Hindawi
Security and Communication Networks
Volume 2022, Article ID 2254959, 11 pages
https://doi.org/10.1155/2022/2254959

Research Article

CCgen: Injecting Covert Channels into Network Traffic

Félix Iglesias, Fares Meghdouri, Robert Annessi, and Tanja Zseby

TU Wien, Institute of Telecommunications, Gusshausstraße 25/E389, 1040 Vienna, Austria

Correspondence should be addressed to Félix Iglesias; felix.iglesias@tuwien.ac.at

Received 25 January 2022; Revised 11 March 2022; Accepted 20 April 2022; Published 29 May 2022

Academic Editor: Wenjuan Li

Covert channels are methods to convey information clandestinely by exploiting the inherent capabilities of common communication protocols. They can be used to hide malware communication as part of cyber attacks. Here, we present CCgen, a framework for injecting covert channels into network traffic that includes modules for common covert channels at the network and transport layer and allows a smooth integration of novel covert channel techniques. Our tool—openly available and implemented in Python—enables the operation on-the-fly in live communications as well as the manipulation of network traffic packet captures. We evaluate a first prototype by generating a varied assortment of covert channels based on state-of-the-art techniques and check their detectability with *Suricata*, a popular, open-source intrusion prevention and detection system. The injected covert channels remain mostly undetected. Our proposal fills a gap within the diversity of openly available tools for cybersecurity research and education. It builds a flexible environment for experts to test analysis algorithms, thus also enabling advanced training in applied network steganography.

1. Introduction

Covert channels enable the exchange of information in secrecy through legitimate communication channels. This exchange is performed in a way that it is undetectable to anyone other than the sender and the receiver of the information. The concept is equivalent to that of steganography, which is merely hiding a secret message inside of a nonsecret carrier. The main difference lies in the fact that a covert channel is considered a type of computer attack that violates security policies.

Nevertheless, in practice, these two concepts overlap and are almost synonymous from a scientific-technical perspective. In their most applied form, covert channels are mainly used for criminal activities and are therefore a major challenge for cybersecurity. This is clearly reflected in Mazurczyk and Caviglione's work [1], which explores a collection of malware that use information hiding. More recently, Cabaj et al. study real-world threats observed between 2011 and 2017 [2] and conclude that data hiding techniques are being more and more implemented by malware while claiming against the lack of effective global countermeasures. Also, note that covert channels rapidly extend to new network environments, such as streaming media, blockchain, or IPv6 [3]. Not surprisingly, there has been growing interest and concerns about the use of covert channels to attack and steal information from modern industrial control networks [4] and cyber-physical systems in general [5] and particularly in scenarios related to critical infrastructures, such as electric vehicle charging systems [6] and smart grids [7].

It is important to mention that there are also positive applications for steganography and covert channels. Among other examples, it can be used to effectively circumvent censorship [8], to implement digital watermarking in VoIP traffic security [9], to develop traceback techniques that identify and isolate network attackers [10], or to build defensive techniques that allow transmitter authentication in low-secure communications within cyber-physical systems [11].

When exploring the theory about covert channels, we can find multiple surveys and overviews that make efforts to show more or less complete taxonomies and classifications. Zhiyong and Yong classify covert channels by means of entropy analysis in [12]. Specifically, for network and application protocols, Zander et al. collect and show techniques

and countermeasures in [13]. In [14], covert channels are analyzed and classified based on the statistical challenge that they present for the detection. Around the same time, Wendzel et al. produce one of the most comprehensive works on covert channels [15], in which they review 109 techniques published between 1987 and 2013 and structure them into a classification of 11 different patterns. Thus, Wendzel et al. set a precedent for the evaluation of future proposals in a way that the focus of detection systems is on patterns rather than on specific techniques.

In a more recent survey [16], Caviglione highlights shortcomings in the field of study due to insufficient formalization and delimitation of the problem space, while stressing the importance of increasing the awareness concerning the security risks caused by network covert channels. This observation is important because it shows that security systems are not adequately designed to detect covert channels and that there is insufficient training among security experts.

Note that, although there is a lot of literature presenting injection techniques and detection methods, there are only a few openly available tools for experimenting. In this respect, Racei et al. present a framework (available on demand) for generating covert channel algorithms in [17]. CCHEF [18] is the main precedent of our proposal. Implemented in C, it allows the evaluation of IPv4 covert channels in both online and offline modes, including storage and timing channels and providing a base framework to which new methods can be added. The last available version of CCHEF includes six techniques for using the time-to-live field (TTL) as a carrier, one for the IP identification field (IPID), and two for masking covert symbols within packet interarrival times (IAT). Beyond isolated repositories on the Internet that offer tools for implementing specific techniques, the most recent framework for generating covert channels is presented by Zuppelli and Caviglione under the name of "pcapStego" in 2021 [19]. So far, pcapStego includes several techniques specifically devised for IPv6 traffic, but the tool is in constant development and authors work on more functionalities and flexibility in future releases.

Our proposal—the CCgen framework—joins these initiatives and improves the state of the art with a tool capable of implementing 18 base techniques published in related journals and conferences, adding additional configuration options that increase the variability of the generated covert channels. New techniques can be easily added as Python-script plugins. CCgen supports both online and offline modes and is suitable for both injection and recovery of covert messages. Moreover, the framework is able to automatically inject multiple covert channels in the same capture (PCAP) as well as perform simultaneous online injections. Such wrapping functionalities are very useful for generating datasets to test detection algorithms, which also paves the way for setting environments to train experts in information hiding and network steganography. In this respect, our framework has been developed to meet testing and research requirements for cybersecurity projects on critical infrastructures and can also provide a basis for cyber security teaching (https://www.nt.tuwien.ac.at/research/communication-networks/network-security-laboratory/#nsII) [20].

For the evaluation of this work, we checked covert channels generated with the CCgen framework with *Suricata* (https://suricata.io/), an updated, popular network intrusion detection system (NIDS). The injected covert channels pass unperceived in most cases, confirming that standard security measures are not properly prepared to face the threat of covert channels.

2. Covert Channel Techniques

Many different techniques have been proposed for the injection of covert channels into diverse communication protocols. In this section, we reduce the scope to the techniques available in the first release of CCgen and use the classification in [14] to present them.

2.1. Value to Symbol (v2s) Correspondence. In a value-to-symbol correspondence, one field value is bound to a unique symbol of the covert message. For example, if the size of IP packets is used to contain a binary covert message, a packet with 40-byte size could stand for "0," whereas a packet with 60-byte size could represent the symbol "1." Cases where more than one value (but not a range of values) is assigned to a single covert symbol are also included in this category. The included techniques are:

(1) *ipflags* creates a binary channel by manipulating the Reserved bit and the Don't Fragment bit of the IP Flags field. This technique is inspired by the work of Ahsan and Kundur [21].

(2) *ipid* uses the 8 most significant bits (MSB) of the IP Identification field. It additionally clears the Don't Fragment bit of the IP Flags field to avoid undesired conflicts. This technique has been proposed in several publications, e.g., [22, 23].

(3) *iplen* manipulates the size of the IP Total Length field. Using the length of data blocks in the network has already been proposed by Girling [24]. In the implemented version, the actual packet length is not manipulated, but just the IP total length field. The number of lengths to use (i.e., the number of potential hidden symbols) is a configurable parameter.

(4) *ipproto* follows the ideas introduced by Wendzel and Zander [25], and hence, it works by using the IP Protocol field as a carrier. The basic version of this technique is intended to switch between TCP and UDP protocols, allowing binary covert channels.

(5) *iptos* exploits the traditionally unused bits of the Type of Service (ToS) field of the IP datagram. It conveys up to six bits per packet, since the two less significant bits (LSB) are reserved. This technique is presented by Postel in [26]. Nowadays, ToS bits are relevant for Differentiated Services (DSCP) [27] and Explicit Congestion Notification (ECN) [28], but are still rarely used.

(6) *srcport* is a simplification of the method proposed by Gimbi et al. for TCP and UDP datagrams [29]. Here, port numbers are directly mapped with a binary encoding of ASCII symbols. The original technique is implemented in the derivative group, technique number 2 in Section 2.4.

(7) *ttl_v2s* uses the TTL field of the IP datagram to hide a binary covert channel as introduced in [30].

(8) *ipaddr* is also proposed by Girling [24], and here, the destination IP address field is the carrier. Note that, in this method, the receiver of the covert communication is not a destination device, and instead, it is placed in a location from which it is able to sniff traffic in the range of the destination addresses used to mask the covert channel.

2.2. Ranges to Symbol (r2s) Correspondence. This type of covert channels maps a range of values with a covert symbol. For instance, taking again the example with the size of IP packets, any packet with a size below or equal 50 bytes could stand for "0," and then, packets above 60 bytes would be "1." Techniques in this group are

(1) *ttl_r2s* uses the TTL field of the IP datagram to hide a binary channel. Both "0" and "1" symbols are represented by a different range of TTL values. This implementation is inspired by the authors of [31].

(2) *srcport_r2s* also follows the work by Lucena et al. [31] but is applied to the Source Port field of the IP datagram after ensuring that packets belong to the TCP or UDP protocol.

(3) *iplen_r2s* stems from the examples and discussions in [24] but allows different lengths to be mapped to the same symbol in order to hamper the detection of the covert channel.

2.3. Containers. We consider that a covert channel is hidden in a container field when the amount of covert information sent per packet is more than 1 byte. Container fields are often accompanied by marker fields, which inform the receiver about the existence of covert information in a given packet. The first release of CCgen comes with two container techniques.

(1) *ipfragment* uses the Fragment Offset field of the IP frame. Therefore, each packet can contain up to 13 bits of covert information. This technique additionally clears the Don't Fragment bit and sets the More Fragment bit of the IP Flags in each modified packet. By doing so, the values of the Fragment Offset would remain unused from the perspective of a conventional communication, so they can be exploited to hide covert data. This method for creating covert channels is discussed by Goher et al. in [32].

(2) *urgent* uses the URG bit of the TCP Flags field and the Urgent Pointer field of the TCP frame. The URG bit acts as a marker: when it is set to "0," the Urgent Pointer contains up to 16 bits of hidden information. This technique has been proposed by Fisk et al. in [33].

2.4. Derivative. Derivative channels occur when covert symbols are not directly hidden in the value of the field but in how this value changes throughout successive packets. They are:

(1) *ttl_dev* manipulates the TTL field of the IP datagram to hide a binary derivative covert channel. Instead of mapping "0" and "1" symbols directly to values, the value of the field switches whether the symbol is "1" and remains the same if the symbol to send is "0." This is one of the methods explained by Zander et al. in [34].

(2) *srcport_dev* implements one of the techniques proposed by Gimbi et al. in [29] and therefore manipulates the Source Port field of TCP or UDP datagrams to convey ASCII symbols in value increments. Instead of establishing the Source Port value at random (as done by default), its value is set as the value of the previous packet plus the decimal equivalence of the ASCII symbol to be covertly sent. This technique additionally requires to build a loop by connecting maximum and minimum allowed values.

2.5. Covert Timing Channels. Covert timing channels use time properties—mainly interarrival times (IATs)—as carriers. Note that most techniques manipulate Inter Departure Times (IDTs) in origin, which transform into IATs in destination (in short, IAT = IDT + transmission_delays). An analytic study of the characteristics of different covert timing channel techniques is discussed in [35]. By default, CCgen comes with three timing techniques.

(1) *timing_ber* agrees on two different IDTs to mask binary symbols. This method is presented by Berk et al. in [36].

(2) *timing_gas* sets a time threshold to discriminate "0" and "1" symbols. If a given IAT is above the threshold, it will mark "1", and "0" if below. This method is discussed by Gasior et al. in [37].

(3) *timing_sha* is originally designed by Shah et al. to interfere with legitimate communications in different types of devices [38]. It uses a base sample interval and adds some delay to IDTs. A covert "1" or a "0" is interpreted depending on if a given IAT is divisible by the interval or only half of it.

3. The CCgen Framework

In this section, we describe the CCgen framework together with other tools and utilities developed for its exploitation. The CCgen framework, documentation, and examples are completely open-source and available for downloading

in our stable repository (https://github.com/CN-TU/py_CCgen).

The core part of the CCgen framework is the CCgen module, which is in charge of performing the actual injection and extraction of covert channels. Around this core module, other modules and functions are added to enable a flexible application and an encapsulated usage. We describe first the core and later the remaining parts of the framework.

3.1. The Core Module: CCgen. The CCgen module is built upon the Python-based Scapy library. From the description given by their developers, Scapy "is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, store or read them using PCAP files, match requests and replies, and much more" (https://pypi.org/project/scapy/).

CCgen needs to be called together with a configuration file. The configuration file contains information about four different sets of parameters, namely:

Files: this set of parameters indicates files related to the message to be covertly sent, the PCAPs to use as inputs and outputs when working in the offline modus, or the file that contains mapping values and specific parameters related to the selected covert channel technique.

Filter: here, parameters identify datagram field values that act as filters or conditions to match the desired covert channel flow, either for injection or extraction. Some examples of parameters to set here are src_ip, dst_ip, src_port, dst_port, or proto.

Channel: among these parameters, we find the covert channel technique to use or decode the expected or desired number of bits per packet and the protocol layer in which the covert channel is located.

Iptables sets the queue number and input/output parameters required for the online operation of CCgen.

In addition to a configuration file, CCgen must be set in one of its four different operational modi. The selected modus will also determine the parameters that are required in the configuration file. Figure 1 shows a schematic overview of CCgen in which operational modi are separately visualized. They are as follows.

3.1.1. Offline Injection. When an offline injection is run, CCgen uses the Scapy modules PcapReader() and PcapWriter(). PcapReader() reads the clean PCAP specified in the configuration file, and PcapWriter() creates a copy of the PCAP but with changes in the datagrams specified by filter conditions. The selected covert channel technique establishes the callback function for changing values by consulting the configured mapping and data file.

3.1.2. Offline Extraction. In the offline extraction, only the PcapReader() module is used to explore the PCAP given as input. When filter conditions are matched, CCgen tries to extract values by calling the configured covert channel technique and mapping conditions. Extracted values are saved in an output data file.

3.1.3. Online Injection. CCgen uses IPtables in this modus to forward selected packets from a network interface to the NFQueue (https://pypi.org/project/NetfilterQueue/). Using a callback function, Scapy reads packets and manipulates them based on the configuration file. After the packet has been manipulated, it is returned to the network interface and sent to its original destination via sockets (https://docs.python.org/3/library/socket.html). The filters in the configuration file also include the queue number, allowing for multiple concurrent queues and thus multiple online covert channels at the same time. The generation of packets depends on a third host that is intercepted by CCgen and performs packet manipulation. In our tool, this is done by a Spammer module by default, which provides a continuous stream of packets with random features and payloads.

3.1.4. Online Extraction. This operational modus employs the same libraries as the online injection mode. However, rather than listening for specific packets and forwarding them to the NFQueue, it simply reads them and triggers events when packets from the outside are detected and match the receive filter. An additional callback function set by the covert channel technique extracts the hidden value and saves it in a data file.

An important detail of CCgen to highlight is that, whenever a packet is modified by the tool, packet checksums are recalculated. Other possible dependencies derived from the manipulation of the packets must be controlled in the specific implementation of each covert channel technique.

3.2. Encapsulating CCgen. Since the CCgen core allows for the injection or decoding of a single covert channel at a time, the framework is designed to wrap it, therefore allowing the automatic injection of a collection of covert channels in different flows of the same capture (PCAP—offline modus). This is necessary mainly to create traffic traces with which to extensively test detection algorithms and perform sensitivity analysis on different parameters, configurations, or generation methods. Hence, the encapsulation of CCgen by the framework is essential to create environments for testing network steganography techniques and detection methods.

Note that CCgen—when working in the offline modus—does not create new traffic traces but uses existing ones to hide covert communications. Therefore, the CCgen framework is designed to search for matching flows in a given PCAP according to a set of users' specifications (e.g., the techniques to be used and the messages to be concealed). The CCgen framework incorporates diverse utility tools to facilitate and accelerate the generation of covert channels and subsequent validation processes. Figure 2 shows a diagram that embraces normal operation steps and functions, namely:

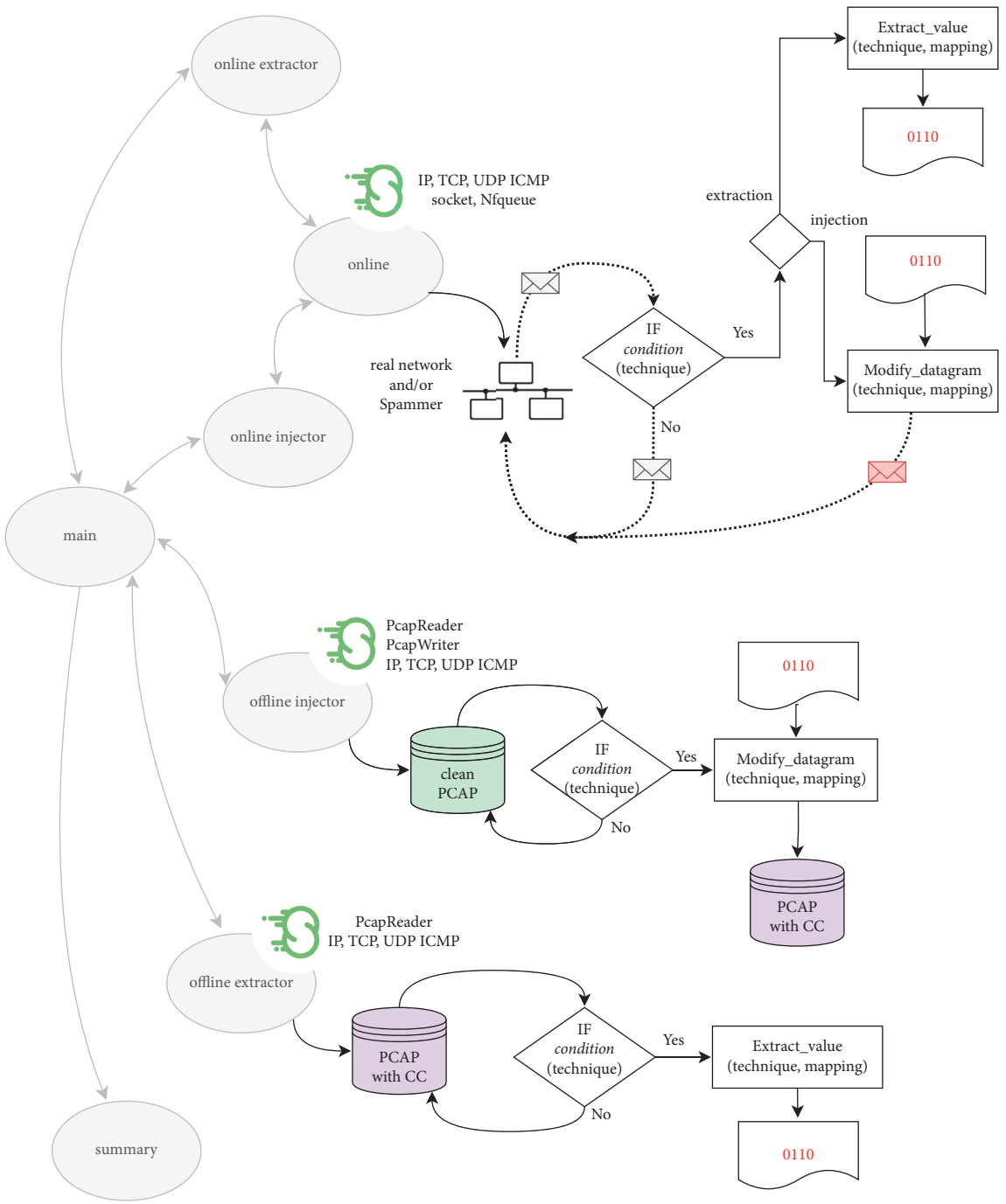

Figure 1: Schema of CCgen operational modi.

(1) *Generation of Binary Files.* Among the available tooling, the framework includes the *text2bin* script to transform text files into binary sequences. CCgen needs to be fed with binary sequences from which it extracts and adapts covert symbols based on the selected mapping and technique.

(2) *Generation of CCgen Configuration Files.* The *gen-CCconfigs* script takes a global wrapper configuration file and a PCAP capture as inputs. Each row of the wrapper configuration file establishes the parameterization of a covert channel model to inject. It contains the following fields:

(i) message_file: the file with the binary sequence to send.

(ii) technique: the technique to covertly convey information.

(iii) key: the type of flow to search in the PCAP capture; for instance, 2 tuple or 5 tuple.

(iv) mapping: the file with specific parameters and/or the symbol-value mapping to use. For example,

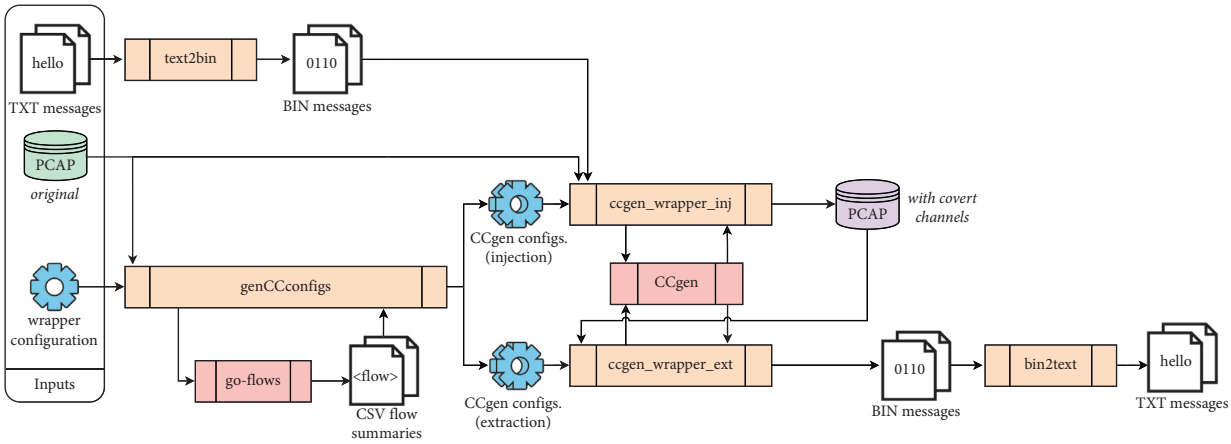

Figure 2: Operation diagram of the CCgen framework.

binaries symbols "0" and "1" are transformed into header values "60" and "120," respectively.

(v) bitspkt: the number of hidden bits that a packet (or IAT) transmits.

(vi) constr is used to specify additional constraints in the flow search. For example, "tcp" stands for only using TCP flows.

(vii) rep: the number of different flows in which the current configuration is to be applied.

genCCconfigs calls go-flows (go-flows is a highly customizable general-purpose flow exporter, openly available for usage and test in https://github.com/CN-TU/go-flows) to extract flow summaries of the given PCAP by taking into account diverse flow keys and therefore generating pools of potentially available flows. Later, it randomly binds the requirements in the wrapper configuration file with flows that match such requirements and removes these flows from the pools afterwards. The outputs of the tool consist of a set of CCgen configuration files, both for injecting covert channels (in the original, clean PCAP) and for extracting them later (from the final, manipulated PCAP).

(3) *Injection of Covert Channels*. The *ccgen_wrapper_inj* script takes the outputs for injection configuration of genCCconfigs and consecutively calls CCgen to inject one by one the desired covert channels. Once the process is finished, a final, manipulated capture (PCAP) with all covert channels is created.

(4) *Extraction of Covert Channels*. The *ccgen_wrapper_ext* script takes the outputs for extraction configuration of genCCconfigs and consecutively calls CCgen to extract one by one the previously injected covert channels from the manipulated PCAP. Extracted covert channels are saved as binary sequences.

(5) *Checking Extracted Binary Sequences*. The *bin2text* script transforms binary sequences into text sentences, therefore being useful to check if covert channels were properly injected and extracted.

4. Evaluation

In this section, we evaluate the CCgen framework for injecting covert channels into network traffic. The evaluation consists of three steps, which are schematized in Figure 3 and cover different goals.

4.1. Injection of Covert Information. We first use the CCgen framework to create a modified PCAP with covert channels. For this, we inject a collection of different covert channels into a clean PCAP. Packet traces used to belong to the WIDE backbone (https://mawi.wide.ad.jp/mawi/), which is maintained by the MAWI working group of the WIDE project. The MAWI WIDE project daily publishes real network traffic traces for research purposes. For convenience, we use a portion of traffic between 300 and 400 Mb in PCAP format. The framework automatically selects flows for injecting 20 covert channels according to the configuration in Table 1. In the table, we can see that each covert channel is used for a different message, named from cod00 to cod19 (all of them being text files between 1.5 kB and 2.5 kB, i.e., around 2000 ASCII characters in average). The information within codXX files has been taken from the URLhaus CSV database dump (https://urlhaus.abuse.ch/), which "contains malware URLs that are either actively distributing malware or that have been added to URLhaus within the past 90 days" (Nov. 2021 version). An example of a covert message can be seen in Listing 1. The techniques shown in Table 1 are explained in the covert channel techniques section. The key column shows the type of fitting flow key; for example, "1tup" means injecting a covert channel in all packets matching a given IPsrc; the *mapping* column shows the mapping file used among the collection provided with CCgen by default; the *bpp* column informs about the number of covert bits per packet (or IAT) reflected by the previous mapping; and the *constr* column stands for the filter constraint, which is sometimes imposed by the desired technique as a prerequisite. The remaining columns show the detection performance of the IDS used on the clean and manipulated PCAPs.

Listing 1: Example of covert message masked within the evaluation covert channels.

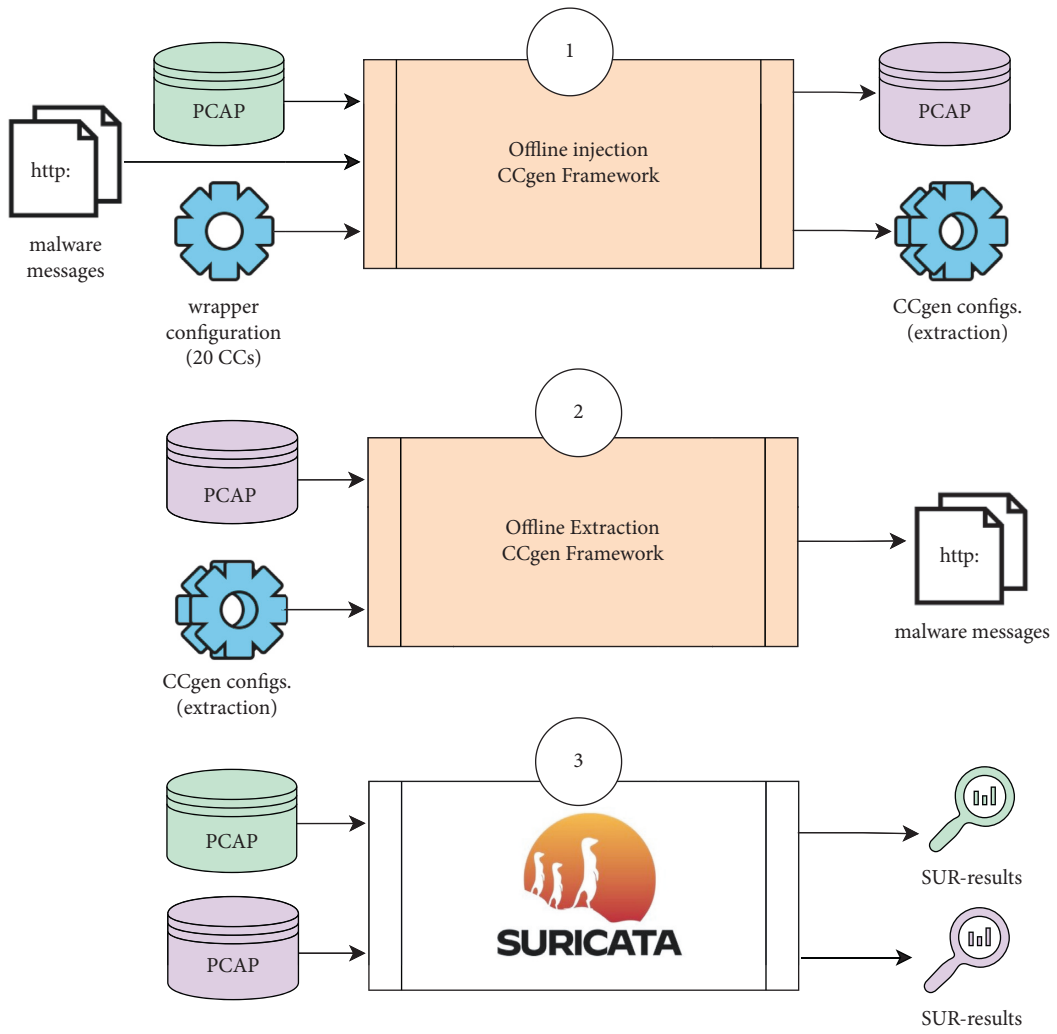

Figure 3: Phases of the evaluation tests.

Table 1: CCgen wrapper configuration used in the evaluation.

Flow	Message	Technique	Key	Mapping	bpp	Constr	*Suricata* Clean PCAP	*Suricata* Modif. PCAP
1	cod00	ipflags	2tup	bin	1	None	No-alarm	No-alarm
2	cod01	ipid	5tup	8 bits	8	None	No-alarm	No-alarm
3	cod02	iplen	2tup	bin_off60	1	None	Invalid-ack	No-alarm
4	cod03	iplen	3tup	8 bits_off50	8	None	No-alarm	No-alarm
5	cod04	ipproto	2tup	ipproto_1 bit	1	None	Out-of-window	Out-of-window, UDP-too-small
6	cod05	iptos	5tup	6 bits	6	None	No-alarm	No-alarm
7	cod06	srcport	3tup	8 bits_off1k	8	tcp	No-alarm	No-alarm
8	cod07	srcport	3tup	8 bits_off1k	8	udp	No-alarm	No-alarm
9	cod08	ttl_v2s	2tup	ttl_v2s	1	None	No-alarm	No-alarm
10	cod09	ipaddr	1tup	ipaddr_8 bits	8	None	Scan/brute-force	Scan/brute-force
11	cod10	ttl_r2s	2tup	ttl_r2s	1	None	No-alarm	No-alarm
12	cod11	srcport_r2s	3tup	srcport_r2s	1	tcp/udp	No-alarm	No-alarm
13	cod12	iplen_r2s	2tup	len_r2s	1	None	No-alarm	No-alarm
14	cod13	ipfragment	5tup	13 bits	13	None	No-alarm	No-alarm
15	cod14	urgent	5tup	16 bits	16	tcp	No-alarm	No-alarm
16	cod15	ttl_dev	2tup	ttl_dev	1	None	No-alarm	No-alarm
17	cod16	srcport_dev	3tup	srcport_dev	8	tcp/udp	No-alarm	No-alarm
18	cod17	timing_ber	2tup	timing_ber	1	None	No-alarm	No-alarm
19	cod18	timing_gas	2tup	timing_gas	1	None	No-alarm	No-alarm
20	cod19	timing_sha	2tup	timing_sha	1	None	No-alarm	No-alarm

Keys: IPsrc (1tup) IPsrc, IPdst (2tup) IPsrc, IPdst, Protocol (3tup) IPsrc, IPdst, Protocol, SrcPort, DstPort (5tup).

(1) 13410,1806361,2021-11-22 17:20:06,http://163.179.166.235:59512/Mozi.m,offline,2021-11-23 06:XX:XX,malware_download,"elf,Mozi",https://urlhaus.abuse.ch/url/1806361/,lrz_urlhaus

(2) 12763,1807008,2021-11-22 22:22:04,http://186.33.79.196:59772/Mozi.m,offline,2021-11-23 02:XX:XX,malware_download,"elf,Mozi",https://urlhaus.abuse.ch/url/1807008/,lrz_urlhaus

(3) 24779,1794349,2021-11-16 11:03:11,https://mtadwq.by.files.1drv.com/y4mvTveuCHvZp4cB2GspJ23YNuA9fDs1gLHq7X7ZzimOgrdJ9fRoVMlN854uXNTWW3c68Tu4iCtRLTgt1tycq6REM3QBaRmdddzP22KiteySsznw6HAu4nmxoDYy7jW-78xYRZJwoffU15Ae68noIzslLMYclGdrKvMT32zLlAYTxhcC7c-7t1Wqz4xURSh-JYETrSLVGZbJfJCzKzlGw/new-documents-2047.iso,offline,2021-11-16 12:XX:XX,malware_download,"bazaloader, BazarLoader",https://urlhaus.abuse.ch/url/1794349/,Cryptolaemus1

(4) 45339,1773124,2021-11-10 15:50:08,http://49.89.72.147:57938/mozi.m,offline,2021-11-13 09:XX:XX,malware_download, None,https://urlhaus.abuse.ch/url/1773124/,tammeto

(5) 20358,1799037,2021-11-18 06:01:09,http://121.23.78.127:37982/i,online,2021-12-02 09:XX:XX,malware_download,"32-bit,arm,elf,Mozi",https://urlhaus.abuse.ch/url/1799037/,geenensp

(6) 26699,1791910,2021-11-16 08:14:13,http://27.6.196.164:59391/i,offline,2021-11-16 09:XX:XX,malware_download,"32-bit,elf,mips, Mozi",https://urlhaus.abuse.ch/url/1791910/,geenensp

(7) 29681,1788923,2021-11-15 14:09:05,http://125.40.128.229:47460/i,offline,2021-11-16 05:XX:XX,malware_download,"32-bit,elf,mips, Mozi",https://urlhaus.abuse.ch/url/1788923/,geenensp

(8) 16009,1803760,2021-11-21 23:20:06,http://59.93.19.102:41706/i,offline,2021-11-22 00:XX:XX,malware_download,"32-bit,elf,mips, Mozi",https://urlhaus.abuse.ch/url/1803760/,geenensp

(9) 12069,1807703,2021-11-23 02:34:07,http://117.194.164.60:54204/Mozi.m,offline,2021-11-24 00:XX:XX,malware_download,"elf,Mozi",https://urlhaus.abuse.ch/url/1807703/,lrz_urlhaus

(10) 3780,1816041,2021-11-25 10:08:13,http://27.6.253.4:45831/Mozi.m,offline,2021-11-25 12:XX:XX,malware_download,"elf,Mozi",https://urlhaus.abuse.ch/url/1816041/,lrz_urlhaus

4.2. Extraction of Covert Information. To check that the injection was properly performed, we use the extraction functionality of the CCgen framework to retrieve covert messages from the modified PCAP capture. Note that the injection and extraction processes are completely independent of each other, so the later can be used to evaluate the former. However, for the extraction, CCgen requires configuration files with proper information to accurately fetch the covert message in the given traces, which in a real scenario would be agreed with the receiver of the covert channel. The CCgen framework automatically generates consistent injection and extraction configuration files to facilitate testing and evaluation.

4.3. Analysis with IDS (Suricata). Once we confirm that covert channels were correctly injected, we analyze both—the original, clean PCAP and the modified PCAP with the popular, open-source IDS *Suricata* (https://suricata.io/). Since there are no specific rules for covert channels in the official *Suricata* sources, sets of rules and parameters are adjusted by the native *Suricata* configuration tool after updating (update date: Jan. 2022). Rulesets and providers are shown in Table 2.

5. Results and Discussion

In conformity with the first two steps described in the Evaluation section, all covert channels created during the injection phase were correctly retrieved in the extraction phase. Potential failures when using the CCgen framework—particularly in the offline modus—are common due to two reasons: (a) possible errors in the codes of the implemented technique (or in the selected mapping) or (b) the impossibility to find a flow in the target PCAP that matches the requirements of a covert channel configuration.

Depending on the technique used, note that the retrieved covert channel in the extraction could display a queue of strange characters (once decoded) after the actual covert message. This is due to the fact that CCgen, during extraction, does not know how many covert symbols to expect and therefore extract information from the complete targeted flow. As an example of this behavior, Figure 4 shows a snapshot of a toy sample in which the technique used for both cases was ipaddr. The differences observed in the extraction are due to variations in the configuration and in the flow selected in each case.

When comparing *Suricata* outputs linked to the selected flows in the original and manipulated PCAPs, we find only small differences. Table 1 shows such results. Note that only in one of the cases of IP protocol manipulation (flow 5), *Suricata* triggered an alarm that was not detected in the original PCAP. This alarm has to do with a manipulated packet that, after changing its protocol to UDP, its size resulted too short for a UPD packet. Such possibility of a potential malformed packet is not controlled within the implementation of the selected technique; in other words, potential defects in modified packets are possible and do not depend so much on the selected technique as on the specific code and chosen mapping. It is important to emphasize that the techniques developed in the early versions of CCgen are progressively refined depending on the conducted tests and the methods used to detect them. Not in vain, this is one of the objectives of CCgen modular design.

Beyond that, *Suricata* triggered two additional alarms (out-of-window and scan/brute-force), but both in the original and in the manipulated PCAPs likewise, meaning that such suspicious behaviors were already there and not

Table 2: *Suricata* rulesets used for the evaluation.

name	Summary	Vendor
et/open	Emerging threats open ruleset	Proofpoint
et/pro	Emerging threats pro ruleset	Proofpoint
Oisf/trafficid	*Suricata* traffic ID ruleset	OISF
Ptresearch/attackdetection	Positive technologies attack detection team ruleset	Positive technologies
Scwx/enhanced	Secureworks suricata-enhanced ruleset	Secureworks
Scwx/malware	Secureworks suricata-malware ruleset	Secureworks
Scwx/security	Secureworks suricata-security ruleset	Secureworks
Sslbl/ssl-fp-blacklist	Abuse.ch SSL blacklist	Abuse.ch
Sslbl/ja3-fingerprints	Abuse.ch suricata JA3 fingerprint ruleset	Abuse.ch
Etnetera/aggressive	Etnetera aggressive IP blacklist	Etnetera a.s.
tgreen/hunting	Threat hunting rules	tgreen
Malsilo/win-malware	Commodity malware rules	Malsilo

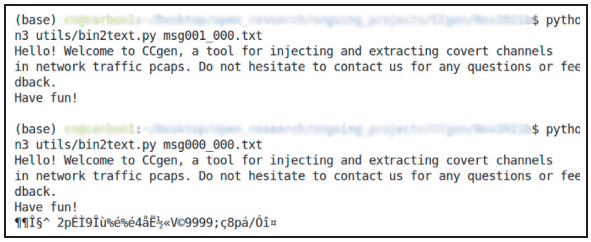

Figure 4: Example of residual noise in a retrieved covert channel.

induced by the performed covert channel injection. Finally, there is one alarm (invalid-ack) that appears in the original PCAP but disappears after the manipulation caused by the covert channel injection.

In general, the results in Table 1 are expected according to the field experience. For instance, in relation to expert knowledge, in a recent report comparing NIDS suitability for covert channel detection [39], Gunadi and Zander state that none of the three popular options analyzed (Snort, Bro, and *Suricata*) are natively prepared to face covert channels. However, the authors envision that Bro seems to be more suitable to develop future extensions with such purpose, since it has a more flexible architecture to implement anomaly detection.

The strategy of using anomaly detection to identify covert channels is explained by the fact that available techniques are multiple, very varied, and can take many different forms depending on the configuration parameters. Therefore, characterizing covert channels using strict rules or signatures is difficult for a significant bunch of possibilities. For instance, note that the difficulty in detecting the different types of covert channels implemented in the initial version of CCgen is variable. Some techniques are easy to detect if ad hoc detectors are incorporated, since they exploit resources and features in a very particular way (e.g., *ipflags*); others methods present statistical characteristics that can be modeled (e.g., *srcport_dev*) or can be detected by statistically modeling legitimate services and assessing deviations as alarm triggers (e.g., *timing_ber*). The statistical study of features is a good measure for implementing general-purpose detectors [14], although communications evolve rapidly and new ways of using protocols change. Statistical-based detection must still face problems related to high rates of false positives, since some services have cadences or show sequences that can be easily mistaken for covert channels, hence overloading security devices. For this reason, using machine learning to help modeling the baseline behavior of communications and define a benchmark normality is a promising approach to be implemented in future detectors.

6. Conclusions

In this paper, we have introduced the CCgen framework, an open-source set of tools for injecting a wide variety of covert channels into both live communications and captured traces in PCAP format. In the first version, the tool is released with a set of 18 established covert channel techniques, and it is designed to allow the easy integration of novel techniques as independent script plugins.

Although they are categorized as cyber attacks, we have shown that the implemented covert channels go undetected by a modern cybersecurity NIDS like *Suricata* when using a default, standard set of rules. This simple proof is not intended to set any absolute statement, merely to highlight that the incorporation of covert channel detection in NIDS is an open issue of undeniable relevance. As observed by Gunadi and Zander, "there is no integrated approach to detect coverts channel yet. This is due to the vast possibilities of covert channels, which means that it is not feasible to address each covert channel separately" [39].

Note that assuming an intended usefulness and efficacy, NIDS is under severe pressure caused by technological development and the rapid evolution of network communications and thus cyber attacks. Future systems require both rule-based solutions and detection methods based on pattern extraction and anomaly identification. In this respect, the CCgen framework is perfect to create environments for stress testing algorithms and detection systems against covert channels.

Finally, CCgen is also an ideal tool for training advanced cybersecurity experts, since covert channels are actually covered by academic programs at technical universities and specialized centers [20].

Data Availability

The CCgen framework tool used to support the findings of this study has been deposited in the GITHUB repository (https://github.com/CN-TU/py_CCgen).

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the project MALware Communication in Critical Infrastructures (MALORI), funded by the Austrian Security Research Programme KIRAS of the Federal Ministry for Agriculture, Regions, and Tourism (BMLRT) under grant no. 873511. The authors acknowledge TU Wien Bibliothek for financial support through its Open Access Funding Programme.

References

[1] W. Mazurczyk and L. Caviglione, "Information hiding as a challenge for malware detection," *IEEE Security & Privacy*, vol. 13, no. 2, pp. 89–93, Mar 2015.

[2] K. Cabaj, L. Caviglione, W. Mazurczyk, and S. A. S. Wendzel, "The new threats of information hiding: the road ahead," *IT Professional*, vol. 20, no. 3, pp. 31–39, 2018.

[3] J. Tian, G. Xiong, Z. Li, and G. Gou, "A survey of key technologies for constructing network covert channel," *Security and Communication Networks*, vol. 2020, 2020.

[4] C. Alcaraz, G. Bernieri, F. Pascucci, J. Lopez, and R. Setola, "Covert channels-based stealth attacks in industry 4.0," *IEEE Systems Journal*, vol. 13, no. 4, pp. 3980–3988, 2019.

[5] A. Abdelwahab, W. Lucia, and A. Youssef, "Covert channels in cyber-physical systems," *2021 American Control Conference (ACC)*, vol. 5, no. 4, pp. 1273–1278, 2021.

[6] C. Alcaraz, J. Lopez, and S. Wolthusen, "Ocpp protocol: security threats and challenges," *IEEE Transactions on Smart Grid*, vol. 8, no. 5, pp. 2452–2459, 2017.

[7] P. Eder-Neuhauser, T. Zseby, J. Fabini, and G. Vormayr, "Cyber attack models for smart grid environments," *Sustainable Energy, Grids and Networks*, vol. 12, no. 10–29, pp. 10–29, 2017.

[8] D. Barradas and N. Santos, "Towards a scalable censorship-resistant overlay network based on webrtc covert channels," in *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good, DICG'20*, pp. 37–42, New York, NY, USA, 2020.

[9] W. Mazurczyk and Z. Kotulski, "New VoIP traffic security scheme with digital watermarking," in *Computer Safety, Reliability, and Security*, J. Górski, Ed., pp. 170–181, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[10] E. Jones, O. Le Moigne, and J.-M. Robert, "IP traceback solutions based on Time to Tive covert channel," in *Proceedings of the 2004 12th IEEE International Conference on Networks (ICON 2004) (IEEE Cat. No.04EX955)*, vol. 2, pp. 451–457, Singapore, November 2004.

[11] X. Ying, G. Bernieri, M. Conti, and R. Poovendran, "TACAN: transmitter authentication through covert channels in controller area networks," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems, ICCPS '19*, pp. 23–34, New York, NY, USA, May 2019.

[12] Z. Cai and Y. Zhang, "Entropy based taxonomy of network convert channels," in *Proceedings of the Power Electronics and Intelligent Transportation System (PEITS), 2009 2nd International Conference on*, vol. 1, pp. 451–455, Shenzhen, Dec 2009.

[13] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 3, pp. 44–57, 2007.

[14] F. Iglesias, R. Annessi, and T. Zseby, "DAT detectors: uncovering TCP/IP covert channels by descriptive analytics," *Security and Communication Networks*, vol. 9, no. 15, 3029 pages, 2016.

[15] S. Wendzel, S. Zander, B. Fechner, and C. Herdin, "Pattern-based survey and categorization of network covert channel techniques," *ACM Computing Surveys*, vol. 47, no. 3, pp. 1–26, apr 2015.

[16] L. Caviglione, "Trends and challenges in network covert channels countermeasures," *Applied Sciences*, vol. 11, no. 4, p. 1641, 2021.

[17] F. Rezaei, M. Hempel, and H. Sharif, "A novel automated framework for modeling and evaluating covert channel algorithms," *Security and Communication Networks*, vol. 8, no. 4, pp. 649–660, 05 2015.

[18] S. Zander and G. Armitage, *"CCHEF - covert Channels Evaluation Framework Design and Implementation"*, Centre for Advanced Internet Architectures, Swinburne University of Technology, 2008.

[19] M. Zuppelli and L. Caviglione, "PcapStego: a tool for generating traffic traces for experimenting with network covert channels," in *Proceedings of the 16th International Conference on Availability, Reliability and Security, ARES 2021*, New York, NY, USA, 2021.

[20] T. Zseby, F. Iglesias Vazquez, V. Bernhardt, D. Frkat, and R. Annessi, "A network steganography lab on detecting TCP/IP covert channels," *IEEE Transactions on Education*, vol. 59, no. 3, pp. 224–232, November 2016.

[21] K. Ahsan and D. Kundur, "Practical data hiding in TCP/IP," in *Proc. ACM Workshop Multimedia and Security*, 2002.

[22] C. H. Rowland, "Covert channels in the TCP/IP protocol suite," *First Monday*, vol. 2, no. 5, 1997.

[23] Bo Xu, J.-zhen Wang, and De-yun Peng, "Practical protocol steganography: hiding data in IP header," in *Modelling Simulation, 2007. AMS '07*, pp. 584–588, First Asia International Conference on, 2007.

[24] C. G. Girling, "Covert channels in LAN's," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 292–296, 1987.

[25] S. Wendzel and S. Zander, "Detecting protocol switching covert channels," in *Proceedings of the Local Computer Networks (LCN), 2012 IEEE 37th Conference on*, pp. 280–283, Clearwater Beach, FL, USA, October 2012.

[26] J. Postel, *RFC0791: Internet Protocol*, ", 1981.

[27] F. Baker, D. L. Black, K. Nichols, L Steven, and Blake, "Definition of the differentiated services field (DS field) in the IPv4 and IPv6 headers," in *Proceedings of the RFC 2474*, USA, December 1998.

[28] S. Floyd, Dr. K. K. Ramakrishnan, and D. L. Black, "The addition of Explicit congestion notification (ECN) to IP," in *Proceedings of the RFC 3168*, USA, September 2001.

[29] J. Gimbi, D. Johnson, P. Lutz, and Bo Yuan, "A covert channel over transport layer source ports," in *Proceedings of the 2012 International Conference on Security and Management*, Las Vegas, NV, USA, July 2012.

[30] H. Qu, P. Su, and D. Feng, "A typical noisy covert channel in the IP protocol," in *Proceedings of the Security Technology, 2004. 38th Annual 2004 International Carnahan Conference on*, pp. 189–192, Albuquerque, NM, USA, October 2004.

[31] N. B. Lucena, G. Lewandowski, and S. J. Chapin, "Covert channels in IPv6," in *Privacy Enhancing Technologies*, D. George and D. Martin, Eds., pp. 147–166, Springer, Berlin Heidelberg, 2006.

[32] S. Zerafshan Goher, "Barkha javed and nazar abbas saqib. "covert Channel detection: a survey based analysis"" in *High Capacity Optical Networks and Emerging/Enabling Technologies*, 2012.

[33] G. Fisk, M. Fisk, C. Papadopoulos, and J. Neil, "Eliminating steganography in internet traffic with active wardens," in *Proceedings of the Revised Papers from the 5th International Workshop on Information Hiding*, pp. 18–35, IH '02, USA, February 2003.

[34] S. Zander, G. Armitage, and P. Branch, "An empirical evaluation of IP Time to Live covert channels," in *Proceedings of the Networks, 2007. ICON 2007. 15th IEEE International Conference on*, pp. 42–47, Adelaide, SA, Australia, November 2007.

[35] F. I. Vázquez, R. Annessi, and T. Zseby, "Analytic study of features for the detection of covert timing channels in networktraffic," *Journal of Cyber Security and Mobility*, vol. 6, no. 3, pp. 245–270, 2017.

[36] V. Berk, A. Giani, and C. George, "Detection of covert channel encoding in network packet delays," *Technical Report TR2005-536*, Dartmouth College, 2005.

[37] G. Wade and Li Yang, "Network covert channels on the android platform," in *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '11, New York, NY, USA, 2011.

[38] G. Shah, A. Molina, and B. Matt, "Keyboards and covert channels," in *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, pp. 59–75, USENIX-SS'06, Vancouver, B.C., Canada, July 2006.

[39] H. Gunadi and S. Zander, *"Comparison of IDS Suitability for covert Channels Detection"*, Technical Report, 2017.