{-1, 1\}$ to improve the stability of the learning process.

5.1.2 Parameters and competitors

To assess the quality of the proposed approach in detecting network covert channels within traffic aggregates, we developed a prototype with Python and TensorFlow[2]. As described in Section 4, the base model composing the ensemble is a specific autoencoder architecture. The *Encoder* is composed of 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 it is instantiated as a dense layer (shared between the encoder and the decoder) including 4 neurons and equipped with a ReLU activation function. Symmetrically, the *Decoder* consists of three fully-connected dense layers with the same dimensions and activation function. The output of the model is yielded by a Dense Layer with the same size as the input and equipped with a *Tanh* activation function since the input is normalized in the range $\{-1, 1\}$.

The analysis of the ensemble performances has been done by varying different values of k. In particular, we tested the approach by including the last 3 and 5 base models while we considered a data chunk size of $\sim 5,000$ instances. Notably, k can influence the capability of the model to keep the memory of past behaviors.

[2] TensorFlow machine learning library. Available online at: https://www.tensorflow.org/ [Last Accessed: May 2023].



(a) *DAE* (b) *U-Net* (c) *Sparse-AE*

Fig. 5 Autoencoder-based architectures used as competitors

As regards the comparison among different base model architectures, we tested our base U-Net-like model against the alternative AE-based models sketched in Fig. 5 that are widely adopted in the literature for similar tasks. Specifically:

- the *Deep Autoencoder* (from now on referred as *DAE*), depicted in Fig. 5a. The encoder is instantiated with two fully-connected dense layers with 9 (input size) and 8 neurons, respectively. The third layer is the latent space and it is instantiated as a dense layer with 4 neurons. The decoder expands the latent representation (still of size 4) through the same (but inverted) 2-layer sequence as the encoder;
- the "unsparse" version of our architecture (from now on referred to as $U\text{-}Net$), depicted in Fig. 5b. Both its encoder and decoder parts are instantiated with two fully-connected dense layers with 9 (input size) and 8 neurons, respectively. The middle (third) layer still consists of 4 neurons;
- the *Sparse Autoencoder* (from now on referred as *Sparse AE*), depicted in Fig. 5c. It features a single-hidden-layer shallow architecture with 32 neurons.

Differently from the solutions described in (Abderrahim et al. (2020); Corizzo et al. (2021)), we adopted a simplified U-Net architecture that only integrates skip connections between corresponding layers of the encoder and decoder. This design choice is mainly due to the nature of the data: in our work, we consider flat data that summarize the underlying distribution, while in the original work, it was used to process images where zooming capabilities can help to improve the performances. As regards its capability in terms of pattern extraction, the main idea is that the model should be able to map the original input to a latent space where similar examples (i.e., tuples with similar traffic patterns) are grouped (in a more efficient way by reducing the number of epochs required for the convergence). Then, the model will tend to map exceptional examples far from normal ones, allowing to reveal covert attacks. All the models have been trained over 16 epochs with a batch size of 16, whereas *Adam* is adopted as the optimizer with a learning rate $lr = 1e^{-4}$.

Concerning the anomaly score, it is empirically estimated by computing the reconstruction error for each instance contained in the training set and extracting the values corresponding to 90^{th}, 95^{th}, and 99^{th} percentiles. This simple strategy allows for easily computing an effective threshold without requiring a large number of resources. Investigating more sophisticated methods to estimate the anomaly threshold is part of our ongoing research, for instance to automatically match the requirements of our scenario (Angiulli et al., 2017).

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

5.1.3 Evaluation metrics

The effectiveness of our approach has been tested by considering some quality metrics widely-adopted in the literature dealing with attack detection tasks (Cassavia et al., 2023, 2022). 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.

Based on these values, we can compute the following metrics:

- *Accuracy*: defined as the fraction of cases correctly classified, i.e., $\frac{TP+TN}{TP+FP+FN+TN}$;
- *Precision* and *Recall*: metrics used to estimate the detection capability of a system since they provide a measurement of accuracy in identifying attacks and avoiding false alarms. Specifically, Precision is defined as $\frac{TP}{TP+FP}$, while Recall as $\frac{TP}{TP+FN}$;
- *F-Measure*: summarizes the model performance and it is the harmonic mean of Precision and Recall. The F-Measures is particularly beneficial when the class distribution is imbalanced and represents the best trade-off between Recall and Precision. Hence, it is often preferred over other metrics when one searches for a unique criterion assessing the goodness of a classification approach.

5.2 Numerical results

To showcase the effectiveness of our approach, we examine the micro-averaging of the previously-defined metrics. Specifically, we will provide a comparison among performances for different schemes and sizes, the impact of the various different base models, and a sensitivity analysis accounting for the scarcity of training data.

5.2.1 Comparing different ensemble schemes and sizes

In this first suite of experiments, we investigate how the combination scheme and the ensemble size influence the predictive performances of the approach. Table 2 reports the results obtained by varying three parameters on the Testset Case 1: the combination strategy, the number of base models, and the threshold used to distinguish between anomalous and normal traffic flows. Specifically, for any combination scheme and ensemble size, we can observe that the usage of a looser threshold (e.g., the 90^{th} percentile) allows for improving the probability of detection (i.e., the Recall) but at the price of a higher number of false alarms. By contrast, a higher threshold (e.g., the 99^{th} percentile) allows for limiting the number of FP, but a lesser Recall value is obtained. In addition, the best result with the higher value of F-Measure is obtained with an ensemble size equal to 3 and by considering the threshold value corresponding to the 99^{th} percentile and the median as a combination strategy. The slight reduction of the predictive performances when increasing the ensemble size appears mainly due to the evolving nature of traffic characterizing IoT ecosystems. Indeed, asynchronous activations of nodes, external triggers, or periodical synchronizations account for broad changes in traffic conditions. Therefore, recent data could be more informative for revealing an attack as the "past history" may not capture the behavior of the actual network traffic conditions.

The same evaluation has been conducted on Testset Case 2 and reported in Table 3. The best result (i.e., the higher value of F-Measure) is achieved again with an ensemble size equal to 3. However, since Case 2 introduces an "advanced" encoding scheme based on multiple TTL values, the adoption of a looser threshold is crucial for its identification: indeed, we

Table 2 Experimental results for different combination schemes and ensemble sizes against Testset Case 1

Ensemble Size	Strategy	Detection Threshold	Accuracy	Precision	Recall	F-Measure
3		90^{th} perc.	0.894	0.771	0.979	0.863
	median	95^{th} perc.	0.947	0.902	0.948	0.924
		99^{th} perc.	**0.955**	**0.950**	0.915	**0.932**
		90^{th} perc.	0.894	0.772	0.974	0.861
	max	95^{th} perc.	0.946	0.901	0.945	0.922
		99^{th} perc.	0.948	0.941	0.902	0.921
		90^{th} perc.	0.892	0.767	0.980	0.860
	avg	95^{th} perc.	0.947	0.901	0.947	0.924
		99^{th} perc.	0.952	0.946	0.910	0.928
5		90^{th} perc.	0.890	0.764	0.977	0.858
	median	95^{th} perc.	0.933	0.863	0.954	0.906
		99^{th} perc.	0.952	0.944	0.911	0.927
		90^{th} perc.	0.893	0.767	**0.981**	0.861
	max	95^{th} perc.	0.939	0.878	0.953	0.914
		99^{th} perc.	0.941	0.944	0.878	0.910
		90^{th} perc.	0.891	0.765	**0.981**	0.859
	avg	95^{th} perc.	0.941	0.884	0.951	0.916
		99^{th} perc.	0.951	0.942	0.912	0.927

The best results are reported in bold

Table 3 Experimental results for different combination schemes and ensemble sizes against Testset Case 2

Ensemble Size	Strategy	Detection Threshold	Accuracy	Precision	Recall	F-Measure
3	*median*	90^{th} perc.	**0.811**	0.715	**0.732**	**0.724**
		95^{th} perc.	0.728	0.735	0.308	0.434
	max	90^{th} perc.	0.779	0.687	0.640	0.663
		95^{th} perc.	0.727	0.735	0.307	0.433
	avg	90^{th} perc.	0.803	0.706	0.720	0.713
		95^{th} perc.	0.731	**0.742**	0.315	0.442
5	*median*	90^{th} perc.	0.786	0.688	0.671	0.680
		95^{th} perc.	0.734	0.703	0.374	0.488
	max	90^{th} perc.	0.788	0.692	0.675	0.683
		95^{th} perc.	0.728	0.707	0.336	0.455
	avg	90^{th} perc.	0.801	0.702	0.717	0.710
		95^{th} perc.	0.732	0.723	0.338	0.461

The best results are reported in bold

can observe that the performances dramatically decrease when the threshold value increases (e.g., 95^{th} percentile).

5.2.2 Comparing different base models

Another relevant part of the analysis focuses on demonstrating the quality of our Sparse U-Net architecture over other unsupervised neural models and the benefits of the ensemble model over a single one. We then compared our solution against the base models described in Section 5.1. Specifically, Table 4 reports the results of our experimentation by comparing the performances of the baseline architectures w.r.t. the ensemble model. To not burden the discussion, we only report the results for an ensemble model with a size equal to 3, as it achieves the best performances. The performance metrics are computed by ranging different sensitivity thresholds. First, we can observe that the adoption of the ensemble strategy improves the performances of the base model (i.e., the Sparse U-Net) for each threshold value. Moreover, the Sparse U-Net outperforms the other AE-Based architectures in terms of both Accuracy and F-Measures, for each considered threshold.

5.2.3 Sensitiveness to the scarcity of training data

Unlike supervised methods, unsupervised ones do not require a costly labeling process performed by the expert to yield labeled data for the training phase. To work as expected, unsupervised models still need to approximate the normality distribution as closely as possible since every behavior that sensibly differs from normality is considered malicious. Thus,

Table 4 Comparison between our deep ensemble-based model and baseline architectures against Testset Case 1

Model Type	Detection Threshold	Accuracy	Precision	Recall	F-Measure
Sparse U-Net	90^{th} perc.	0.882	0.743	**0.993**	0.850
	95^{th} perc.	0.921	0.822	0.976	0.893
	99^{th} perc.	0.936	0.942	0.865	0.902
DAE	90^{th} perc.	0.869	0.724	**0.993**	0.837
	95^{th} perc.	0.910	0.801	0.975	0.880
	99^{th} perc.	0.905	**0.962**	0.750	0.843
Sparse AE	90^{th} perc.	0.875	0.737	0.979	0.841
	95^{th} perc.	0.901	0.795	0.951	0.866
	99^{th} perc.	0.902	0.922	0.778	0.844
U-Net	90^{th} perc.	0.875	0.736	0.982	0.841
	95^{th} perc.	0.907	0.799	0.968	0.876
	99^{th} perc.	0.819	0.853	0.563	0.678
Ensemble (k=3)	90^{th} perc.	0.894	0.771	0.979	0.863
	95^{th} perc.	0.947	0.902	0.948	0.924
	99^{th} perc.	**0.955**	0.950	0.915	**0.932**

The best results between Sparse U-Net and Ensemble are reported in bold, while the best results among the base models are underlined

Table 5 Results for different amounts of training data against Testset Case 1

Training Perc.	Strategy	Detection Threshold	Accuracy	Precision	Recall	F-Measure
25%	median	90^{th} perc.	0.866	0.718	0.995	0.834
		95^{th} perc.	0.905	0.789	0.979	0.874
		99^{th} perc.	0.942	0.924	0.904	0.914
	max	90^{th} perc.	0.873	0.729	0.997	0.842
		95^{th} perc.	0.905	0.789	0.982	0.875
		99^{th} perc.	0.923	0.870	0.908	0.888
	mean	90^{th} perc.	0.869	0.722	0.997	0.838
		95^{th} perc.	0.904	0.786	0.982	0.873
		99^{th} perc.	0.928	0.886	0.902	0.894
Average Performances 25%			**0.902**	**0.801**	**0.961**	**0.870**
50%	median	90^{th} perc.	0.849	0.691	0.998	0.817
		95^{th} perc.	0.903	0.781	0.990	0.873
		99^{th} perc.	0.943	0.901	0.934	0.917
	max	90^{th} perc.	0.862	0.710	0.998	0.830
		95^{th} perc.	0.913	0.803	0.985	0.885
		99^{th} perc.	0.944	0.904	0.932	0.918
	mean	90^{th} perc.	0.853	0.697	0.998	0.821
		95^{th} perc.	0.903	0.783	0.988	0.873
		99^{th} perc.	0.942	0.897	0.935	0.916

Table 5 continued

Training Perc.	Strategy	Detection Threshold	Accuracy	Precision	Recall	F-Measure
Average Performances 50%			**0.901**	**0.796**	**0.973**	**0.872**
100%	median	90^{th} perc.	0.894	0.771	0.979	0.863
		95^{th} perc.	0.947	0.902	0.948	0.924
		99^{th} perc.	0.955	0.950	0.915	0.932
	max	90^{th} perc.	0.894	0.772	0.974	0.861
		95^{th} perc.	0.946	0.901	0.945	0.922
		99^{th} perc.	0.948	0.941	0.902	0.921
	mean	90^{th} perc.	0.892	0.767	0.980	0.860
		95^{th} perc.	0.947	0.901	0.947	0.924
		99^{th} perc.	0.952	0.946	0.910	0.928
Average performances 100%			**0.930**	**0.872**	**0.944**	**0.904**

Bold values represent the average performances for each training size by ranging the main parameters (strategy and threshold)

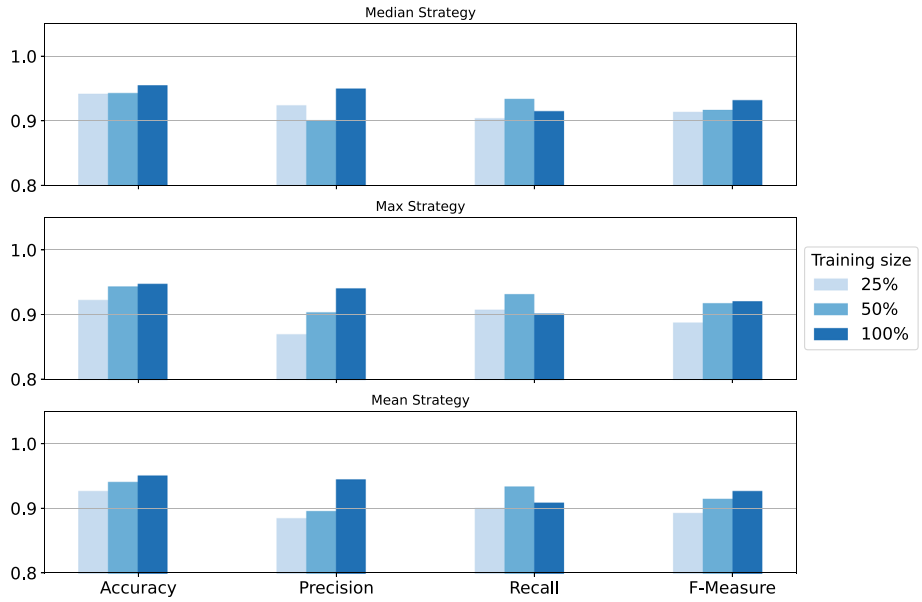


Fig. 6 The evaluation metrics computed for each combination strategy by varying different training sizes

the models must rely on a verified subset of legitimate traffic flows (even if limited) during the training to provide reliable results.

In this respect, we assess the robustness of our approach by ranging different amounts of available training data (namely, 25%, 50%, and 100%). Table 5 reports the experiments with different thresholds, ensemble strategies, and training percentage sizes against Testset Case 1. We can observe that also when reducing the training size to 25% the approach continues to achieve good performances for each metric.

For only the best threshold value (i.e., 99^{th} percentile), we depicted the obtained results in Fig. 6. For each evaluation measure, the lack of data does not dramatically affect the detection capability of the approach; by contrast, the ensemble model (especially for the median strategy) continues to provide reliable predictions.

5.3 Convergence analysis and deployment assessment

We now analyze the model behavior at learning time to investigate whether our architecture allows a fast training by reducing the number of epochs required to learn an effective detection model. Figure 7 depicts the loss behavior over the number of epochs. In particular, for each epoch we computed the loss value by averaging the losses obtained by each model for each chunk in the same epoch. The shading area indicates the standard deviation. As shown, the model requires a limited number of training epochs to converge. This is mainly due to the adoption of the skip connections, which enables a fast convergence of the training algorithm, as expected.

Lastly, an important aspect concerns the feasibility of deploying our AI-based framework in realistic IoT deployments. Even if performing the detection within simple sensors/devices is seldom feasible, a possible solution is to locate the needed layers within the gateways often used to connect and orchestrate the IoT nodes. However, many cost-effective middleboxes

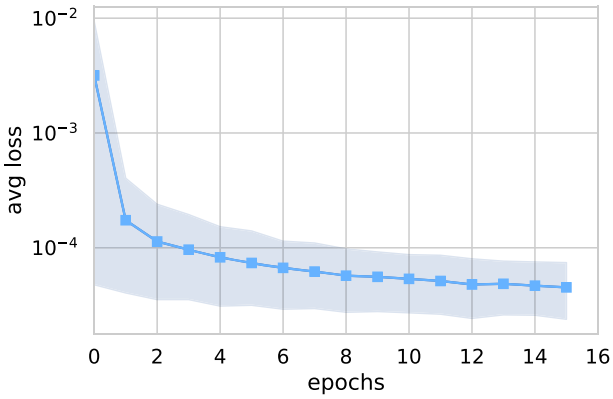


Fig. 7 Per-chunk average loss over the training epochs

are endowed with limited hardware-storage capabilities, thus requiring a lightweight and optimized approach. As regards operations needed to gather information, techniques based on code layering or kernel augmentation are prime candidates (Vieira et al., 2021). In fact, collecting the TTL values with a per-packet granularity usually accounts for an additional transmission delay of $\sim$ 100 ns when using the extended Berkeley Packet Filter and 1 ms with a C implementation exploiting libpcap over commodity hardware (Caviglione et al., 2021). The use of TTL is a good example of a real covert communication mechanism, which directly stores data within a field/feature of network packets (Caviglione and Mazurczyk, 2022). Extending the gathering/detection phase is straightforward as it only requires to collect data from a different portion of the header and compute statistics of a similar complexity. Yet, if hidden communications exploit more advanced schemes, e.g., manipulation of textual entries in DNS queries, the proposed idea needs some additional tweaks, i.e., a suitable inspection mechanism and an effective set of metrics should be searched for. Luckily, real attacks targeting simple IoT nodes rarely employ sophisticated data hiding mechanisms (Mazurczyk and Caviglione, 2015; Neshenko et al., 2019; Noor and Hassan, 2019; Caviglione and Mazurczyk, 2022). The availability of efficient traffic inspection tools and statistics computed by security appliances can further ease the design of the required metrics. A relevant network security advancement concerns the definition of "independent" metrics, i.e., indicators not bounded to a specific field or data hiding mechanism. For instance, the presence of a flow containing a covert communication could be spotted through anomalous jitters or general behaviors of IoT nodes, such as aggressive battery drains. This is still an open problem, which is also part of our future research.

As regards the ML phases of training and inference, the former can be performed offline, while the prediction time is 0.5 ms for a single data batch and 2 seconds for the whole test set. Such footprints open to many design choices. First, the approach can be used for real-time processing of traffic flows within the edge entity in charge of managing the IoT ecosystem. Other optimizations are possible, for instance by taking advantage of the "stateless" nature of our approach. In fact, covert communications are spotted by using information on the overall traffic (grouped in time slots), which prevents the need of storing information on a per-flow basis. Second, part of the detection could be offloaded: the local middlebox could send "local" information to as-a-Service platform, which could perform computing-intensive detection strategies. Evaluating the use of softwarized and cloud-native blueprints to prevent malware hiding data within future IoT deployments is part of our ongoing research.

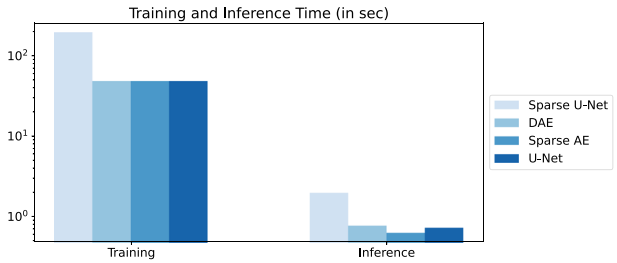


Fig. 8 Training and inference execution times in seconds (see the logarithmic scale)

For the sake of completeness, we also compared the execution times of both the training and the inferences phases of the models. Figure 8 depicts the results. As shown, our model performs slightly worse compared to the other solutions. This is mainly because it relies on data windows, worsening the overall performances. However, since our approach is able to process a single window in 16 seconds, it is possible to deploy the model in a real-time fashion, as well as on resource-constrained devices, especially in terms of storage capabilities.

6 Conclusions and future work

In this work, we showcased the use of an ensemble of autoencoders for detecting network covert channels targeting IoT scenarios. Specifically, we leveraged an incremental learning scheme for extracting a number of base detectors to build a robust ensemble model and then mitigate typical "catastrophic forgetting" risks. Our approach has been designed to be lightweight and requires a limited number of training examples at time to be effective. Experiments demonstrated that *i*) with a reduced ensemble size, the model is able to improve its detecting performances as the recent data could be more informative w.r.t. the past ones, *ii*) a deep ensemble-based model can improve the performances of the base model, especially it can achieve a probability of detection (i.e., Recall) of $\sim$ 91% while exhibiting a good precision $\sim$ 95%, *iii*) the proposed model is robust to the training data scarcity, and finally *iv*) our approach achieves better performance when compared to different baseline architectures.

Future works aim at broadening our current experimentation. In fact, even if hidden communications observed in real attacks primarily rely upon simple mechanisms, we only investigated the use of the TTL field within the IPv4 header. Hence, part of our research activity is aimed at considering hidden communications targeting other protocols (e.g., IPv6) or implementing more advanced cloaking schemes, such as those based on HTTP cookies. Another relevant part of our ongoing research focuses on setting a middlebox to test our detection approach with "live" traffic. This will allow to understand additional processing or energy consumption that could impair the real-time behavior desirable for many IoT applications.

Further advancements concern refining the AI-based framework proposed in this work. In more detail, the fast pace at which threats are evolving requires constantly "tweak" the components used to spot malware hidden communication, i.e., the data and the model are tightly-coupled with the abused protocol or field. Thus, a first advancement deals with the design of suitable "intermediate" representations that can handle different threats in a coherent manner, e.g., via protocol-agnostic or data-hiding-independent indicators of the covert data exchange. We are also interested in exploring the feasibility of incorporating a lifelong change

point detection approach to enhance the resilience of our method against the forgetting problem.

Lastly, despite the fact that our approach does not process sensitive information, both the learning and detection phases require gathering network traffic. We also considered the AI/ML framework co-located with a gateway or a router placed at the border of the network. Therefore, a future advancement explores the feasibility of using a more distributed approach or a federated scheme. It would be possible to cope with privacy issues typical of the IoT world (e.g., preventing that traffic information has to be remotely sent for processing) as well as to guarantee scalability properties (e.g., traffic is collected by different resource-constrained entities).

Acknowledgements This work was partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU.

Author Contributions All authors equally contributed to the work.

Funding Open access funding provided by Università della Calabria within the CRUI-CARE Agreement.

Data Availability The original datasets used in this research are borrowed from a third-party source and modified for our purposes and can be obtained upon request.

Declarations

Conflicts of interest The authors declare that they have no conflict of interest.

References

Abderrahim, N.Y.Q., Abderrahim, S., Rida, A. (2020). Road segmentation using u-net architecture. In *2020 IEEE international conference of moroccan geomatics (Morgeo)* (pp. 1–4). https://doi.org/10.1109/Morgeo49228.2020.9121887

Ahmad, Z., Shahid Khan, A., Wai Shiang, C., et al. (2021). Network intrusion detection system: a systematic study of machine learning and deep learning approaches. *Trans on Emerging Telecommunications Technologies 32*(1):e4150:1–e4150:29. https://doi.org/10.1002/ett.4150

Alcaraz C, Bernieri G, Pascucci F, et al. (2019). Covert channels-based stealth attacks in industry 4.0. *IEEE Systems Journal 13*(4):3980–3988. https://doi.org/10.1109/JSYST.2019.2912308

Angiulli, F., Fassetti, F., Manco, G., et al. (2017). Outlying property detection with numerical attributes. *Data Mining and Knowledge Discovery, 31*(1), 134–163. https://doi.org/10.1007/s10618-016-0458-x

Antonakakis, M., April, T., Bailey, M., et al. (2017). Understanding the Mirai Botnet. In *26th USENIX security symposium* (pp. 1093–1110)

Balandina, E., Balandin, S., Koucheryavy, Y., et al. (2015). IoT Use Cases in healthcare and tourism. In *2015 IEEE 17th conference on business informatics* (pp. 37–44), https://doi.org/10.1109/CBI.2015.16

Bengio, Y., Pascal, L., Dan, P., et al. (2006). Greedy layer-wise training of deep networks. In Advances in Neur. *Inf. Proc. Sys., 19*, 153–160.

Blinowski, G.J., Piotrowski, P. (2020). CVE based classification of vulnerable IoT systems. In *Theory and applications of dependable computer systems: proceedings of the fifteenth international conference on dependability of computer systems* (pp. 82–93). https://doi.org/10.1007/978-3-030-48256-5_9

Cabuk, S., Brodley, C.E., Shields, C. (2004). IP covert timing channels: design and detection. In *Proceedings of the 11th ACM conference on computer and communications security* (pp. 178–187). https://doi.org/10.1145/1030083.1030108

Caputo, D., Verderame, L., Ranieri, A., et al. (2020). Fine-hearing Google Home: why silence will not protect your privacy. *Journal of Wir Mob Net Ubiq Comp, and Dep Appl 11*(1):35–53. https://doi.org/10.22667/JOWUA.2020.03.31.035

Carrara, B., Adams, C. (2015). On acoustic covert channels between air-gapped systems. In *Foundations and practice of security: 7th international symposium*. Springer, pp. 3–16, https://doi.org/10.1007/978-3-319-17040-4_1

Cassavia, N., Caviglione, L., Guarascio, M., et al. (2022). Ensembling sparse autoencoders for network covert channel detection in IoT ecosystems. In *Foundations of intelligent systems: 26th international symposium* (pp. 209–218). https://doi.org/10.1007/978-3-031-16564-1_20

Cassavia, N., Caviglione, L., Guarascio, M., et al. (2023). Federated Learning for the efficient detection of steganographic threats hidden in image icons. In: Pervasive knowledge and collective intelligence on web and social media. Springer Nature Switzerland, Cham, pp. 83–95. https://doi.org/10.1007/978-3-031-31469-8_6

Caviglione, L., & Mazurczyk, W. (2022). Never mind the malware, Here's the Stegomalware. *IEEE Security & Privacy, 20*(5), 101–106. https://doi.org/10.1109/MSEC.2022.3178205

Caviglione, L., Merlo, A., Migliardi, M. (2018). Covert channels in IoT deployments through data hiding techniques. In *2018 32nd Intl. Conf. on Adv. Inf. Net. and Appl. Workshops* (pp. 559–563). https://doi.org/10.1109/WAINA.2018.00144

Caviglione, L., Mazurczyk, W., Repetto, M., et al. (2021). Kernel-level tracing for detecting stegomalware and covert channels in linux environments. *Computer Networks* 191:108,010:1–108,010:12. https://doi.org/10.1016/j.comnet.2021.108010

Caviglione, L., Comito, C., Guarascio, M., et al. (2023). Emerging challenges and perspectives in deep learning model security: a brief survey. *Systems and Soft Computing* pp. 200,050:1–200,050:7. https://doi.org/10.1016/j.sasc.2023.200050

Chakkaravarthy, S. S., Sangeetha, D., & Vaidehi, V. (2019). A survey on malware analysis and mitigation techniques. *Computer Science Review, 32*, 1–23. https://doi.org/10.1016/j.cosrev.2019.01.002

Corizzo, R., Dauphin, Y., Bellinger, C., et al. (2021). Explainable image analysis for decision support in medical healthcare. In *2021 IEEE international conference on big data* (pp. 4667–4674), https://doi.org/10.1109/BigData52589.2021.9671335

Darwish, O., Al-Fuqaha, A., Brahim, G.B., et al. (2019). Using hierarchical statistical analysis and deep neural networks to detect covert timing channels. *Applied Soft Computing 82*:105,546:1–105,546:15. https://doi.org/10.1016/j.asoc.2019.105546

Elsadig, M.A., Gafar, A. (2022). Covert channel detection: machine learning approaches. *IEEE Access 10*:38,391–38,405. https://doi.org/10.1109/ACCESS.2022.3164392

Faber, K., Corizzo, R., Sniezynski, B., et al. (2023). Vlad: Task-agnostic vae-based lifelong anomaly detection. *Neural Networks, 165*, 248–273. https://doi.org/10.1016/j.neunet.2023.05.032

Folino, F., Folino, G., Guarascio, M., et al. (2021). On learning effective ensembles of deep neural networks for intrusion detection. *Information Fusion, 72*, 48–69. https://doi.org/10.1016/j.inffus.2021.02.007

Folino, G., Guarascio, M., & Papuzzo, G. (2019). Exploiting fractal dimension and a distributed evolutionary approach to classify data streams with concept drifts. *Applied Soft Computing, 75*, 284–297. https://doi.org/10.1016/j.asoc.2018.11.009

Frolova, D., Kogos, K., Epishkina, A. (2021). Traffic normalization for covert channel protecting. In *2021 IEEE conference of russian young researchers in electrical and electronic engineering* (pp. 2330–2333). https://doi.org/10.1109/ElConRus51938.2021.9396163

Guarascio, M., Zuppelli, M., Cassavia, N., et al. (2022). Detection of network covert channels in iot ecosystems using machine learning. In *ITASEC* (pp. 102–113)

Gupta, B. B., Tewari, A., Jain, A. K., et al. (2017). Fighting against phishing attacks: state of the art and future challenges. *Neural Computing and Applications, 28*, 3629–3654. https://doi.org/10.1007/s00521-016-2275-y

Hinton, G., & Salakhutdinov, R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504–507. https://doi.org/10.1126/science.1127647

Ho JW (2019). Covert channel establishment through the dynamic adaptation of the sequential probability ratio test to sensor data in IoT. *IEEE Access 7*:146,093–146,107. https://doi.org/10.1109/ACCESS.2019.2945974

Jung, W., Cui, K., Koltermann, K., et al. (2022). Light auditor: power measurement can tell private data leakage through iot covert channels. In *Proceedings of the 20th ACM conference on embedded networked sensor systems* (pp. 518–532). https://doi.org/10.1145/3560905.3568535

Li, Y., Chen, Z., Zha, D., et al. (2022). Automated anomaly detection via curiosity-guided search and self-imitation learning. *IEEE Trans on Neural Networks and Learning Systems, 33*(6), 2365–2377. https://doi.org/10.1109/TNNLS.2021.3105636

Mazurczyk, W., & Caviglione, L. (2015). Information hiding as a challenge for malware detection. *IEEE Security & Privacy, 13*(2), 89–93. https://doi.org/10.1109/MSP.2015.33

Mazurczyk, W., & Caviglione, L. (2021). Cyber reconnaissance techniques. *Communications of the ACM, 64*(3), 86–95. https://doi.org/10.1145/3418293

McLaren, P., Russell, G., Buchanan, B. (2017). Mining malware command and control traces. In *2017 Computing conference* (pp. 788–794). https://doi.org/10.1109/SAI.2017.8252185

Neshenko, N., Bou-Harb, E., Crichigno, J., et al. (2019). Demystifying IoT security: an exhaustive survey on iot vulnerabilities and a first empirical look on internet-scale iot exploitations. *IEEE Communications Surveys & Tutorials, 21*(3), 2702–2733. https://doi.org/10.1109/COMST.2019.2910750

Noor, M. M., & Hassan, W. H. (2019). Current research on internet of things (IoT) security: A survey. *Computer Networks, 148*, 283–294. https://doi.org/10.1016/j.comnet.2018.11.025

Nowakowski, P., Zórawski, P., Cabaj, K., et al. (2021) Detecting network covert channels using machine learning, data mining and hierarchical organisation of frequent sets. *Journal of Wir Mob Net Ubiq Comp, and Dep Appl 12*(1):20–43. https://doi.org/10.22667/JOWUA.2021.03.31.020

Parisi, G. I., Kemker, R., Part, J. L., et al. (2019). Continual lifelong learning with neural networks: A review. *Neural Networks, 113*, 54–71. https://doi.org/10.1016/j.neunet.2019.01.012

Repetto, M., Caviglione, L., Zuppelli, M. (2021) bccstego: A framework for investigating network covert channels. In *The 16th Intl Conf. on Avail., Rel. and Sec. (ARES)* (pp 1–7). https://doi.org/10.1145/3465481.3470028

Sahu, A. K., Sharma, S., Tanveer, M., et al. (2021). Internet of things attack detection using hybrid deep learning model. *Computer Communications, 176*, 146–154. https://doi.org/10.1016/j.comcom.2021.05.024

Sivanathan, A., Gharakheili, H. H., Loi, F., et al. (2018). Classifying IoT devices in smart environments using network traffic characteristics. *IEEE Trans on Mobile Computing, 18*(8), 1745–1759. https://doi.org/10.1109/TMC.2018.2866249

Skowron, M., Janicki, A., Mazurczyk, W. (2020) Traffic fingerprinting attacks on internet of things using machine learning. *IEEE Access 8*:20,386–20,400. https://doi.org/10.1109/ACCESS.2020.2969015

Thakkar, A., & Lohiya, R. (2021). A review on machine learning and deep learning perspectives of ids for iot: recent updates, security issues, and challenges. *Archives of Computational Methods in Engineering, 28*, 3211–3243. https://doi.org/10.1007/s11831-020-09496-0

Vaccari, I., Chiola, G., Aiello, M., et al. (2020). MQTTset, a new dataset for machine learning techniques on MQTT. *Sensors 20*(22):6578:1–6578:17. https://doi.org/10.3390/s20226578

Velinov, A., Mileva, A., Wendzel, S., et al. (2019). Covert channels in the MQTT-based internet of things. *IEEE Access 7*:161,899–161,915. https://doi.org/10.1109/ACCESS.2019.2951425

Vieira, M.A.M., Castanho, M.S., Pacífico, R.D.G., et al. (2021). Fast packet processing with eBPF and XDP: concepts, code, challenges, and applications. *ACM Computing Surveys 53*(1):16:1–16:36. https://doi.org/10.1145/3371038

Vormayr, G., Zseby, T., & Fabini, J. (2017). Botnet communication patterns. *IEEE Communications Surveys & Tutorials, 19*(4), 2768–2796. https://doi.org/10.1109/COMST.2017.2749442

Wendzel, S., Zander, S., Fechner, B., et al. (2015). Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys, 47*(3), 1–26.

Wendzel, S., Caviglione, L., Mazurczyk, W., et al. (2021) A revised taxonomy of steganography embedding patterns. In *Proceedings of the 16th Intl Conf. on Avail., Rel. and Sec. (ARES)* (pp 1–12). https://doi.org/10.1145/3465481.3470069

Zander, S., Armitage, G., Branch, P. (2006) Covert channels in the IP time to live field. In *Australian Tel. Net. and Appl. Conf.* (pp 1–5)

Zander, S., Armitage, G., & Branch, P. (2007). A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials, 9*(3), 44–57. https://doi.org/10.1109/COMST.2007.4317620

Zorawski, P., Caviglione, L., Mazurczyk, W. (2023). A long-term perspective of the internet susceptibility to covert channels. *IEEE Communications Magazine* pp 1–7. https://doi.org/10.1109/MCOM.011.2200744

Zuppelli, M., Caviglione, L. (2021) pcapStego: A tool for generating traffic traces for experimenting with network covert channels. In *The 16th Intl Conf. on Avail., Rel. and Sec. (ARES)* (pp 1–8). https://doi.org/10.1145/3465481.3470067