Detecting Network Covert Channels using Machine Learning, Data Mining and Hierarchical Organisation of Frequent Sets

Piotr Nowakowski, Piotr Żórawski, Krzysztof Cabaj*, and Wojciech Mazurczyk
Warsaw University of Technology, Warsaw, Poland
{P.Nowakowski, P.Zorawski, K.Cabaj, W.Mazurczyk}@ii.pw.edu.pl

Received: December 29. 2020; Accepted: February 18, 2021; Published: March 31, 2021

Abstract

Due to continuing improvements in defensive systems, malware developers create increasingly sophisticated techniques to remain undetected on the infected machine for as long as possible. One flavor of such methods are network covert channels, which, to transfer secret data, utilize subtle modifications to the legitimate network traffic. As currently there is no one-size-fits-all approach which would be effective in detecting covert communication in an efficient and scalable manner, more research effort is needed to devise a suitable solution. That is why, in this paper we propose to utilize machine learning and data mining accompanied by hierarchical organization of frequent sets to detect network covert channels: both distributed and undistributed. The obtained experimental results prove that the proposed approach is effective and efficient.

Keywords: Distributed Network Covert Channels (DNCCs), Network Security, Information Hiding, Data mining, Machine Learning.

1 Introduction

Cybercriminals are continuously devising novel methods that would enable them to delay or thwart the detection of their malicious activities on the infected device. The main reason for that is the constant improvement of the existing defensive systems. The most popular techniques used to cloak the attackers include [15]: multistage loading, fileless operation capabilities, encrypted and obfuscated payloads, anti-analysis mechanisms, and recently also various types of information hiding techniques [5].

During the last years, many types of data hiding methods have been devised, but lately an increased attention is brought towards techniques that, to transfer secrets, subtly modify network traffic. This subgroup of information hiding is called *network covert channels* and due to their nature they serve well purpose mostly due to their ephemeral nature. So far, in the literature a plethora of different techniques utilizing different network protocols have been proposed [18], [9], [14]. It must be also noted that, commonly, attackers use covert channels to cloak, e.g., communication with Command & Control (C&C) servers, exfiltration of stolen data, or downloading of additional malware modules or configurations [1].

Moreover, it is predicted that the trend of using data hiding techniques by attackers is likely to prevail in the near future [1] and soon we will experience a rise in the utilization of far more sophisticated network covert channels than are currently deployed [8], [5]. A recent branch of such complex data hiding methods that are increasingly gaining attention are called *distributed network covert channels* (DNCCs) [2]. They are defined as channels that during covert communication spread the secret data

*Corresponding author: Department of Electronics and Information Technology, Institute of Computer Science, Nowowiejska 15/19, 00-65 Warsaw, Poland, Tel: +48-22-234-77-11 among many flows/protocols/hosts or use multiple data hiding methods within the same flow or within various Protocol Data Units (PDUs) [6]. It is worth noting that, in contrast, the typical (undistributed) network covert channels are storage or timing channels that to embed secret data utilize PDUs of a single flow or protocol [8]. Utilization of the DNCC for the attacker can be profitable because it may improve the overall stealthiness and resulting data hiding capacity of the hidden communication. This is possible as in the DNCC smaller parts of secret data are transmitted using several covert channels, flows, or hosts.

It must be emphasized that, up till now, in the existing literature only limited attempts have been made to detect distributed network covert channels [2]. Moreover, the proposed detection solutions were mostly focused on DNCCs that use many data hiding techniques simultaneously.

This work is aimed to improve this situation and it is an extended version of the previous conference paper [10] where we proposed a novel approach for network covert channel detection which utilizes multistep traffic processing combined with data mining and statistical techniques.

However, it must be noted that in this paper we significantly extend the initial concept and its evaluation. In more detail, the novel contributions of this work are:

- We introduce a new steganographic communication error detection and correction scheme for the DNCC,
- We add two more detection strategies which are based on machine learning (ML) – one utilizing only raw data from our preprocessing software and the second where we apply ML to the raw data enhanced by the metadata produced by our original concept,
- We conduct an extensive experimental evaluation where we determine the characteristics of all detection variants.

The rest of the paper is organized as follows. Section 2 presents the most notable works related to the detection of network covert channels. In section 3 the network covert channels we have selected for the experimental evaluation are presented. Next, in section 4 the proposed network covert channel detection strategy is introduced. Then, the methodology as well as the experimental testbed used are presented in section 5, while the obtained results are enclosed in section 6. Finally, section 7 summarizes our work and outlines potential future directions.

2 Related work

Nowadays, most of the research concerning the detection of information hiding techniques is devoted to discovering data hiding in various types of image, audio, or video files. However, as network covert channels become increasingly popular among attackers, this trend is slowly changing, especially that such techniques are used not only by human beings but they start appearing even in malware [1], as a technique, e.g., for protecting an attacker's C&C channel.

The simplest method that can be utilized for the detection of a network covert channel is to use the network warden [7]. Network warden can be treated as an Intrusion Detection System (IDS) which contains rules allowing the detection of well-known covert channels. During warden activity, all network traffic is subjected to inspection to find clues for network covert channel utilization. For example, we can consider a situation when in the IP packet Don't Fragment (DF) flag is set to '1', however the Fragment Offset (FO) field is carrying data other than zeros. This can be treated as a clear indication of the covert communication as under normal circumstances the FO field should contain nonzero values only if More Fragments (MF) flag is set to '1'. In this case, the network warden can normalize such disobeying fields and thus remove any secret data that is stored there. Detection of such 'misuse' of protocols' header fields is effective for well-known attacks and in the situation when security researchers provide rules.

However, such an approach is unsuitable for the detection of new, completely unknown methods. For such threats, behavioral analysis with anomaly detection should be used. In the literature, we can find numerous approaches which rely on various techniques. A brief description of the most relevant ones is presented below.

In [19] the authors utilize Markov model for the detection of anomalous traffic. Initially, they are using network traffic without covert channels to generate a model for normal TCP/IP stack activity, which they call TMM – TCP Markov Model. Furthermore, during the detection phase, unknown traffic, potentially harmful containing hidden data is compared to the previously 'clean' model using Kullback-Leibler (KL) statistical test. If the calculated value is greater than the chosen threshold, the traffic is marked as containing covert data.

Similar approach is described in [13], however, in this case the authors propose to use neural networks to learn the initial sequence number (ISN) generator. Such a model can be later used for the detection of covert channels that encode data in ISN number – for example, NUSHU tool. The proposed detection system observes the network traffic, predicts the next ISN number using the learned model, and compares it with the observed ISN carried in TCP segments. Enhanced neural networks, more specifically Recurrent Neural Network (RNN), are used in the [12]. What is interesting in this research, network covert channels are detected by performing an analysis of system calls taken from the monitored machine.

Other researchers propose to detect covert communication using machine learning-based approaches. In [11] the authors utilize Support Vector Machine (SVM) classifier for the detection of timing covert channels. As input data, various properties of inter-packet delay for the consequent 100 and 2000 packets are used. In [16] authors introduce two-step solution which uses density-based clustering. In the first step, the network data is clustered. Then, in the second step, the introduced classification algorithm can analyze the clusters and check if any of them is characterized with different density. In this case, it can be treated as outliers, i.e., as a sign of covert transmission.

When compared with existing works, the proposed detection method introduced in this paper uses machine learning together with data mining frequent pattern discovery. Such an approach has not been evaluated yet in the literature for network covert channel detection. The only work that deals with this topic is our previous papers published in [2] and [10]. However, in [2] only one *minimal support* parameter was used, while the method proposed in [10] utilizes multiple discovery executions and automatic analysis of the gathered results and is applicable to the DNCCs that rely on many data hiding techniques which are scattering covert data into many flows and hosts.

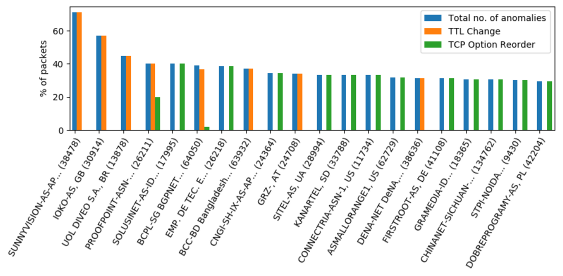
As already mentioned, this paper is a significantly extended version of the conference paper [10] where we improve several deficits of the initial approach as well as we apply machine learning techniques to further improve its detection performance and perform extensive experimental evaluation.

3 Network covert channels selected for the DNCC creation

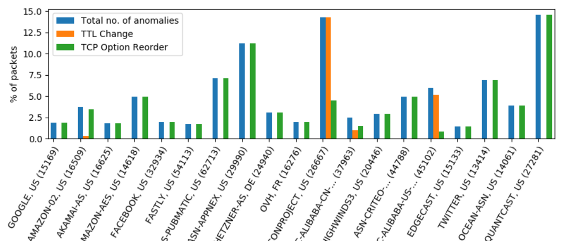
In this section, we first discuss our motivation behind using the three network covert channels that we chose for the experimental evaluation. Then, we present their detailed inner workings. Note that we deliberately selected network covert channels which rely on protocols that reside in three different layers of the TCP/IP stack, i.e., network (IPv4), transport (TCP), and application (HTTP).

3.1 Rationale behind covert channel selection

From the network traffic perspective, it is commonly believed that the utilization of covert channels would cause anomalies that should be easy to spot and that modern networks are no longer affected by



(a) Sorted by AS with the most frequently occurring anomalies.



(b) Sorted by most popular AS.

Figure 1: Network anomalies by AS, detected within web traffic to Alexa top 10,000 most popular websites (for the sake of readability, only top 20 positions are shown) [10].

routing or network protocol anomalies. A classical network anomaly example is a sudden change in the Time To Live (TTL) value (in the IPv4 header) within the same network connection. TTL-based mechanism is designed to prevent routing loops by limiting the number of network hops that a single packet can traverse. Each router on the communication path decreases the packet's TTL value and drops it if it reaches zero. However, if the TTL value has suddenly changed, it may also mean that the routing path has been changed, e.g., due to router failure and has been switched to the alternative routing path. In other cases, the TTL field can be used by simple covert channels [17], thus its sudden change may also indicate that the covert communication is taking place. If such an assumption is genuine, then each detected traffic anomaly should be treated as a clear indicator of the intentional packet manipulation performed either by the two hosts involved in network transmission or a malicious third party. In effect, a naive approach for covert channel detection would simply rely on enumeration of all identified traffic anomalies.

In this paper, we choose three covert channels that introduce anomalies to the normal overt network traffic. To verify whether these anomalies are occurring naturally in benign traffic, we have performed a very basic experiment. Specifically, we generated benign web traffic to the world's most popular websites and observed if any unexpected anomalies exist. The experiments were performed by launching an automated headless Chromium browser[1] to visit top 10,000 most popular websites listed by Alexa[2]. By analyzing the captured web traffic and grouping the responses' IPv4 source addresses by corresponding Autonomous Systems (AS), we were able to identify communication networks that exhibited a significant number of network anomalies. In the result, it was possible to establish a baseline using the most popular and commonly trusted networks.

The results of the conducted traffic measurements are presented in Figure 1. It is visible that for certain ASs up to 71% of the packets may contain an unexpected anomaly and even the most popular and benign networks exhibit up to 5% anomaly rates. For the readability purpose, the figures contain only 20 AS entries from all of the 1536 ASs observed during the experiment. Note that the complete results for the conducted measurements are presented in Table 1. Based on the obtained results, it can be observed that the dominant anomaly is TCP Options Reorder, which is prevalent in nearly all observed traffic. TTL changes are observed only with up to 15% of the traffic coming from examined ASs. Note that we were not able to verify whether TTL changes are caused by routing instability, load balancing mechanisms, or network traffic tampering. Unfortunately, based on the performed experiments, we were unable to analyze anomalies in the Application Layer. This is because the vast majority of the modern web servers

[1]https://github.com/pyppeteer/pyppeteer
[2]https://www.alexa.com/topsites

communicate using encrypted HTTPS traffic with the help of Transport Layer Security (TLS) protocol.

Based on the traffic analysis presented above, it is evident that the naive detection methods will be not effective as it will be difficult to differentiate between legitimate and covert traffic. Thus, a more sophisticated approach is needed. Note that we used the knowledge of the discovered anomalies to construct the network covert channels that we used during the performed experimental evaluation. We described the chosen data hiding techniques in the following subsection.

Observed anomalies [% of all pkts]	TTL changes	TCP Options reorder
Average value	0.789	4.685
Standard Deviation	4.131	6.029
Maximum value	71.154	40.000
Percentile		
5th	0.000	0.037
25th	0.000	0.618
50th	0.000	2.146
75th	0.000	6.698
80th	0.000	8.491
85th	0.073	10.496
90th	0.555	13.158
95th	3.522	16.667

Table 1: Distribution of the observed anomalies in network traffic from all observed ASNs (Alexa's top 10,000 websites) [10].

3.2 Network Covert Channels Functioning

During the research presented in this paper, we have developed and utilized three different steganographic methods functioning across three layers of the TCP/IP stack:

- TTL field modulation within IPv4 packet header (Network Layer),
- TCP Options reorder within TCP segment header (Transport Layer),
- HTTP Headers reorder within HTTP 1.1 requests (Application Layer).

The first method conceals secrets within the TTL field of the IPv4 header by encoding data symbols into deviations from the normal value of TTL field. This technique works as follows. When a new suitable network connection is established, the first few packets within the connection are left unmodified to establish a stable baseline value of TTL at the receiver's side. Then, the secret data is transmitted by incrementing or decrementing (i.e., modulating) the field's value using a predefined offset (i.e., symbol) which corresponds to a given data bit sequence. If there is no change to the TTL value, then it means that no secret data was transmitted. For example, the TTL value of IPv4 packets originating from a Linux host is set by default to 64. By passing through network routers, the TTL value is decreased by 1 for each network hop. At the covert receiver, the TTL value is observed with a smaller value (for example, 62) that should remain constant throughout the connection. If the TTL value changes unexpectedly from 62 to 60, then the symbol '-2' is transmitted, which can be mapped to a predefined data bit sequence.

Covert channel that relies on TCP Options reorder has been implemented by tampering with the order of 'Options' field within TCP segment's header. This field is an array which can carry variable values of optional parameters. Such an order is not strictly defined within the TCP standard aside from the terminating symbol 0, i.e., `END OF OPTIONS` and the array's size must be divisible by 32 bits. In our research, we have noticed that TCP segments usually contain a symbol 8 option, i.e., `TIMESTAMP` and multiple instances of symbol 1, i.e., `NO OPERATION` used to align the array length to 32-bit boundaries.

Because the order of options is ignored by the TCP/IP stack, it is possible to manipulate it to conceal secret information. In our implementation, we are swapping the position of the TIMESTAMP option with the first occurrence of the NO OPERATION to transmit a symbol or leave the order intact to send no data. This method is capable of sending only 1 symbol per TCP segment, so to transfer arbitrary binary data a specific signaling protocol is required. To transmit data, a two-step process is utilized. If there is no change to the options order, there is no hidden transmission. When the two options are swapped, we are signaling a control bit indicating that the next TCP segment will contain hidden data. If the following segment has also swapped the order of options, it means that bit '1' was transmitted, otherwise if no change is observed '0'.

Finally, HTTP header reorder covert channel utilizes HTTP protocol to transmit secret data by manipulating the order of headers contained within the request. The order of headers within the HTTP request is not strictly defined, thus it can be used to transfer covert data. The implemented technique works as follows. We assume that the repeated HTTP requests (sent by the same browser and with the same domain and URL) preserve the order of HTTP headers. If this is the case, then the first request is used by the covert communication parties to store the initial header order. Then, the difference in the header order in subsequent requests can be used to encode the covert data. Our algorithm uses index-based encoding. To transfer data, one of the headers is shifted to the beginning of the request. The index of this header in the original request is the symbol transferred (which corresponds to a group of bits). For example, we can send 2 bits of secret data per request by using four indexes: 1, 2, 3, 4 which map to 00, 01, 10, 11 secret data bits. If we consider, for example, the original HTTP request as:

```
    POST / HTTP/1.1
(0) Host: srv1-on.sdn
(1) User-Agent: python-requests/2.22.0
(2) Accept-Encoding: gzip, deflate
(3) Accept: */*
(4) Connection: keep-alive
(5) Content-Type: application/xml
(6) Content-Length: 406
```

and if we want to encode the symbol '2', the header with index 2 (Accept-Encoding) should be relocated to the beginning, resulting in a request:

```
    POST / HTTP/1.1
(0) Accept-Encoding: gzip, deflate
(1) Host: srv1-on.sdn
(2) User-Agent: python-requests/2.22.0
(3) Accept: */*
(4) Connection: keep-alive
(5) Content-Type: application/xml
(6) Content-Length: 406
```

3.3 Introduced error detection and correction scheme

In our previous work [10] the utilized covert transmission method used serial access to the bitstream of the linked input/output data file and the secret data bits were sent sequentially in the same order as they appeared in the data file. Such an approach was successful in cases where the covert transmission was not interrupted or altered. However, if a single secret bit was lost or erroneously inserted, then all secret data received later became shifted and unusable. This means that when the secret was transferred in the form of the ASCII content and only one bit was transmitted in the wrong order, then, in result, the whole extracted text at the receiving side was unreadable.

To summarize, the originally used serial method of transmitting secret data used in [10] has the following disadvantages:

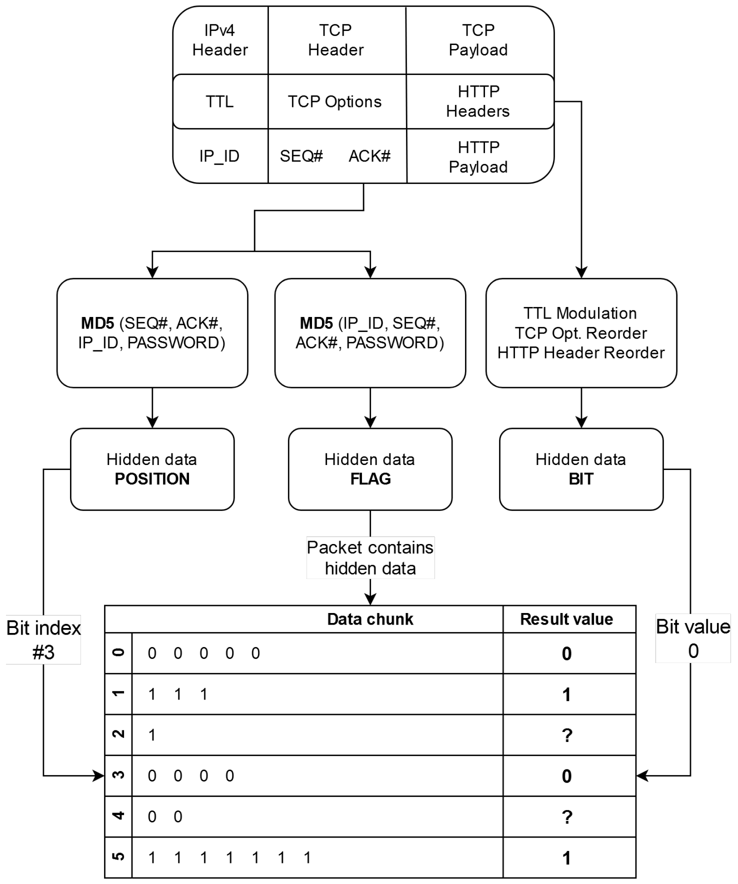


Figure 2: Simplified diagram of chunked data filtering, positioning, and data carriers. Selected IPv4 and TCP header fields are hashed with the MD5 algorithm to determine whether a packet should contain hidden data and the bit index. Modifications of the TTL value, TCP Options or HTTP Headers carry data bit.

- There was no indication of which packet contains steganographic data (it was achieved internally within the sender software), so the covert receiver had to closely monitor the whole incoming traffic for specific changes in packet fields,
- Due to the lack of synchronization and error correction mechanisms the resulting DNCC was highly sensitive to packet losses, reordering, or unexpected changes in the packet content which caused secret data corruptions. However, if a single secret bit was lost or erroneously inserted, the received secret data became altered,
- It required monitoring and modification of the order of options in all TCP packets to avoid data corruption in the TCP Options reorder module.

Considering the above, in this paper, we develop an upgraded transmission method which addresses all above-mentioned issues by introducing two improvements: *(i)* packet hashing to filter out packets that do not carry steganographic data and *(ii)* index data bit position within the data chunk regardless of the packet order sequence. Figure 2 visually explains the logic behind these improvements – both introduced mechanisms are described in detail below.

3.3.1 Mechanism for selecting packets to carry secret data

As already mentioned, in this mechanism we decided to use a hash-based solution. A group of fields (such as `IP ID`, `TCP ACK`, `TCP SYN`) is unlikely to be modified during the transmission. These values along with the chosen *salt* value are combined into a text string and then hashed using MD5 algorithm. The resulting hash can be then treated as a random value which allows to verify at the receiving side whether the packet carries the secret data bits or not. To have more control over the frequency of the secret data embedding, we evaluate if N lower bits of the hash, interpreted as a single integer, are less or equal to M. The N and M are selected to match as closely as possible the desired data frequency. For example, if $N = 3$, the possible values for the hash fragment are 0, 1, 2, 3, 4, 5, 6, 7. If $M = 5$, out of 8 possible values 6 will trigger packet modification, so the probability is $6/8 = 75\%$. In the experimental runs, N and M are set to match the desired packet modification probability (which is one of the considered parameters) as closely as possible. To determine if a packet contains secret data, the receiver has to follow the same procedure of generating the hash values as the covert sender.

3.3.2 Position-independent data transmission protocol

Instead of the raw serial transmission as used in [10], in this paper, the secret data is split into K-bit chunks which are transmitted sequentially (K is one of the tunable transmission parameters). The covert sender and the covert receiver spend a predefined amount of time on each chunk (a 'chunk time' interval) and then move on to the next one. The chunks themselves are transmitted using a technique similar to the hashmap implementation mentioned in the previous subsection. Just like for the packet selection mechanism, a group of fields from the packet are used to generate a hash value. The hash is used to select which bit of the chunk will be transmitted. The covert sender transmits bits until each bit is transmitted at least X times (1 is the minimum and a higher value can be set for increased reliability but at a cost of the lower 'usable' covert transmission speed). It should be also noted that, due to the randomness, some bits (i.e., positions in the current chunk) will be transmitted more often than the requested minimum. Thus, it is important to adjust the 'chunk time' interval (reflecting the throughput capability of the modules' configuration) to guarantee the transmission of the minimal acceptable number of bit repetitions for a chunk.

For modules which can transmit more than one secret bit per modified packet, the hash is used to place the first bit in the chunk normally, then it is used (again) as a seed for a pseudo-random number generator (a simple Linear Congruential Generator) to determine the positions for the rest of the bits. Similarly to the packet selection method described above, the covert receiver can calculate the bit positions by following the same algorithm as the covert sender. The value of each bit position is determined by the majority of the transmitted contents at this position (thus a simple error correction code is used). Note that the proposed algorithm is completely immune to packet reordering and offers adjustable protection against packet loss via duplication of the transmitted bits.

4 Frequent sets tree-based detection approach

In our previous work [10] we introduced a steganography detection method which is based on a specially constructed tree which stores the detected frequent sets for various values of *minimal support*. In the following sections, we briefly review the most important steps concerning raw data preprocessing, tree construction, and outlier finding. During the detailed presentation and discussion of the obtained results in section 6, we refer to this type detection strategy as 'Tree-based'.

4.1 Extraction of packet anomalies from traffic traces

The first step of the traffic analysis is performed directly on packet capture dumps, where each transmitted packet is simultaneously tracked as Layer 3, Layer 4, and Layer 7 connections. By grouping packets into individual multilayer connections, we can compare all subsequent pairs of neighboring packets within the connection. The values of all header fields are compared between packets and any differences are listed in a JSON log file. If the protocol specification allows a given field to have a variable list of values, then we compare the order of specified values to determine whether any value pairs were reordered. The resulting JSON file contains all discovered differences between packets within network connections. However, it must be noted that such an anomaly detection technique is viable only for the header fields that are relatively constant within network connections, e.g., TTL value in the IPv4 header. Values that are completely changed with each passing packet (for instance, TCP Sequence number) are out of the scope of this work. The resulting JSON file contains only these packets which exhibit unexpected changes in the header fields. Figure 3 graphically illustrates the preprocessing steps described in this section.

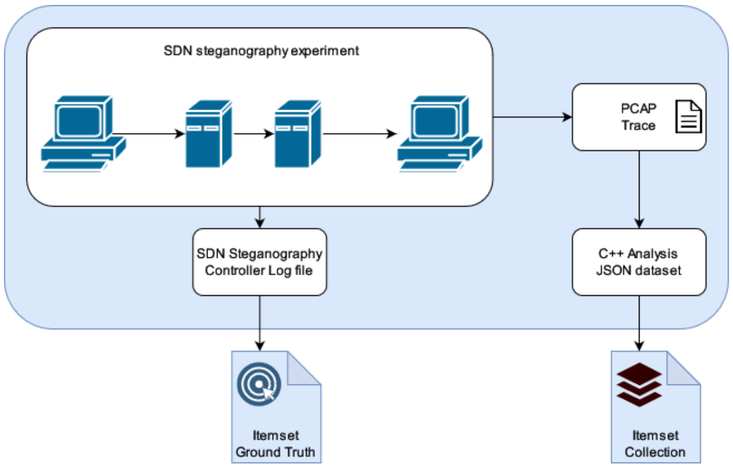

SDN steganography experiment
PCAP Trace
SDN Steganography Controller Log file
C++ Analysis JSON dataset
Itemset Ground Truth
Itemset Collection

Figure 3: Shared pipeline scheme – from the logs of SDN controllers and network traces gathered during an experiment the item set collection and the associated ground truth (steganograhic/not steganographic traffic) are generated.

4.2 Frequent set-based detection of traffic anomalies

The obtained file contains only information concerning changes between consecutive packet pairs. Even for benign traffic, this file may contain a significant amount of data. On top of that, the obtained information cannot be processed directly by statistical techniques because the detected anomalies are context-specific and thus applying such means will remove any context from the results.

In our previous research, we utilized the frequent set mining technique to identify network connections with recurring anomalies (to which we refer as ’FIM’ later in this paper). The discovery process takes as an input a JSON file obtained during the previously described step and uses the implementation from the PyFIM library[3]. Each detected header change is supplied with the layer-specific connection metadata as a single transaction in the dataset and the resulting frequent sets were analyzed. In our previous approach [2], we tried to apply the FIM algorithm directly as a detection method. The rationale

[3]http://www.borgelt.net/pyfim.html

behind such an approach was that with the proper parameters' tuning, the algorithm should return no results if there was no covert communication in the dataset, otherwise it should return one or more frequent item sets corresponding to the type of covert channels present in the dataset.

However, in this paper, we propose a different approach, where the FIM algorithm is executed multiple times with various *minimal support* parameters which control the minimum acceptable support of the discovered item sets. This changes the output significantly – with the higher *minimal support* discovered item sets are more general and correspond to a larger portion of the initial data (per item set). This 'cross-section' view is beneficial as some anomaly traits may not be evident at the lower level while obvious when using higher-level representation.

The obtained frequent sets from multiple runs of the PyFIM library are grouped by the *minimal support* parameter and stored in a JSON file. Exemplary detected frequent item sets are presented in Listing 1. Each detected item set contains the field names and values which provide metadata describing the network connection and the detected anomaly. As an example, set *A* describes the detected anomaly where the TCP Options order was changed unexpectedly within TCP connection from `192.168.1.30:36824` to `192.168.2.3:80`. In addition, the *support* field denotes the detected item set support.

```
A = {"IP_Addr_Dst": "192.168.2.3",
     "IP_Addr_Src": "192.168.1.30",
     "TCP_Port_Dst": 80, "TCP_Port_Src": 36824,
     "IHP:TCP_OPTIONS_REORDER": "TCP_OPTIONS_REORDER",
     "IP_Protocol": 6,
     "support": 1},
B = {"IP_Addr_Dst": "192.168.2.5",
     "IP_Addr_Src": "192.168.1.50",
     "TCP_Port_Dst": 80, "TCP_Port_Src": 48536,
     "IHP:TCP_OPTIONS_REORDER": "TCP_OPTIONS_REORDER",
     "IP_Protocol": 6,
     "support": 1},
C = {"IHP:HTTP_HEADER_REORDER": "HTTP_HEADER_REORDER",
     "IP_Addr_Dst": "192.168.2.3",
     "IP_Addr_Src": "192.168.1.30",
     "TCP_Port_Dst": 80, "TCP_Port_Src": 53796,
     "IP_Protocol": 6,
     "support": 1},
D = {"IP_Addr_Dst": "192.168.2.3",
     "IP_Addr_Src": "192.168.1.30",
     "TCP_Port_Dst": 80,
     "IHP:TCP_OPTIONS_REORDER": "TCP_OPTIONS_REORDER",
     "IP_Protocol": 6,
     "support": 305},
E = {"IP_Addr_Dst": "192.168.2.3",
     "IP_Addr_Src": "192.168.1.30",
     "TCP_Port_Dst": 80,
     "IP_Protocol": 6,
     "support": 1050},
```

Listing 1: Exemplary detected frequent item sets [10].

4.3 Grouping the detected frequent sets into a hierarchical tree

Because with the increasing *minimal support* parameter sets discovered by the PyFIM library tend to be less specific, it is reasonable to organize them into hierarchical tree with the least specific set as the root node of the tree and the most specific nodes as leaves. By organizing sets into a hierarchical tree, an additional relationship context is introduced into the dataset, which helps with data analysis. Figure 4 showcases all steps performed during frequent item sets discovery and tree construction.

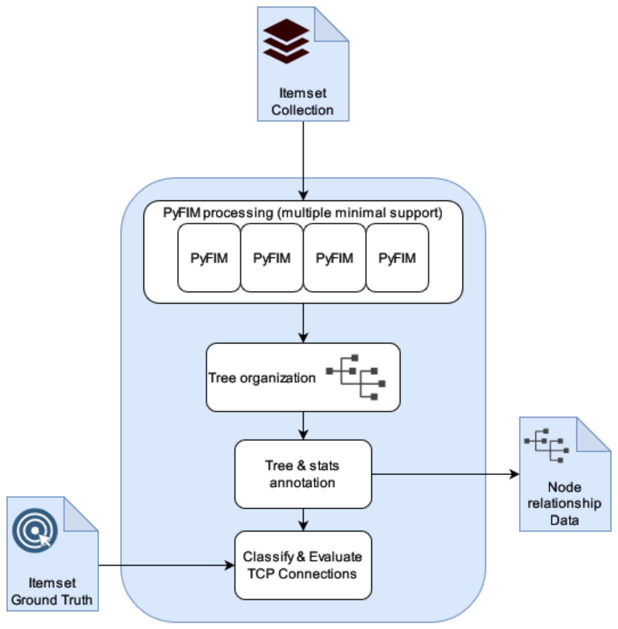


Figure 4: Tree-based approach pipeline: the individual item sets are grouped by the Frequent Itemsets algorithm and this grouping is performed in multiple attempts with different minimal support parameter.

To build a hierarchical tree, it is required to introduce relationship operators to compare frequent sets. The tree nodes are organized into parent-child node relationships when the following criteria are met:

- The parent node has a greater *support* value,
- The parent node key set is a subset of the child key set,
- The overlapping key values are equal in both parent and child (excluding *support* value).

The hierarchical tree building algorithm is executed using top-down approach from the highest *minimal support* and all discovered frequent item sets are attempted to fit into the tree. If a frequent set does not match the criteria for any node in the tree, it is withheld for later iterations. If all possibilities of fitting withheld nodes are exceeded, the node is attached to a new tree root.

By following the criteria and the algorithm provided above, the exemplary sets A, B, C, D, and E presented in the Listing 1 can be organized into the tree structure presented in Figure 5.

$$\begin{tikzcd}
 & E \arrow[dl, dash] \arrow[dr, dash] & & B \\
D \arrow[d, dash] & & C & \\
A & & &
\end{tikzcd}$$

Figure 5: Tree structure constructed from the exemplary frequent sets [10].

The most important benefit of organizing the frequent item sets into a hierarchical structure is the possibility to quickly summarize the type of traffic by briefly examining root nodes, and when further details are required, the tree can be followed to more specific nodes.

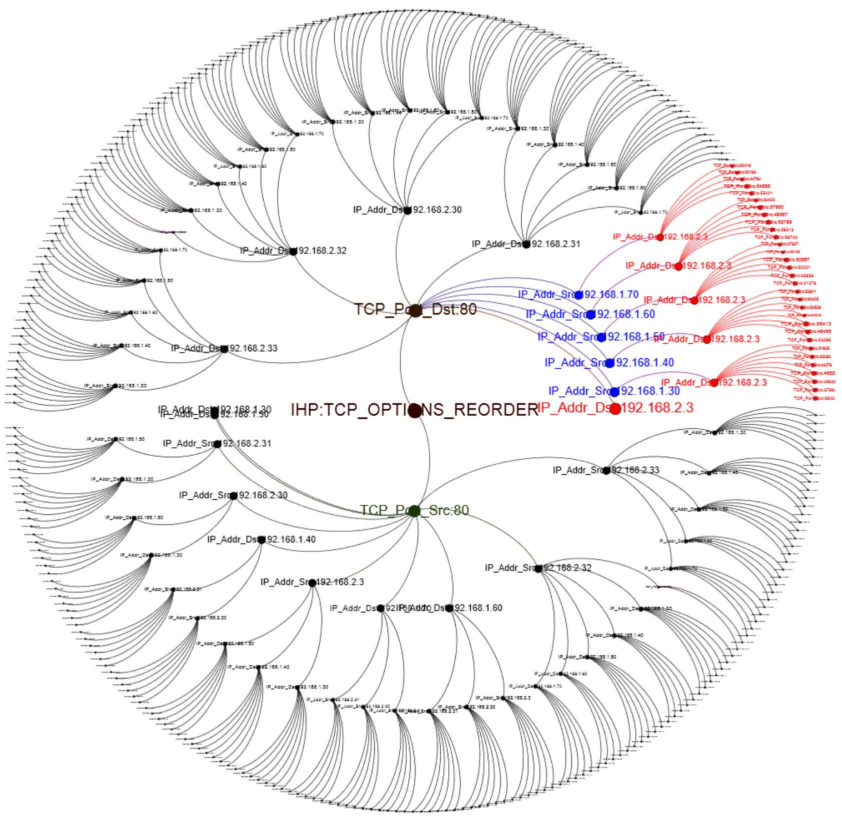


Figure 6: Exemplary hierarchical tree with classified nodes for the scenario with the DNCC using TCP Options Reorder covert channel between 5 clients and 1 server (red color denotes nodes classified as *outliers*, blue color denotes *compound outlier* nodes) [10].

4.4 Finding outliers within the tree structure

Independently of the tree building, the frequent item sets can be sorted into collections based on the types of features inside them. For example, all frequent sets which contain items associated with the source IP address, destination IP address, source port, and destination port can be treated as one collection, regardless of the actual values of these fields. As an example, both sets A and B from the previous subsection belong to the same collection because they feature the same keys, despite their values being different.

To find the outliers, firstly the median of frequent sets supports for each collection is calculated. Then, a frequent set can be classified as an outlier based on the deviation of the discovered frequent set support from the collection's median (with an adjustable threshold). More precisely, a frequent set is considered normal if it satisfies the following equation:

$$1.0 - T < \frac{S}{M} \leq 1.0 + T$$

where S is the frequent set support, M is the median support for the collection, and T is the adjustable threshold. Otherwise, it is considered as an outlier.

The introduced method generally marks frequent sets of high specificity. This is understandable, as naturally the number of frequent sets decreases if the specificity decreases – there is just not enough data to generate statistics for their proper classification. Thus, a second step of the algorithm is needed to mark less specific frequent sets and it works as follows. After the initial outlier marking, each node of the tree is recursively checked if it can become a *compound outlier*. A node is considered as *compound outlier* if more than $X\%$ (a configurable parameter) of its children are outliers, either normal or compound. This action marks additional nodes closer to the root of the tree as outliers which were missed during the previous step.

Finally, the steganographic classification can be performed. For each leaf (which represents the most specific frequent sets describing TCP connections), we recursively go up through the tree, looking for a parent node which was marked as either a compound or a normal outlier. If no such node was found when the root was reached, the leaf (the TCP connection) is considered as negative, i.e., no covert communication was found. Otherwise, as soon as such a node is discovered, the leaf is marked as positive, i.e., steganographic modifications were present and the classification process for this leaf ends.

The result of such a detection algorithm is a hierarchical tree with annotated outlier nodes. Each node within the tree contains an instance of frequent set alongside the support and metadata regarding the set collection it belongs to. The tree can be traversed top-down to list nodes classified as steganographic — each frequent set node describes a portion of the network traffic.

An exemplary tree has been depicted in Figure 6. In this scenario, the DNCC is using TCP Options Reorder data hiding method between 5 clients and 1 server. Red color denotes nodes classified as *outliers*, blue color denotes *compound outliers*. By following the tree structure from the root node, it is evident that the traffic has been grouped with the leading TCP Options Reorder the root node, which at the next level was evenly divided by TCP source and destination ports – HTTP requests (Destination port = 80) and responses (Source port = 80). At the lower level, the destination port was divided into source and destination addresses. Here nodes related to IPv4 source addresses 192.168.1.30, 192.168.1.40, 192.168.1.50, 192.168.1.60, 192.168.1.70 were classified as *compound outliers* and further down, destination address 192.168.2.3 and individual TCP connections were marked as *outliers*. This shows that the proposed classification approach was able to perfectly identify the DNCC scheme utilized in this scenario.

5 Experimental methodology and testbed

During the conducted experiments, we want to evaluate and compare the performances of various steganography detection approaches proposed in this paper. In more detail, we present our results, findings, and conclusions concerning three detection strategies:

- **Tree-based**: this is our previous strategy originally proposed in [10] which uses only the tree of item sets (Figure 4),
- **Basic ML**: it is a Machine Learning approach which relies only on the preprocessed data of the network traffic (Figure 7),
- **Hybrid**: this is also a Machine Learning approach, however, in this case it operates on the preprocessed data and additionally on the information obtained from the tree of item sets (Figure 7) – so it combines both detection methods mentioned previously above.

Note that the two detection methods which utilize ML techniques (i.e., 'ML Basic' and 'Hybrid') use the trained Machine Learning model obtained via the Driverless AI (DAI)[4]. DAI is an automatic machine learning platform which can handle automatic feature engineering, model selection, tuning, validation, and deployment. We have decided to use it during the experimental evaluation as it has been previously successfully deployed in a similar cybersecurity scenario [4].

[4]https://www.h2o.ai/products/h2o-driverless-ai

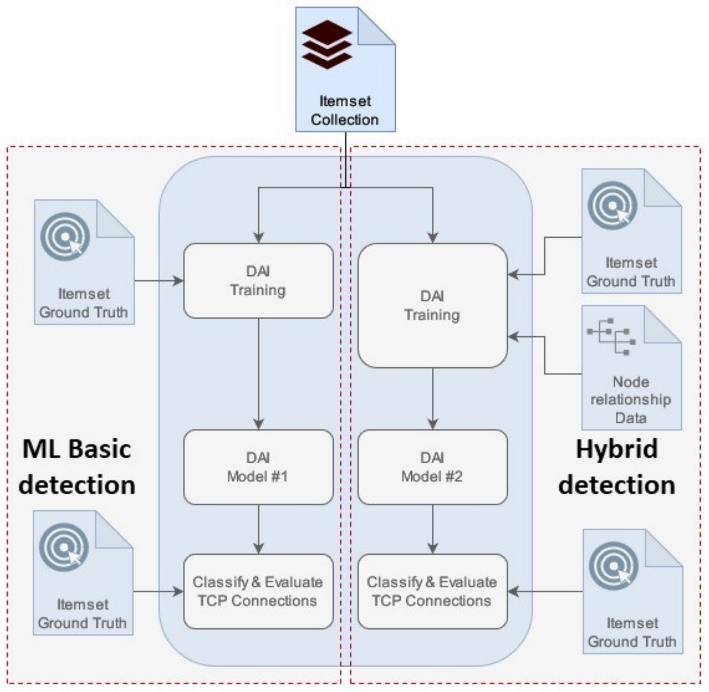


Figure 7: CSV pipeline scheme – two AI models are trained on the data: the first one only with the information from the pcap item set ('ML Basic' approach), the second one with the information from the pcap item set and from the tree node relationship ('Hybrid' method).

All three above-mentioned detection strategies are evaluated using raw network traffic as an input, which simulates various types of DNCC channels in the IoT environment consisting of temperature sensors. During the conducted experiments, we compare decisions of the detection algorithms (i.e., whether the traffic is considered as containing or not steganographic data embedded) – with the ground truth taken from the configuration of the recorded network traffic. The ground truth contains the information which connections, in this particular test run had the steganographic transmission enabled. In effect, for each of the three strategies mentioned above, we are able to calculate true positives and false positives. Using these results, we can prepare ROC (Receiver Operating Characteristic) curves and calculate AUC (Area Under Curve) parameter used in the final comparison of detection methods.

Experimental traffic was generated during our research in the testbed, which is described in detail in subsection 5.1. During the conducted experiments, we introduce various network anomalies, which are described in section 5.2. This part of the experiment evaluates how well our coding scheme (see section 3.3.2) performs as well as what impact on the detection rate such network conditions have. Moreover, during the conducted experiments we examine various configurations of the DNCC, which in effect produce a test set containing more than 7600 test runs. All details concerning the generated test set which is used later in the evaluation process are enclosed in subsection 5.3.

5.1 Test-bed

Our experiment scenario consists of two separated LANs (Figure 8). The first one consists of machines that mimic IoT devices and the second one which contains a central HTTP server. This emulates the real-world setup, where such LANs would be connected via a WAN or Internet. Due to the fact that our research mainly focuses on DNCC and not on the detailed analysis of the particular data hiding method used for its creation, all experiments are performed in the virtual environment. Moreover, for the sake of simplicity of the utilized topology, the default routers of both LANs are connected directly by a virtual link. In such scenario, the clients in the first network have data that needs to be secretly transmitted to the second network using data hiding techniques. In this testbed, covert transmission is performed by a custom SDN application running on a SDN controller host. During our experiments, we change the number of clients which represent IoT devices, sending data to one or multiple central servers.

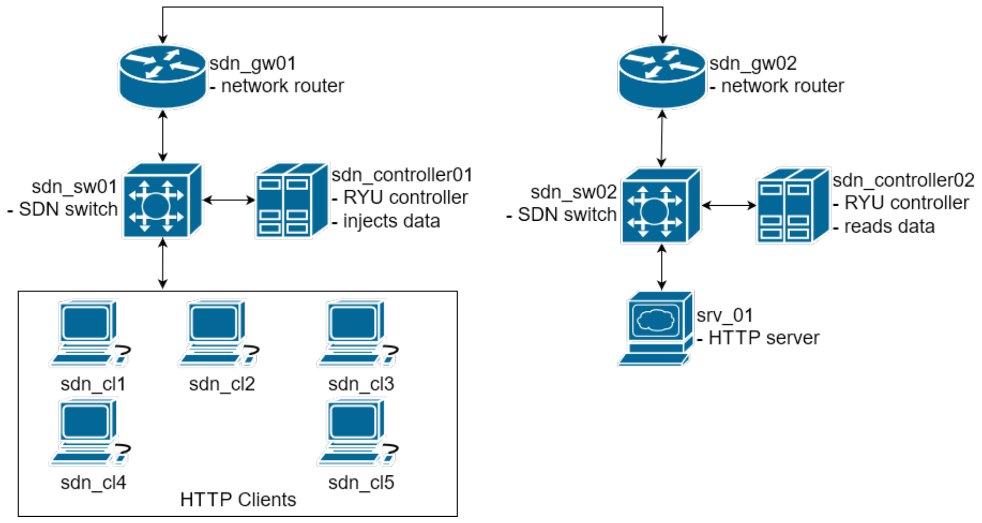


Figure 8: Utilized experimental test-bed [10].

Gateway machines are connected to the Internet, which allows communication between LANs. The IoT sensors periodically transmit measurements to the server machine, which create a flow of packets that passes through both SDN switches, where the covert messages are injected (at the first SDN switch) and extracted (at the second SDN switch).

In our testbed we used the following software:

- All Virtual Machines are running Centos 7.5.1804 (Core),
- Open vSwitch v 2.10.1 on the SDN Switches,
- RYU Python SDN controller,
- Apache HTTPD v 2.4.6 on the HTTP Server machine.

Devices `sdn_cl1`, `sdn_cl2`, `sdn_cl3`, `sdn_cl4`, `sdn_cl5` act as the smart temperature sensor nodes. A Python script is used to push sensor readings via HTTP `POST` messages to the central HTTP server running on `sdn_srv01`. Machine `sdn_sw01` runs Open vSwitch network switch which is managed by the `sdn_controller01` machine running RYU Controller and the device `sdn_gw01` acts as a router for this network. A similar configuration is used on the receiving side of the test-bed, with `sdn_sw02` running another Open vSwitch controlled by RYU running on `sdn_controller02` machine. Machine `sdn_gw02` acts as a router for the receiver network.

As the ML-based platform, as already mentioned, we chose *Driveless AI dai*-1.8.0 which is an automatic Machine Learning platform and *CUDA 10.2*, i.e., the parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs)[5]. One

[5]https://developer.nvidia.com/cuda-zone

of the most important features of DAI we utilize is the auto-tuning of the ML algorithms, which allows a focus on the main problem, i.e., the covert transmission detection and not on the tuning of ML parameters.

5.2 Introducing anomalies to the network traffic

In modern computer networks, network anomalies occur sporadically and are typically introduced by differences in network stack implementations, network routing issues, or undefined behavior of network protocols. In general, network anomalies are introduced along the routing path and longer paths are more likely to contain anomalies. As already mentioned, the testbed utilized in this research consists of two Local Area Networks directly connected to each other through a short routing path. This makes the traffic exchanged using the above-mentioned testbed very unlikely to contain any sporadic network anomalies (which, as stated above, occur typically in communication networks). On the other hand, the steganographic methods utilized in this testbed intentionally introduce network anomalies as a data carrier for covert channel transmission. In result, the vast majority of anomalies within the test-bed traffic are introduced by steganography, thus the classification process could be greatly simplified.

To make the classification process nontrivial, we have decided to introduce benign network anomalies into our testbed to replicate the anomaly distribution typical to what we discovered in WAN traffic. The introduction of benign anomalies makes the steganographic data transmission more prone to data corruption and in more aggressive cases, the covert transmission is impossible to continue due to the large amount of errors. Thus, to make the steganographic data transmission more resilient to data corruption, we developed a dedicated error detection and correction scheme which was described in detail in section 3.3. With the utilization of such mechanism, we are able to introduce an arbitrary number of benign network anomalies while maintaining high steganographic transmission accuracy. In effect, the covert channel detection algorithms can be validated in our testbed in an environment close to the real-world case.

To simulate various network anomalies, we utilize existing steganography modules to introduce random data corruption in the covert transmission layer, namely, random data bits:

- **insertion**: non-steganographic packets are randomly modified during transmission to introduce a random steganographic value that can be read by the receiver,
- **deletion**: steganographic packets are randomly deleted during transmission to simulate data loss,
- **flipping**: steganographic packets are randomly modified to invert the data bits carried in the message.

The chosen data corruption methods make it difficult to synchronize the covert data stream, thus classic error correction codes cannot be directly applied to the transmission, as the stream can be shifted and the meaning of the individual data and parity bits may be lost after random bit insertion, deletion, or flipping. Considering the above, to ensure the correct data transmission, we have designed and implemented solutions to achieve reliable covert data transmission in a channel with high network anomaly noise. In more detail, we use: *(i)* false network anomaly filtering, *(ii)* data bit indexing within continuous network stream, and *(iii)* error correction through redundancy. These solutions have been explained in detail in subsection 3.3.

5.3 Test set

During preparation of the test set, we have evaluated different DNCC configurations as well as various conditions of the network by introducing traffic anomalies which are similar to these observed in real-world communication networks (see section 5.2). As a result, we have produced a test set which contains

7657 test runs. The following summarizes the details of the test set configuration. Each experiment lasted 60 seconds and during experiments the simulated data corruption was set to 2, 5, 6.6, 10, and 20%, respectively. Moreover, the experiments have been recorded with 1, 2, or 3 steganographic modules enabled and they were instructed to modify roughly every second (50% transmission rates) or fifth (20% transmission rate) packet.

The experimental scenario involved client-server communication 1-to-1, 1-to-5, 5-to-1, and 5-to-5. In 5-to-1 and 5-to-5 scenarios, the number of actual steganographic clients can be 1, 2, or 5 (i.e., there could be 4, 3 or 0 clients which never had their packets modified). Similarly, in the 1-to-5 and 5-to-5 scenarios, the number of steganographic servers could be 1, 2, or 5 (i.e., there could be 4, 3 or 0 servers which never received modified packets). Only the network traffic from a steganographic client to a steganographic server is considered by the algorithm for packet modification. Moreover, clients were generating network traffic to the servers via HTTP requests sent with varying frequency, i.e., every 1, 5, or 10 seconds. Finally, some scenarios intentionally had the steganography disabled (the number of covert communication clients equals 0).

6 Results

During the first part of the conducted research, we measure the overall accuracy of each of the detection strategies, i.e., without differentiation of any specific type of DNCC traffic. Figure 9 presents ROC curves for the various variants of the Tree-based detection approach. During the performed experiments, we explore how the results change for various ratios of suspicious children in the tree necessary to mark a parent node as suspicious and, thus, in effect decide that the steganographic traffic appears in this test run. In Figure 9 we present the results when this parameter changes in the range from 0.1 to 0.9. As it can be seen, practically for all values, the resulting AUC is greater than 0.8 and the best results are achieved for 0.5 (AUC=0.845).

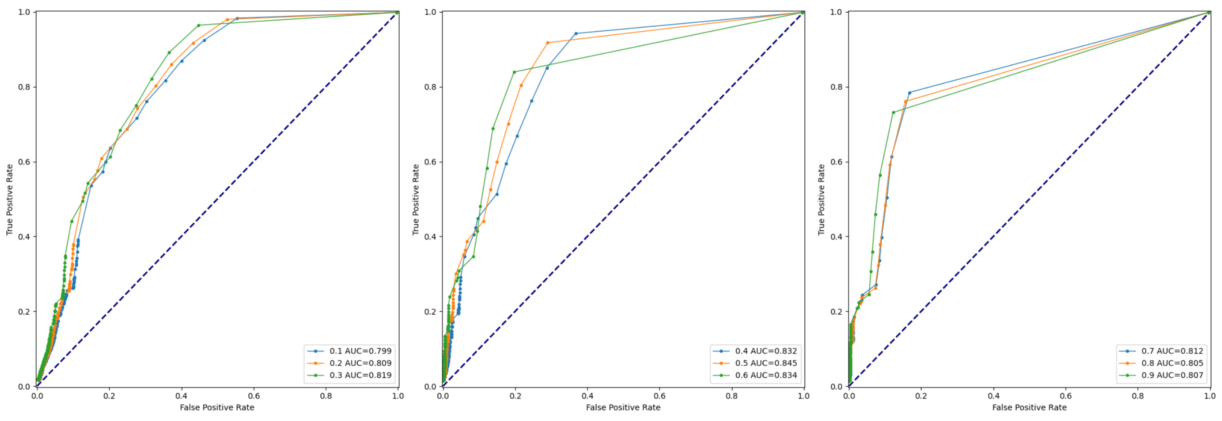
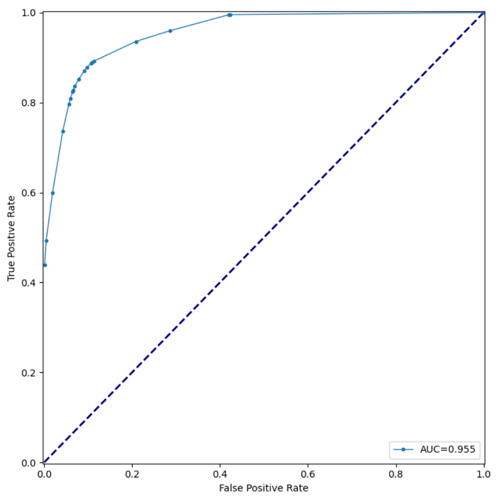
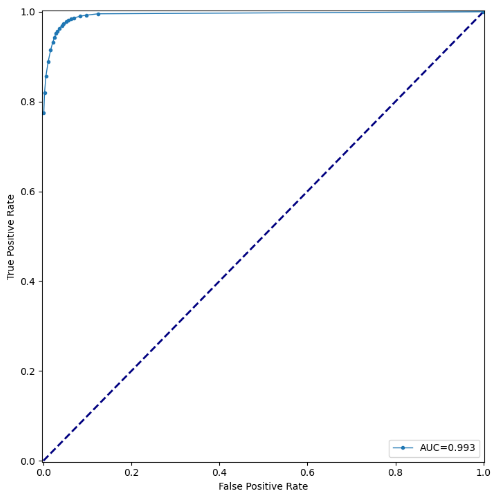


Figure 9: ROC plots for the Tree-based detection method for various ratios of suspicious children in the tree.

Next, Figure 10 illustrates ROC curves for detection strategies which utilize Machine Learning implemented in the DAI software. The subfigure (a) presents detection results for the 'Basic ML' approach where only preprocessed data are used for training and classification. On the other hand, the subfigure (b) showcases results of the 'Hybrid' method when the ML algorithm utilizes data enhanced with meta-data derived from the Tree-based solution (thus it can be considered as a combined concept). In both ML-based approaches (i.e., 'Basic ML' and 'Hybrid') the DAI software automatically tests various classification algorithms from the presented list: XGBoost GBM, LightGBM Random Forest, XGBoost



(a) 'ML Basic' approach utilizing only the pre-processed data.



(b) 'Hybrid' method using pre-processed data enhanced by the information from the tree of item-sets.

Figure 10: ROC plots presenting detection results for the ML-based detection strategies.

Dart, GLM, RuleFit, LightGBM, and FRTL. Results from the reports generated via DAI experiments show that for the final evaluation XGBoost GBM model has been chosen as the best performing one [3]. The measured performances of these models on the training data set are presented in Tables 2 and 3.

Table 2: Detection results for the 'ML Basic' method.

Scorer	Final ensemble scores on validation	Final ensemble standard deviation on validation	Final test scores	Final test standard deviation
ACCURACY	0.6828	0.0014	0.6833	0.0026
AUC	0.9399	0.0005	0.9404	0.0009
F1	0.6828	0.0014	0.6833	0.0026
F2	0.6828	0.0014	0.6833	0.0026
GINI	0.8799	0.0011	0.8809	0.0019
MCC	0.6375	0.0016	0.6381	0.0029

Table 3: Detection results for the 'Hybrid' approach.

Scorer	Final ensemble scores on validation	Final ensemble standard deviation on validation	Final test scores	Final test standard deviation
ACCURACY	0.7681	0.0018	0.7739	0.0018
AUC	0.9741	0.0003	0.9754	0.0003
F1	0.7689	0.0018	0.7739	0.0018
F2	0.7689	0.0018	0.7739	0.0018
GINI	0.9481	0.0007	0.9508	0.0007
MCC	0.7349	0.0021	0.7416	0.0021

The results presented above prove that the best detection method is the 'Hybrid' approach. This variant achieved AUC=0.993 and it outperformed 'Basic ML' scenario (AUC=0.955) and the previously proposed Tree-based scheme (AUC=0.845).

The next investigated issue concerns the detection accuracy for various steganographic transmission rates. Figure 11 showcases the accuracy results depending on the number of steganographically modified packets. In this figure, we present data gathered from all conducted experiments (background plot). Because the most interesting features are presented for lower values on the X-axis, in the small (foreground plot), we demonstrate also the magnified part of the plot for this area. The same approach will be used for other plots presented in this section as well. Due to the fact that during the presented research we utilized an additional error detection and correction scheme (see section 3.3 for a detailed description) on the X axis we indicate the number of modified packets and not the effective steganographic data rate. However, these two features are correlated as the more network packets contain secret data the higher resulting overall covert channel bandwidth.

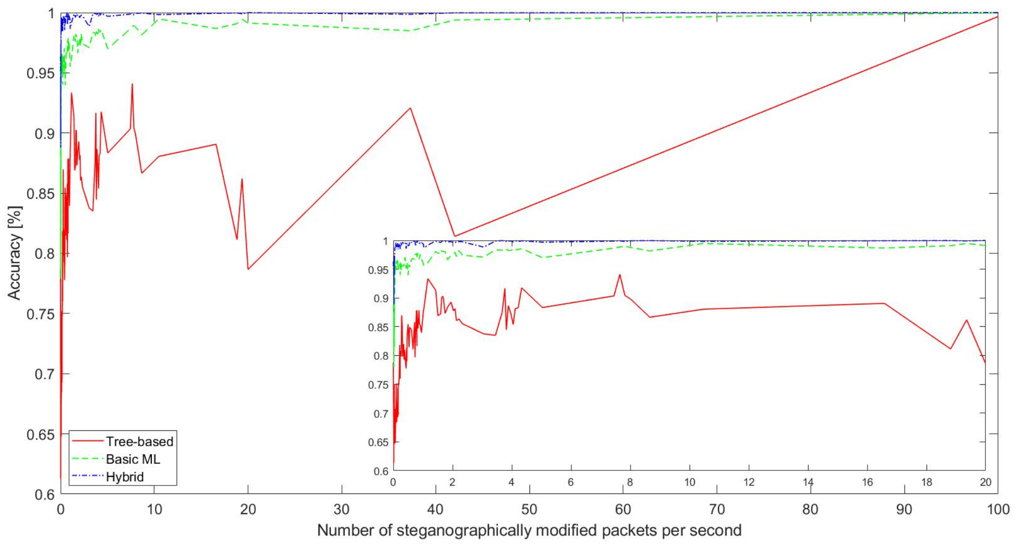


Figure 11: Accuracy of the detection depending on the number of the steganographically modified packets.

The presented results clearly show that higher steganographic rates are detected almost in all cases. However, it must be noted that reducing the steganographic rate allows the steganographer to 'stay under the radar' and remain undetected. Like in the first part of the conducted experiments, the best results have been obtained for the 'Hybrid' detection method.

The red solid curve in Figure 11 represents the results of the Tree-based detection method. We can observe that this approach has some instabilities which are associated with the detection of new frequent sets and reorganization of the tree. However, what is interesting to note is that such behavior is not transferred when using the same data for the 'Hybrid' detection strategy.

The last investigated issue concerns the detection accuracy for various numbers of steganographic clients depending on the number of modified packets. Figures 12-14 demonstrate the accuracy of different detection strategies for the varied number of steganographic clients (1, 2, or 5). In the next three plots, we present the results separately for each detection method, i.e., 'Tree-based', 'Basic ML', and 'Hybrid'. In the first plot, due to the high instability of the introduced Tree-based method, we cannot easily determine if this detection strategy performs better for a smaller or greater number of steganographic clients. However, as our previous research shows, the easiest to detect should be the DNCC configuration only with one client. In fact, such a situation is illustrated in Figure 12, only in the range from around 5 to 20 packet modifications per second, where this method achieves the best detection accuracy.

For the remaining two approaches, i.e., 'Basic ML' and 'Hybrid' results are much more stable and it can be easily observed that the resulting detection accuracy for five clients (blue curve in all three

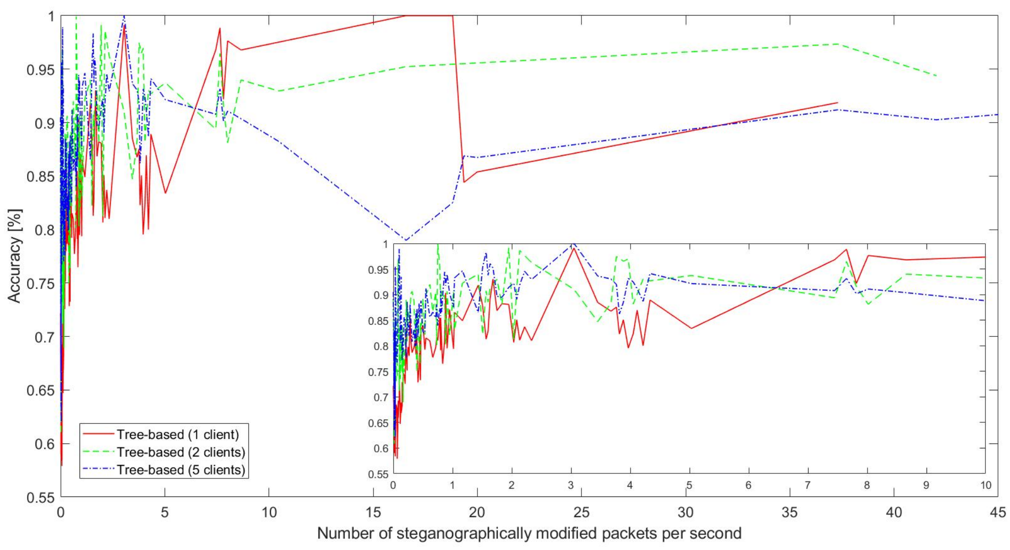


Figure 12: Accuracy of the detection depending on the number of the steganographically modified packets ('Tree-based' method).

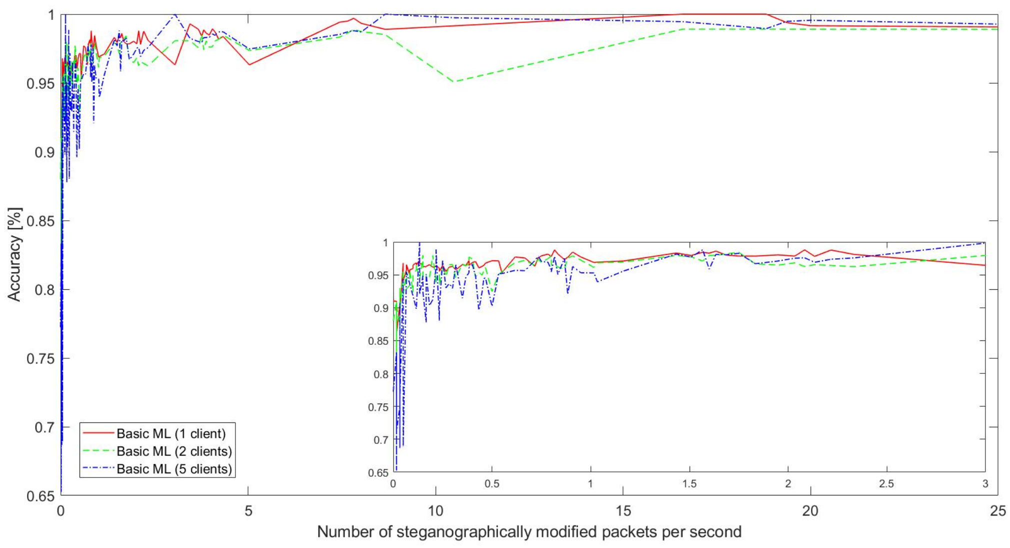


Figure 13: Accuracy of the detection depending on the number of the steganographically modified packets ('Basic ML' method).

plots) is the worst. As it was discussed earlier in this section, despite the high instability of the Tree-based method, the 'Hybrid' approach is one with almost invisible instability. These results confirm our previous research predictions that the utilization of the DNCC with a higher number of steganographic clients is harder to detect.

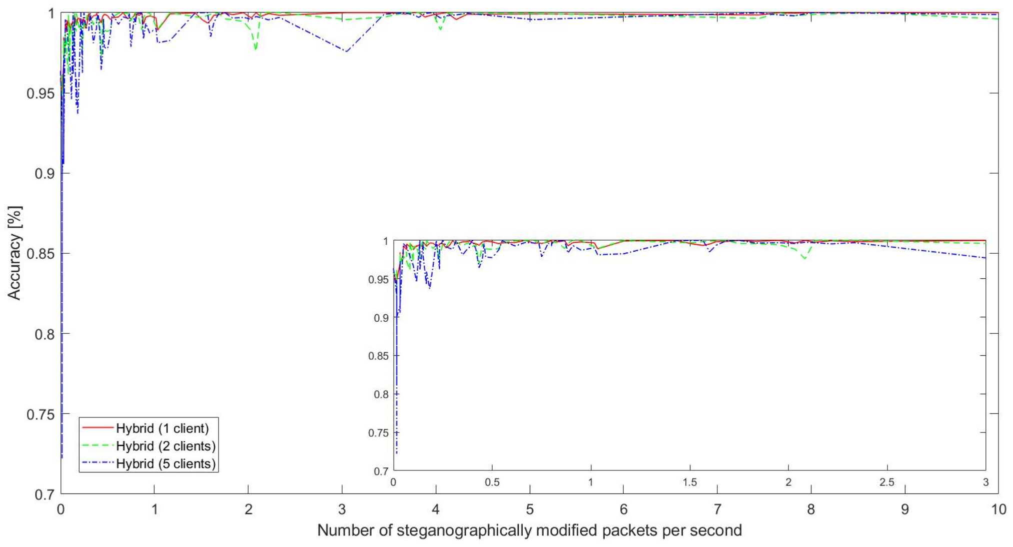


Figure 14: Accuracy of the detection depending on the number of the steganographically modified packets ('Hybrid' method).

7 Conclusions

In this paper, we present our extended research concerning the detection of steganographic transmissions. The proposed method can be used for the detection of simple covert communication attempts, i.e., where only one data hiding method in one network stream is used, as well as for more stealthy solutions, e.g., DNCC channels. To improve the detection accuracy after preprocessing phase data mining frequent sets are mined multiple times with various values of the *minimal support* parameters. Using the obtained results, a special tree structure which contains the discovered frequent item sets is constructed. In this paper, we introduced and described the detection method which is based on the analysis of such item sets tree. During the conducted experiments, we evaluated the proposed approach using steganographic traffic which was generated in the simulated IoT environment. In more detail, we proposed three detection strategies – one as described above and two others which utilize machine learning algorithms implemented in the DAI software. During these experiments, we compared ML algorithms – one which uses only preprocessed data and the second one which utilizes preprocessed data enhanced with the metadata obtained from the previously introduced tree-based method. Obtained research results proved that both ML-based approaches outperformed the original solution. However, it must be noted that the addition of the metadata generated by our original approach increases the resulting detection rate of ML-based detection. The measured AUC parameter for such a combined ('Hybrid') approach increased from 0.845 to 0.993. This results in a rise in performance by almost 15%. Our future work involves a more extensive study of both ML-based and hybrid solutions for steganographic transmission detection. First of all, we plan to investigate the accuracy and performance of other classes of ML algorithms. Secondly, we would like to utilize other data mining pattern discovery algorithms, which could produce more meta-data further used by ML solutions.

Acknowledgments

This material is based upon the work supported by the Air Force Office of Scientific Research under the award number FA9550-17-1-0254. The supported project is named CoCoDe – Covert Communication Detection. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

References

[1] K. Cabaj, L. Caviglione, W. Mazurczyk, S. Wendzel, A. Woodward, and S. Zander. The new threats of information hiding: The road ahead. *IT Professional*, 20(3):31–39, June 2018.

[2] K. Cabaj, W. Mazurczyk, P. Nowakowski, and P. Żórawski. Towards distributed network covert channels detection using data mining-based approach. In *Proc. of the 13th International Conference on Availability, Reliability and Security (ARES'18), Hamburg, Germany*, pages 12:1–12:10. ACM, August 2018.

[3] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'16), San Francisco, California, USA*, page 785–794. ACM, August 2016.

[4] M. Gregorczyk, P. Żórawski, P. Nowakowski, K. Cabaj, and W. Mazurczyk. Sniffing detection based on network traffic probing and machine learning. *IEEE Access*, 8:149255–149269, August 2020.

[5] W. Mazurczyk and S. Wendzel. Information hiding: Challenges for forensic experts. *Communication of the ACM*, 61(1):86—94, December 2017.

[6] W. Mazurczyk, S. Wendzel, and K. Cabaj. Towards deriving insights into data hiding methods using pattern-based approach. In *Proc. of the 13th International Conference on Availability, Reliability and Security (ARES'18), Hamburg, Germany*, pages 1–10. ACM, August 2018.

[7] W. Mazurczyk, S. Wendzel, M. Chourib, and J. Keller. Countering adaptive network covert communication with dynamic wardens. *Future Generation Computer Systems*, 94:712–725, November 2019.

[8] W. Mazurczyk, S. Wendzel, S. Zander, A. Houmansadr, and K. Szczypiorski. *Information Hiding in Communication Networks: Fundamentals, Mechanisms, Applications, and Countermeasures*. John Wiley & Sons, February 2016.

[9] A. Mileva and B. Panajotov. Covert channels in tcp/ip protocol stack - extended version. *Central European Journal of Computer Science*, 4(2):45–66, June 2014.

[10] P. Nowakowski, P. Zórawski, K. Cabaj, and W. Mazurczyk. Network covert channels detection using data mining and hierarchical organisation of frequent sets: An initial study. In *Proc. of the 15th International Conference on Availability, Reliability and Security (ARES'20), Virtual Event, Ireland*, pages 1–10. ACM, August 2020.

[11] P. L. Shrestha, M. Hempel, F. Rezaei, and H. Sharif. A support vector machine-based framework for detection of covert timing channels. *IEEE Transactions on Dependable and Secure Computing*, 13(2):274–283, April 2016.

[12] Y. Sun, L. Zhang, and C. Zhao. A study of network covert channel detection based on deep learning. In *Proc. of the 2018 2nd IEEE Advanced Information Management,Communicates,Electronic and Automation Control Conference (IMCEC'18), Xi'an , China*, pages 637–641. IEEE, May 2018.

[13] E. Tumoian and M. Anikeev. Network based detection of passive covert channels in tcp/ip. In *Proc. of the 2005 IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05), Sydney, NSW, Australia*, pages 802–809. IEEE, November 2005.

[14] S. Wendzel, S. Zander, B. Fechner, and C. Herdin. Pattern-based survey and categorization of network covert channel techniques. *ACM Computing Surveys*, 47(3):1–26, April 2015.

[15] I. You and K. Yim. Malware obfuscation techniques: A brief survey. In *Proc. of the 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA'10), Fukuoka, Japan*, pages 297–300. IEEE, November 2010.

[16] Q. Yuwen, S. Huaju, S. Chao, W. Xi, and L. Linjie. Network covert channel detection with cluster based on hierarchy and density. *Procedia Engineering*, 29:4175–4180, January 2012.

[17] S. Zander, G. Armitage, and P. Branch. An empirical evaluation of ip time to live covert channels. In *Proc. of the 2007 15th IEEE International Conference on Networks (ICON'07), Adelaide, South Australia, Australia*, pages 42–47. IEEE, November 2007.

[18] S. Zander, G. Armitage, and P. Branch. A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys & Tutorials*, 9(3):44–57, September 2007.

[19] J. Zhai, G. Liu, and Y. Dai. A covert channel detection algorithm based on tcp markov model. In *Proc. of the 2010 International Conferrence on Multimedia Information Networking and Security (MINES'10), Nanjing, China*, pages 893–897. IEEE, November 2010.

Author Biography

Piotr Nowakowski received his B.Sc. in electronics (2016) and M.Sc. in computer science (2018) from the Faculty of Electronics and Information Technology at Warsaw University of Technology (WUT), where he currently pursues a PhD degree. His research interests include: network security, reverse engineering, parallel processing and computer graphics.

Piotr Żórawski received the B.Sc. and M.Sc. degrees in computer science from the Faculty of Electronics and Information Technology, Warsaw University of Technology (WUT), in 2016 and 2019, respectively, where he is currently working as a teaching assistant. His research interests include computer and network security, dynamic malware analysis, and reverse engineering. He took part in projects for EU and U.S. Air Force.

Krzysztof Cabaj holds M.Sc (2004), Ph.D. (2009) and D.Sc. (habilitation) (2019) in computer science from Faculty of Electronics and Information Technology, Warsaw University of Technology (WUT). He is currently hired as University Professor at Institute of Computer Science, WUT. Former instructor of Cisco certificated Academy courses: CCNA Routing & Switching, CCNA Security and CCNP at International Telecommunication Union Internet Training Centre (ITU-ITC). His research interests include: network security, honeypots, dynamic malware analysis, data-mining techniques, IoT and Industrial Control Systems security. He is author or co-author of over 70 publications, and supervisor of more than twenty five M.Sc. and B.Sc. degree theses in the field of information security. He took part in over a dozen research projects, among others for EU, ESA, Samsung, US Army and US Air Force. Co-leader of Computer Systems Security Group at Institute of Computer Science.

Wojciech Mazurczyk received the B.Sc., M.Sc., Ph.D. (Hons.), and D.Sc. (habilitation) degrees in telecommunications from the Warsaw University of Technology (WUT), Warsaw, Poland, in 2003, 2004, 2009, and 2014, respectively. He is currently a University Professor with the Institute of Computer Science at WUT and a head of the Computer Systems Security Group. He also works as a Researcher at the Parallelism and VLSI Group at Faculty of Mathematics and Computer Science at FernUniversitaet, Germany. His research interests include bio-inspired cybersecurity and networking, information hiding, and network security. He is involved in the technical program committee of many international conferences and also serves as a reviewer for major international magazines and journals. From 2016 he is Editor-in-Chief of an open access Journal of Cyber Security and Mobility, and from 2018 he is serving as an Associate Editor of the IEEE Transactions on Information Forensics and Security. He is also a Senior Member of IEEE.