Covert Channels in Transport Layer Security: Performance and Security Assessment

Corinna Heinz[1], Marco Zuppelli[2*], and Luca Caviglione[2]
[1]FernUniversität in Hagen, Hagen, Germany
ch@sysv.de
[2]National Research Council of Italy, Genova, Italy
{marco.zuppelli, luca.caviglione}@ge.imati.cnr.it

Received: March 29, 2021; Accepted: September 2, 2021; Published: December 31, 2021

Abstract

The ability of creating covert channels within network traffic is now largely exploited by malware to elude detection, remain unnoticed while exfiltrating data or coordinating an attack. As a consequence, designing a network covert channel or anticipating its exploitation are prime goals to fully understand the security of modern network and computing environments. Due to its ubiquitous availability and large diffusion, Transport Layer Security (TLS) traffic may quickly become the target of malware or attackers wanting to establish a hidden communication path through the Internet. Therefore, this paper investigates mechanisms that can be used to create covert channels within TLS conversations. Experimental results also demonstrated the inability of de-facto standard network security tools to spot TLS-based covert channels out of the box.

Keywords: covert channels, transport layer security, network intrusion detection.

1 Introduction

Information hiding techniques are becoming largely applied to network traffic for a wide-array of tasks, for instance, to watermark flows and trace them across the Internet or to implement traffic engineering policies [1]. Despite such licit usages, information hiding is still primarily deployed by cybercriminals, especially to endow malware with the ability of act unnoticed while performing exfiltration of sensitive data, orchestrating nodes of a botnet, retrieving additional payloads or exchanging commands to activate a remote backdoor [2, 3]. To emphasize the adoption of steganographic or hiding techniques to support parts of the cyber kill chain, such a new-wave of threats has been named *stegomalware* [4]. Attacks observed "in the wild" exploit such methods for a vast range of applications: possible examples are the obfuscation of hazardous routines, steganographic injection of malicious code within innocuous digital media, and creation of abusive inter processes communication services to bypass sandboxes or execution enclaves. However, the most popular and effective utilization of information hiding concerns the creation of a *network covert channel* that is a hidden communication path laying within an overt traffic flow acting as the carrier.

Originally introduced by Lampson in 1973, covert channels are *"[channels] not intended for information transfer at all*" [5]. The recent literature proposes several methods to create hidden communication paths in a variety of scenarios, such as cloud and fog environments, virtualized or containerized entities, and multi-core/CPU frameworks, just to mention the most popular [6].

Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications (JoWUA), 12(4):22-36, Dec. 2021
DOI:*10.22667/JOWUA.2021.12.31.022*

*Corresponding author: Institute for Applied Mathematics and Information Technologies, National Research Council of Italy, Via de Marini 6, Genova, Italy, I-16149, Web: `http://imati.cnr.it`

For the case of network covert channels, different portions of the protocol stack or behaviors of the traffic can be used as the carrier where to inject the secret information. To this aim, data can be hidden in unused header bits, as well as by encoding secrets in a pre-defined order of attributes or by intentionally generating transmission errors [7]. Despite the used mechanism, an effective covert channel requires to use a popular carrier to not appear as an anomaly. The resulting channels should have an adequate capacity of containing secrets and the ability to withstand to impairments and transmission errors. In this vein, the Transport Layer Security (TLS) protocol surely fulfills such requirements. First, it is widely used to provide encryption and enhanced security to a variety of applications and protocols, e.g., the Internet Message Access Protocol [8]. Second, it is inherently reliable and demonstrated its ability of working correctly in several settings, including wireless and mobile scenarios.

Concerning the creation of network covert channels for data exchange and to support attacks, the literature showcases a wide range of works. For instance, in [9] the use of the Dynamic Host Configuration Protocol to establish hidden communication paths is presented, whereas a technique to embed secrets in the Stream Control Transmission Protocol is discussed in [10]. Moreover, in the recent work [11], authors demonstrate several methods to exploit features of the Message Queuing Telemetry Transport, which is gaining momentum mainly due to its adoption in many Internet of Things deployments. Another possible approach is presented in [12], where authors propose different techniques for hiding data within a VoIP traffic flow, namely in the header, by reordering packets or a combination of the two. Other possible approaches are partially protocol-agnostic and are based on some variation of the transmission rate or the timing statistics characterizing the stream of packets [13]. In [14] authors review many techniques, with emphasis on hiding mechanisms that can take advantage of features of modern smartphones such as accelerometers. Regarding the utilization of TLS conversations, at the best of our knowledge, only the theoretical work in [15] proposes the use of TLS headers for sending secret information. In more detail, authors discuss how to replace the random data used for the handshake procedure to negotiate encryption parameters with an arbitrary content.

In this perspective, the popularity of the TLS makes it an interesting target for information-hiding capable applications and steganographic attacks. Therefore, in this work we investigate different mechanisms that can be used to inject data within a TLS conversation with the aim of anticipating flaws and provide ideas to engineer suitable countermeasures. We also investigate if de-facto standard network security tools are able to spot the presence of covert communications within TLS traffic. Thus, the main goal of this work is to narrow the array of possibilities used by malware developers for remaining unnoticed when exploiting covert channels. Another objective of our investigation concerns the evaluation of the most popular security tools mainly to highlight their inadequacy when used to counteract information-hiding capable threats [2], [16], and [3]. Summing up, the contributions of this paper are:

- the review and design of seven TLS-capable covert channels and a thorough performance assessment of three techniques;
- the evaluation of three popular network intrusion detection systems (i.e., Bro/Zeek, Snort and Suricata) when dealing with covert channels nested in TLS traffic;
- the discussion of possible ideas to engineer effective countermeasures.

This paper is an extended version of the work [17] and has the following differences/improvements: the discussion on TLS-capable covert channels has been enhanced and the performance evaluation has further refined with additional results. Moreover, the assessment of three network security tools has been added by considering TLS covert channels when used to exfiltrate stolen information and to transfer real-world attacks, such as file-less malware and malicious payloads.

The rest of the paper is structured as follows. Section 2 provides some background information on TLS and introduces the attack model, while Section 3 showcases data hiding approaches that can be used

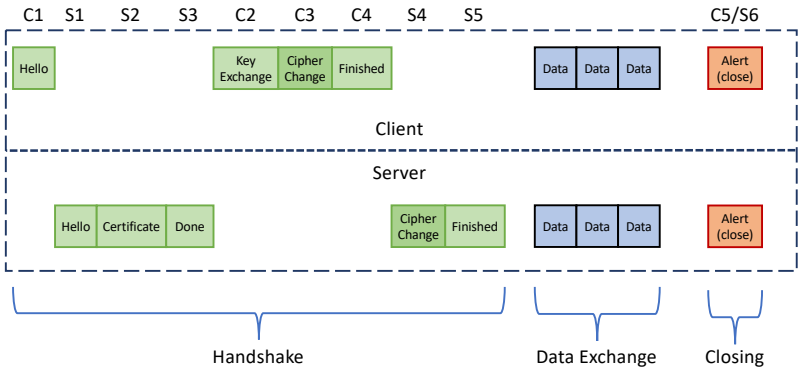

Figure 1: General structure of a TLS 1.2 conversation. After `ChangeChiperSpec`, all the following messages are encrypted.

for building covert channels. Section 4 discusses the performance of three TLS-based covert channels in realistic settings, whereas Section 5 investigates if standard tools can be considered suitable for detection and proposes some countermeasure. Lastly, 6 concludes the paper and outlines future research directions.

2 Background

This section firstly describes the main features of the TLS protocol with emphasis on those that can be exploited for information hiding purposes. Then it introduces the considered attack model and the use of network covert channels.

2.1 Transport Layer Security

The TLS protocol has been introduced as an improvement of the (now deprecated) Secure Socket Layer. In this paper, we refer to TLS 1.2, defined in RFC 5246 [18], which is widely deployed and can be considered more susceptible to attacks if compared to the new 1.3 release. Despite the used version, the ultimate goal of TLS is to provide confidentiality and integrity of the transmitted data, while offering identification and authentication of communicating endpoints.

Even if TLS has been engineered to be independent from a specific cryptographic algorithm, it needs at least one asymmetric cipher for the key exchange and authentication, a symmetric cipher for the actual data encryption, and a Message Authentication Code (MAC) to ensure message integrity. Owing to this design choice, TLS can enjoy the pros of both encryption methods, i.e., the security of the asymmetrical procedure when exchanging keys and the speed of the symmetrical procedure during the data transmission phase [19]. For the identification and authentication of the communicating parties, it relies upon X.509 certificates and a block of random data (called premaster secret) is used to generate the keys for the endpoints. Such a secret is encrypted with an asymmetric cipher, e.g., the RSA, and it represents a proof of the possession of the private key. Moreover, the premaster secret allows to identify the server. Both endpoints derive a set of symmetric session keys, which are used with the symmetric cipher, e.g., AES256 Cypher Block Chaining (CBC), to encrypt the payload. To verify the integrity of the message, a keyed-Hash MAC (HMAC) procedure is adopted to prevent modification of the TLS conversation via a Man-in-the-Middle (MitM) attack.

The TLS architecture is composed of two layers: the TLS Record Protocol responsible of encapsulating data from upper layers and the TLS Handshake Protocol enabling endpoints to negotiate parameters and authenticating themselves. The various TLS messages are grouped in records of a size up to $32,767$ bytes, and each record is encrypted and authenticated with the HMAC in an independent manner. The `Type` field (one per record) points at the higher-level protocol of the TLS message: possible choices are handshake, cipher spec, alert or data message.

As an example, Figure 1 depicts a toy client-server TLS conversation. With **C** and **S** we denote messages sent by the client and the server, respectively, which are followed by a number identifying their temporal sequence. As a preliminary step, the client sends a `ClientHello` message (**C1**) with the list of the supported cipher suites, ordered by preference. The message also contains a random byte sequence used for key generation, the supported compression methods and optional extensions. The server responds with a `ServerHello` message (**S1**) also containing a random block of data, the cipher and compression methods matching those requested by the client. The server then sends a certificate (**S2**): if the authentication of the client is required, it will also send a client certificate request (omitted in the figure for the sake of clarity). A `ServerHello` done message (**S3**) concludes the responses. The client then sends the asymmetric encrypted premaster secret (**C2**), which is necessary to generate the session keys at the server side. The `ChangeCipherSpec` message (**C3**) announces that all subsequent records will be secured and authenticated with the parameters agreed during the negotiation phase. The handshake phase is completed via a `Client Finished` message (**C4**) containing a hash value computed over all the messages exchanged so far. This guarantees that the server can detect unauthorized tampering of the unencrypted messages exchanged during the handshake. The messages (**S4**) and (**S5**) have the same purpose as (**C3**) and (**C4**). Upon finishing the handshake phase, messages containing information can be exchanged (the resulting flow has been indicated in the figure as **Data**). To close the connection, a proper sequence of alert/close messages is exchanged (see, messages (**C5**) and (**S6**) in the figure). In case of missing or out of sequence messages, an error is raised: in fact, the TLS considers them as a MitM attempt or an indication of a truncation attack.

2.2 Attack Model and Network Covert Channels

In this work, we consider an attacker trying to exfiltrate data towards a remote host or a malware wanting to download an additional payload. Owing to the enforced security policies or the presence of tools like a firewall or an Intrusion Detection System (IDS), communications are blocked or can be easily detected, hence voiding the attempt. Therefore, we consider a threat trying to bypass blockages by communicating via a network covert channel exploiting a licit TLS conversation. In the following, we denote the two peers wanting to exchange information in a stealthy manner as the Secret Sender (SS) and the Secret Receiver (SR), respectively. The carrier containing the secret is some entity/behavior of the network traffic implementing the overt communication between an Overt Sender (OS) and an Overt Receiver (OR). Figure 2 depicts the reference scenarios and the attack models. In the first case, the SS and the SR are placed in some intermediary network nodes to act in a MitM flavor. This scenario has a limited applicability since a manipulation of the encrypted stream from an intermediate node will be easily spotted and cause a sudden abort of the TLS communication. Yet, a MitM attack can still happen when hiding data during the very beginning stages of the handshake phase, since the TLS stream is not encrypted and the OS and the OR are still negotiating cyphers and keys. Moreover, a MitM threat could be feasible if the attacker can alter/substitute the system-wide SSL libraries, thus having a prior knowledge on the used certificates/keys. The second case is more common and considers the SS co-located with the OS (also the SR could run in the same host of the OR). This scenario, for instance, represents a threat running in the host of the victim contacting a remote server instrumented by the attacker. By hiding data within a legitimate TLS conversation, the secret endpoints implement a TLS

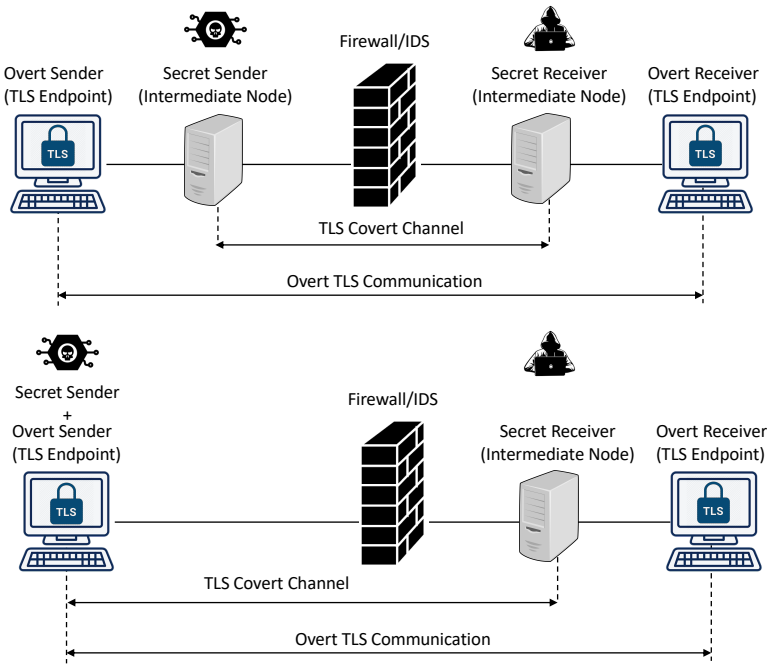

Figure 2: Reference scenarios and attack models for network covert channels.

covert channel, which can be used to exchange information and complete the attack.

In general, a covert channel is described via different but interdependent metrics [14]. As a paradigmatic example, the amount of secret data that can be sent per time unit, defined as *steganographic bandwidth* cannot be increased without influencing the *undetectability* of the channel. In fact, the more the carrier is manipulated to contain secrets, the higher the chance of artifacts that can reveal the presence of hidden information. Similarly, the *robustness* of the channel, i.e., the number of errors and alterations the secret can withstand, may require to deploy an error correction algorithm causing an erosion of the available space within the carrier or account for computational overheads causing lags in the device of the victim [20].

3 Design of Covert Channels Targeting TLS

This section showcases seven different covert channels, each one exploiting a well-defined characteristic of the TLS protocol. Emphasis will be put to identify the pros and cons of each technique.

3.1 Record-Length Encoding

Since overt TLS messages can be split into an arbitrary number of records, a secret information can be hidden by modulating their size and number. To this aim, the SS encodes the secret message according to a scheme shared in advance with the SR and generates the associated record lengths. The overt messages are then artificially split into records of the needed lengths and forwarded to the TLS peer. This modification has no influence on the TLS payload data and the overt TLS endpoints are still able to correctly decrypt the information.

The steganographic bandwidth of this channel is influenced by the block cipher used by the TLS protocol. In fact, when selecting a common block cipher with blocks of 16 bytes, the two minimum record lengths are 64 bytes (for example, denoting a 1 bit) and 80 bytes (for a 0 bit) plus 5 bytes for

the record header. This leads to 1 byte of secret data for every 616 bytes of TLS traffic, on the average. The limit of the approach is that the secret endpoint should be able to eavesdrop the whole TLS stream and packet losses will make the secret impossible to decode. Moreover, having too many small TLS records may cause artifacts in the traffic that can reveal the channel. Recalling that the maximum size of a TLS record is $18,432$ bytes, too many small protocol data units may cause a wastage of resources due to overheads, thus tiny packets are usually avoided in optimized applications.

3.2 Random Values and Initialization Vector

According to the protocol specification, `ClientHello` messages can also contain random values. In this case, the `Random` field can be filled with 32 bytes of arbitrary data. A similar behavior characterizes the `Session ID`, which is used to resume a TLS session. If the client does not exploit such a feature, random data will be discarded and then it can be replaced with arbitrary information. Another field that can be targeted for data hiding purposes is the `Initialization Vector` of each encrypted TLS data record when a CBC cipher is used (e.g., the AES256-CBC).

The aforementioned quantities can be only modified by a TLS peer, otherwise the manipulation of the TLS stream will be detected and the connection will be shut down. Still, both the initial `ClientHello` message and the various `Initialization Vectors` are sent through the network unencrypted, so an eavesdropper can extract the hidden information. As an example, the SS can hide the data in the `Initialization Vector` within the data records of the TLS stream. The SR intercepts the handshake messages and extracts the secret information.

As regards the performance of injection methods targeting random data and the `Initialization Vector`, the achievable bandwidth is high, since several bytes per record can be used. When using the `Initialization Vector` with AES256-CBC, 16 bytes per record are available for the covert channel. The injection of data within the `Initialization Vector` is poorly detectable, especially if the covert sender scrambles the information to resemble a randomly-generated pattern.

3.3 Record ContentType

The Record Layer uses the `ContentType` to specify the content of a record. An attacker wanting to create a covert channel within the TLS stream can use non-critical alert messages by exploiting unassigned alert identifiers (see, [18] for a list of TLS Alerts maintained by IANA) or an empty record. Then, by interleaving records of various types, the secret information can be encoded. As an example, a simple encoding scheme could consider the alternation of TLS data messages and TLS alert messages. For instance, a data message represents a `1` bit, while an alert message a `0` bit. The SR inspects the sequence of records with type TLS alert and TLS data and then decode the secret information according to the agreed scheme.

The achievable steganographic bandwidth depends on the size of the minimal (encrypted) record size. For instance, with a cipher using blocks of 16 bytes, the minimal record size is equal to 69 bytes. Therefore, a single byte of covert information is transmitted in 552 bytes of the TLS protocol data, which is about 0.18% of the capacity of the total network stream.

Concerning the robustness of the channel, the SR has to eavesdrop all packets composing the TLS stream, as even a single loss could cause losing the sync and disrupt the covert communication. This makes the transmission of long sequences difficult, especially without triggering some heuristics deployed within an IDS or a traffic analyzer. Moreover, implementing this method in realistic scenarios could require to face some limitations. Specifically, many TLS implementations (such as the widely-deployed OpenSSL) cause an endpoint to drop the connection after 5 consecutive "warning" messages

or 32 empty records. Thus, without suitable workarounds (e.g., the adoption of an encoding scheme preventing suspicious bursts of records) the effective dangerousness of the channel is partially downsized.

3.4 ClientHello Extension

The TLS specification requires that a server ignores unsupported extensions appended to a `ClientHello` message. Thus, the SS can encode data by creating its own extension enclosing the secret and appending it to the `ClientHello` message. To avoid disrupting or altering the overt traffic, the SS must not inject data in already used or reserved extensions. Extensions can have a maximum size of $65,535$ bytes, but each one requires 2 bytes for identification purposes and 2 bytes for the length. Since extensions are exchanged once per connection, the maximum data that can be sent through this covert channel is $65,531$ bytes, which is further reduced if the OS and the OR use the field [21]. To decrypt the information, the SR only needs to eavesdrop the first message, i.e., the `ClientHello`. Due to its simplicity, this approach does not offer a suitable stealthiness. In fact, unknown extensions can be easily spotted by an IDS, thus yielding the resulting covert channel highly detectable.

3.5 Name and Maximum Fragment Encoding

A more advanced use of extensions can profit from the manipulation of the Server Name Indication (SNI) defined in RFC 6066 [21]. Put briefly, the SNI is a standard field commonly observable in many TLS traffic traces, hence its presence is not perceived as an anomaly (e.g., it is usually present when requesting a website through the HTTPS protocol). To create a covert channel, the SS could inject and encode a secret within the SNI. A possible scheme could alter individual letters of the hostname to transport few bits of information. For instance, the sender can exploit capitalization of letters to inject secrets, e.g., it can indicate a `1` with `www.example.com` or a `0` with `www.Example.com`. Another possible idea is to map `1` or `0` values in capital/non-capital letters. In this case, the string `1010011` can be transmitted via an SNI equal to `www.ExAmpLE.com`. Accordingly, to decode the information only the interception of the `ClientHello` message is required. The steganographic bandwidth of the channel varies with the length of the string used in the SNI and the adopted encoding but it is usually very limited. As regards the stealthiness of the channel, it should be considered as moderate: despite the SNI alteration could not lead to misbehaviors in the TLS conversation, it can be easily spotted by checking the name of the server.

Another exploitable extension is the Maximum Fragment length, which is used by the client to request that the server will not send records exceeding a specified maximum size [21]. In more detail, the values 1, 2, 3 or 4 denote a maximum fragment length of 512, $1,024$, $2,048$, and $4,096$ bytes, respectively. Then, values can be manipulated by the SS to indicate a secret bit. The Maximum Fragment length extension is seldom used: even if a manipulation would not cause the disruption of the channel, its presence can represent an anomaly or be used as an indicator to spot the covert communication attempt. Moreover, "artificial" fragmentation may cause bandwidth overheads or slightly increased CPU usages, which could lead to anomalous battery drains or "execution signatures" that can be exploited by third-party security tools [22].

3.6 Alteration of Ciphers and Compressions

When using TLS to establish a secure connection, the very first step requires to negotiate the communication parameters including the used cryptographic algorithms. The lists of cipher suites and compression algorithms can be altered to encode information (e.g., the order of the algorithms can be arranged as to encode a secret). Each cipher is specified by a 16-bit integer, whereas the compression methods use a 8-bit integer. In this case, the SS can encode the secret information according to the agreed scheme and it

generates the `ClientHello` message with the appropriate list of ciphers and compressions. Then, such a list is sent to the OR. The SR should intercept the `ClientHello` message and extract the `Ciphers` and `Compression` fields to decode the data. The vectors containing the cipher suites and the compression algorithms can be altered by TLS peers without violating the protocol and the transmission of data can only take place once per TLS connection.

The amount of covert data that can be transmitted is influenced by two factors. First, the method used to encode the secret information, e.g., using 20 cipher suits to encode the number 20 is less efficient than using a scheme based on permutation of values. Second, the number of available cipher suites and compression algorithms depends on the used TLS library. This method should be considered suitable for transferring very small amounts of secret data, since using large and uncommon lists of cipher suites and compression algorithms can reveal the presence of the covert communication.

3.7 PDU Corruption / Loss

To transport data, TLS depends on the TCP, which provides reliable end-to-end connections. Therefore, according to [18], a peer detecting an error or a missing PDU in the TLS stream should terminate the connection, as this will cause the MAC to fail and potentially indicating a MitM or replay/reordering attack. Such a behavior can be exploited to covertly transmit a very limited amount of data. For instance, by causing the abortion of a specific connection, it would be possible to indicate a single bit, e.g., the signal to activate a backdoor. This method can be easily detected as the overt TLS stream will be shutdown resulting in a macroscopic anomaly.

4 Performance Evaluation of TLS Channels

This section discusses the performance of covert channels built with the Record-Length Encoding (Section 3.1), Initialization Vector (Section 3.2), and Record ContentType (Section 3.3) methods, denoted in the following as RLE, IV and RCT, respectively. Such methods have been selected as they are characterized by a nice tradeoff between steganographic bandwidth and undetectability.

To have a controlled environment, we performed trials by using hosts (3.3 GHz quadcore CPU with 32 GB of RAM) running Debian 10, connected through a 100 Mbit/s Ethernet switch. The considered scenario is the one depicted in Figure 2, with covert channels implemented in end nodes. Investigating channels targeting TLS traffic in a MitM manner is part of our ongoing research. The overt traffic has been generated via an instrumented browser retrieving static contents from a web server running Apache 2.4. The various TLS-capable covert channels have been implemented in C (see, `https://github.com/CoriHe/TLSCC` for the source code). Numerical results have been collected by the sender part of the covert channel implementation. Concerning the amount of secret data to be exchanged, we considered three use-cases, i.e., we tested the channels when transferring 10 kbytes, 100 kbytes and 1 Mbytes of data. Such values have been introduced to consider the exfiltration of simple information (e.g., personal details or an entire address book) or a complete digital document (e.g., an industrial report or a highly-compressed image). To evaluate the robustness of the covert channels, we also performed trials by adding impairments in the overt network traffic. To this aim, we used `iptables` to introduce artificial packet losses, i.e., 0%, 1%, 2%, 3% and 5%, and `tc` to add 50 ms, 200 ms, and 500 ms of delay.

Lastly, to have a proper statistical relevance, for each experiment, 10 repeated trials have been made and average values will be presented in the following.

Covert Channel	Secret Data [kbyte]	Overt Data Sent [kbyte]		η [%]
		AVG	STDEV	
IV	10	47	0	21.22
IV	100	460	0	21.69
IV	1,000	4,600	0	21.73
RCT	10	6,493	4,471	0.15
RCT	100	64,924	10,170	0.15
RCT	1,000	649,228	33,549	0.15
RLE	10	6,161	2,807	0.16
RLE	100	61,600	6,551	0.16
RLE	1,000	615,998	22,693	0.16

Table 1: Exchanged data volumes.

4.1 Numerical Results

As a first step for characterizing the behavior of TLS-based covert channels, we investigated performances in terms of produced data volumes. To better quantify performances, we introduced a condensed indicator defined as $\eta = \frac{BW_{steg}}{BW_{overt}}$, where BW_{steg} and BW_{overt} are the bandwidths of the covert and the overt traffic, respectively. In general, a SS acting in a MitM fashion cannot influence the rate of the flow produced by the OS, but can only inject secret information by manipulating the overt flow. In this vein, η gives an idea of the ability of an embedding method of "converting" part of the capacity of the TLS conversation into exploitable steganographic bandwidth.

Table 1 showcases results averaged over the entire dataset (including trials with impairments). Specifically, for each hiding method, the table reports the volume of overt traffic needed to transfer a well-defined amount of secret information. As shown, the covert channel using the IV technique requires less overt data for transmitting a secret. This is due to the IV method exploiting a TLS connection using AES in CBC mode and a minimal record size of 64 bytes. Hence, for each record, the channel can use 16 bytes for injecting the secret. In contrast, for the case of RCT, the steganographic capacity is reduced to 1 byte of covert data per 552 bytes of TLS traffic, while for RLE it equals to 1 byte of covert data per 616 bytes of TLS traffic. The different performance levels obtained by the proposed methods can be also assessed in terms of the η indicator. In this case, the IV achieves significantly higher results ($\sim 21\%$) when compared to RCT and RLE, which can only take advantage of the throughput of the overt flow in a limited manner ($0.15 - 0.16\%$).

Concerning the performance of the channel in term of bandwidth and connection time when in the presence of different impairments, Table 2 presents numerical results for the SS and the SR exchanging 1 Mbyte of secret data. Owing to the use of TCP offering a reliable transport service, neither artificial packet loss nor the additional delays have prevented the correct delivery and decoding of the secret data. However, network impairments can heavily influence the overt transmissions and thus the performances of the covert channel. Specifically, the total duration for covertly transmitting the secret is significantly influenced by the high network latency, thus the nested covert channel experiences a performance loss as well. For example, when no additional delay is introduced, the time needed to transfer the secret is equal to $\sim$0.5 s, $\sim$57 s, and $\sim$54 s, for the IV, RCT, and RLE methods, respectively. However, if a 50 ms delay is added, then the connection time increases to $\sim$6 s, $\sim$471 s, and $\sim$470 s, respectively. In the case of additional packet losses, the presence of the HTTP/TCP protocol hierarchy seems to mitigate their impact (e.g., by "spreading" the burst of errors over multiple connections [23]). In fact, in our trials, packet losses affected the BW_{steg} mainly for the IV method, which reduces of $7 - 10\%$, whereas the other channels appeared to be insensitive. We point out that, aggressive impairments can disrupt the HTTPS

Method IV - 1 Mbyte Secret Data

	Connection Time [s]		BW_{overt} [Mbit/s]		BW_{steg} [Mbit/s]	
Packet Loss [%]	AVG	STDEV	AVG	STDEV	AVG	STDEV
0	0.49	0.00	74.67	1.42	16.23	0.31
1	0.50	0.01	74.00	1.56	16.08	0.34
2	0.53	0.13	71.52	10.78	15.55	2.34
3	0.58	0.15	66.69	13.20	14.50	2.87
5	0.56	0.10	67.72	10.81	14.72	2.35
Delay [ms]	AVG	STDEV	AVG	STDEV	AVG	STDEV
50	6.12	1.89	6.64	2.35	1.44	0.51
200	19.43	3.79	1.97	0.44	0.43	0.10
500	54.67	11.41	0.70	0.16	0.15	0.03

Method RCT - 1 Mbyte Secret Data

	Connection Time [s]		BW_{overt} [Mbit/s]		BW_{steg} [Mbit/s]	
Packet Loss [%]	AVG	STDEV	AVG	STDEV	AVG	STDEV
0	56.83	0.44	91.40	0.69	0.14	0.00
1	56.77	0.34	91.49	0.39	0.14	0.00
2	56.88	0.29	91.32	0.46	0.14	0.00
3	56.98	0.48	91.16	0.77	0.14	0.00
5	56.88	0.29	91.32	0.46	0.14	0.00
Delay [ms]	AVG	STDEV	AVG	STDEV	AVG	STDEV
50	471.19	11.07	11,030.60	262.27	0.17	0.00
200	2,377.37	646.49	2,317.94	553.06	0.03	0.00
500	7,446.33	452.41	699.76	0.16	0.01	0.00

Method RLE - 1 Mbyte Secret Data

	Connection Time [s]		BW_{overt} [Mbit/s]		BW_{steg} [Mbit/s]	
Packet Loss [%]	AVG	STDEV	AVG	STDEV	AVG	STDEV
0	53.70	0.26	91.77	0.44	0.15	0.00
1	53.97	0.26	91.31	0.43	0.15	0.00
2	53.86	0.37	91.50	0.62	0.15	0.00
3	53.90	0.24	91.42	0.40	0.15	0.00
5	53.93	0.24	91.38	0.41	0.15	0.00
Delay [ms]	AVG	STDEV	AVG	STDEV	AVG	STDEV
50	468.62	13.64	10.52	0.31	0.02	0.00
200	2,202.00	474.57	2.32	0.41	0.00	0.00
500	8,030.39	2,105.47	0.65	0.14	0.00	0.00

Table 2: Performances of the TLS covert channels when dealing with different network impairments.

flow causing the server to drop the conversation. Therefore, even if the covert channel is robust enough, short-lived or abruptly-terminated overt flows should be considered when evaluating the amount of secret information to be sent.

5 Security Assessment

This section deals with the evaluation of the "level of insecurity" that can be caused by TLS-based covert channels when deployed in production-quality and realistic network scenarios. To this aim, we tested various network security tools when handling the channels introduced in this work.

To conduct trials, we considered the same setting depicted in Figure 2 and already presented in Section 4. To run the various firewalls (intended here with the generic acceptation of IDS, intrusion prevention system, and network security monitoring), we used a machine with Ubuntu 20.04 on an Intel Core i9-9900KF CPU@3.60GHz and 32 GB RAM. In our tests, we considered three different, de-facto standard security tools: Snort[1], Bro/Zeek[2] and Suricata[3]. In general, risks arising by network covert channels are largely underestimated both by security experts and users [2, 4, 3]. Thus, we decided to test selected tools in an "out of the box" fashion. Specifically, we considered Snort with two different set of rules (the "standard" set and the "enhanced" one available to registered users), while for the case of Bro/Zeek and Suricata we considered standard configurations.

Concerning the utilization patterns of TLS covert channels, we considered three different types of hidden data in order to model realistic attacks. The first case deals with a channel used to deliver a file-less malware of 179 bytes, i.e., a PowerShell script allowing to infect the victim with the CryptoWorm threat[4]. The second case considers an obfuscated payload of $2,046$ bytes retrieved via the covert channel, i.e., a collection of malicious instructions to run the Emotet malware[5]. The third case models the exfiltration of sensitive data via the covert channel from the host of the victim towards a remote server. Specifically, we model the stolen data with a string composed of $1,000$ randomly-generated bytes to consider the presence of some form of encryption or scrambling to improve the undetectability of the hidden content.

In all the trials, the used security tools were not able to detect the covert communication or raise a warning. Such a finding reinforces the general idea that many firewalls and IDSes cannot be considered a valid countermeasure against information-hiding-capable threats without massive configuration and tweaking efforts (see, e.g., [24] for the case of covert channels targeting the IPv6 protocol).

Since standard IDS and monitoring tools failed to spot the TLS covert channels, we investigated the behavior of the traffic in order to understand if the detection can be shifted at the network level by means of suitable traffic signatures. To this aim, we performed further experiments to collect traffic in three different use cases, specifically: clean condition (no hidden data, the OS and OR exchange HTTP POST messages of 315 bytes), exchange of a well-defined text string in order to model the activation of a backdoor or a command sent to a botnet, and when transferring a file of 10 kbytes.

Obtained results are summarized in Table 3. As shown, the table reports the collected statistics in terms of exchanged data, number of TLS records and the amount of bytes per record (BPR), as well as their detailed breakdown, denoted in the table as, Change Cipher Spec (CCS), alert, HandShake (HS) and data. As can be seen, when transmitting a simple, single message, the IV covert channel exhibits values similar to the one observed for the clean case. Volumes are instead inflated when in the presence of larger

[1] `https://www.snort.org`
[2] `https://zeek.org`
[3] `https://suricata-ids.org`
[4] `https://github.com/chenerlich/FCL/blob/master/Malwares/CryptoWorm.md`
[5] `https://github.com/chenerlich/FCL/blob/master/Malwares/Emotet.md`

Channel	N. Bytes	N. Records	CCS	Alert	HS	Data	BPR
Clean Overt Stream (HTTPS)							
-	1001	6	1	0	3	2	166
HTTPS traffic + hidden string							
IV	1296	7	1	1	3	2	185
RCT	15255	146	1	75	3	67	104
RLE	10520	127	1	1	3	122	82
HTTPS traffic + 10 kbytes of hidden data							
IV	47,122	674	1	1	3	669	101
RCT	6,406,646	94,267	1	50,158	3	44,105	101
RLE	6,169,024	80,135	1	1	3	80,130	76

Table 3: Traffic statistics characterizing different use cases for the TLS-based covert channels.

transfers. The RCT and RLE methods lead to greater impacts in terms of modifications experienced by the overt traffic. Both channels have similar values for the CCS, HS, and BPR. Yet, they are also characterized by major differences for the transferred bytes, number of records, and data. Thus, the development of a middlebox for detecting the presence of a covert channel within a TLS flow can take advantage of signatures and discrepancies in the observed traffic.

5.1 Development of Countermeasures

As hinted, signatures in the traffic can be be used to develop detection mechanisms. Unfortunately, they could be ineffective when in the presence of very short communications or could require a non-negligible modeling effort to have a "baseline" behavior to spot anomalous traffic patterns. In the perspective of engineering countermeasures against network covert channels targeting TLS traffic, the most effective approach can exploit a TLS froward proxy. Specifically, the proxy accepts requests from TLS clients, terminates them and establishes new connections with the desired servers/endpoints. Thus, only the application data is retained from the original connection and a new, sanitized network conversation is created. Unfortunately, this approach disrupts the original communication (especially, the end-to-end semantic characterizing the higher layers of the protocol stack) and the authenticity of the peer cannot be verified anymore without taking additional measures on the client side. Such an approach could lack scalability, especially if the proxy has to protect many users producing traffic at high rates.

Another possible solution is to engineer an advanced set of rules to allow firewalls or IDSes to reveal anomalies by inspecting the behavior of TLS conversations. According to our investigations, possible metrics/indicators that can spot hidden information within a TLS flow are the presence of invalid or rarely used parameters or extensions, as well as metrics for describing TLS record types, sizes and frequency.

As a concluding remark, threat-independent signatures (e.g., abnormal energy drains [22]) or low-level measurements [25], [26] should be taken into account since they are promising mechanisms to detect TLS-based covert channels.

6 Conclusions and Future Work

In this paper, we have presented seven different covert channels targeting TLS conversations along with a thorough performance and security assessments for three hiding methods (i.e., IV, RCT and RLE). Thus, the main contribution of the work is in the evaluation of potential TLS channels mainly to anticipate

attackers and malware developers interested in increasing their stealthiness, especially when exfiltrating data or orchestrating compromised nodes.

In all cases, production-quality network security tools (i.e., Bro/Zeek, Snort, and Suricata) failed to reveal the presence of the channel when deployed "out of the box". According to collected results, a possible approach to detect a covert channel within a TLS communication could use some information gathered at the network level. In this case, results have indicated that the IV method is difficult to spot, mainly due to its reduced network footprint. Instead, methods RCT and RLE produce some "signatures", e.g., large number of alerts and data messages alternating in time that can be used to feed a detection framework. Unfortunately, this requires great traffic volumes to "tune" suitable models.

Future works aim at developing and testing more sophisticated countermeasures, for instance the development of ad-hoc proxies to completely sanitize TLS streams from information-hiding attacks. Another relevant part of our research is focused on the creation of a framework leveraging some form of artificial intelligence for inspecting traffic and synthesizing suitable rules for the automatic configuration of de-facto standard network security tools. We also aim at investigating possible covert channels created via manipulated/bogus SSL libraries installed within the host of the victim. Lastly, we are working towards exploring more sophisticated attacks, especially those acting in a MitM flavor.

Acknowledgments

This work has been supported by EU Project SIMARGL - Secure Intelligent Methods for Advanced Recognition of Malware and Stegomalware (`https://simargl.eu`), Grant Agreement No 833042.

References

[1] A. Iacovazzi and Y. Elovici. Network flow watermarking: a survey. *IEEE Communications Surveys & Tutorials*, 19(1):512–530, September 2016.

[2] 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.

[3] W. Mazurczyk and L. Caviglione. Information hiding as a challenge for malware detection. *IEEE Security & Privacy*, 13(2):89–93, April 2015.

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

[5] B. W. Lampson. A Note on the Confinement Problem. *Communication of the ACM*, 16(10):613–615, October 1973.

[6] Z. Wang and R. B. Lee. Covert and side channels due to processor architecture. In *Proc. of the 22nd Annual Computer Security Applications Conference (ACSAC'06), Miami Beach, Florida, USA*, pages 473–482. IEEE, December 2006.

[7] 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.

[8] R. Holz, J. Amann, O. Mehani, M. Wachs, and M. Kaafar. Tls in the wild: An internet-wide analysis of tls-based protocols for electronic communication. In *Proc. of the 2016 Network and Distributed System Security Symposium (NDSS'16), San Diego, California, USA*, pages 1–15. NDSS, February 2015.

[9] R. Rios, J. Onieva, and J. Lopez. Covert communications through network configuration messages. *Computers & Security*, 39:34–46, November 2013.

[10] W. Fraczek, W. Mazurczyk, and K. Szczypiorski. Hiding information in a stream control transmission protocol. *Computer Comm.*, 35(2):159–169, January 2012.

[11] A. Velinov, A. Mileva, S. Wendzel, and W. Mazurczyk. Covert Channels in the MQTT-Based Internet of Things. *IEEE Access*, 7:161899–161915, November 2019.

[12] W. Mazurczyk. Voip steganography and its detection—a survey. *ACM Computing Surveys*, 46(2):1–21, December 2013.

[13] C. Wang, Y. Yuan, and L. Huang. Base communication model of ip covert timing channels. *Frontiers Computer Science*, 10(6):1130–1141, December 2016.

[14] W. Mazurczyk and L. Caviglione. Steganography in modern smartphones and mitigation techniques. *IEEE Communications Surveys & Tutorials*, 17(1):334–357, August 2014.

[15] J. Merrill and D. Johnson. Covert channels in ssl session negotiation headers. In *Proc. of the 2015 International Conference on Security and Management (SAM'15), Las Vegas, USA*, pages 70–72. The Steering Committee of The World Congress in Computer Science, Computer, July 2015.

[16] J. Kaur, S. Wendzel, O. Eissa, J. Tonejc, and M. Meier. Covert channel-internal control protocols: Attacks and defense. *Security and Communication Networks*, 9(15):2986–2997, October 2016.

[17] C. Heinz, W. Mazurczyk, and L. Caviglione. Covert channels in transport layer security. In *Proc. of the 2020 European Interdisciplinary Cybersecurity Conference (EICC'20), Rennes, France*, pages 1–6. ACM, November 2020.

[18] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. IETF RFC 5246, August 2008. https://www.rfc-editor.org/rfc/rfc5246.txt.

[19] E. Crockett, C. Paquin, and D. Stebila. Prototyping post-quantum and hybrid key exchange and authentication in tls and ssh. In *Proc. of the 2nd NIST PQC Standardization Conference (PQC'19), Santa Barbara, California, USA*, page 858. NIST, July 2019.

[20] M. Urbanski, W. Mazurczyk, J.-F. Lalande, and L. Caviglione. Detecting local covert channels using process activity correlation on android smartphones. *International Journal of Computer Systems Science and Engineering*, 32(2):71–80, March 2017.

[21] D. Eastlake 3rd. Transport Layer Security (TLS) Extensions: Extension Definitions. IETF RFC 6066, January 2011. https://www.rfc-editor.org/rfc/rfc6066.txt.

[22] L. Caviglione, M. Gaggero, E. Cambiaso, and M. Aiello. Measuring the energy consumption of cyber security. *IEEE Communications Magazine*, 55(7):58–63, July 2017.

[23] L. Caviglione. Can satellites face trends? the case of web 2.0. In *Proc. of the 2009 International Workshop on Satellite and Space Communications (IWSSC'09), Siena-Tuscany, Italy*, pages 446–450. IEEE, September 2009.

[24] W. Mazurczyk, K. Powójski, and L. Caviglione. Ipv6 covert channels in the wild. In *Proc. of the 3rd Central European Cybersecurity Conference (CECC'19), Munich, Germany*, pages 1–6. ACM, November 2019.

[25] L. Caviglione, W. Mazurczyk, M. Repetto, A. Schaffhauser, and M. Zuppelli. Kernel-level tracing for detecting stegomalware and covert channels in linux environments. *Computer Networks*, 191:108010:1–108010:12, May 2021.

[26] L. Caviglione, M. Zuppelli, W. Mazurczyk, A. Schaffhauser, and M. Repetto. Code augmentation for detecting covert channels targeting the IPv6 Flow Label. In *Proc. of the 7th International Conference on Network Softwarization (NetSoft'21), Tokyo, Japan*, pages 450–456. IEEE, June 2021.

Author Biography

Corinna Heinz received her Bachelor degree in Computer Science in 2015 at the FernUniversität in Hagen. There, she continued her studies to earn her Masters degree in 2020 with focus on network security. From 2005 until 2020 she worked at an internet service provider, where her primary responsibilities included server housing and domain registrations, as well as auditing the information security management system. Her current position is system architect at a governmental agency.

Luca Caviglione received the Ph.D. degree in electronics and computer engineering from the University of Genoa, Genoa, Italy. He is a Senior Research Scientist with the Institute for Applied Mathematics and Information Technologies of the National Research Council of Italy, Genoa. His research interests include optimization of large-scale computing frameworks, wireless and heterogeneous communication architectures, and network security. He is an author or co-author of more than 150 academic publications and several patents in the field of p2p and energy-aware computing. He has been involved in many research projects funded by the European Space Agency, the European Union, and the Italian Ministry of Research. He is a Work Group Leader of the Italian IPv6 Task Force, a contract professor in the field of networking/security and a professional engineer.

Marco Zuppelli is a second year PhD student at University of Genoa and a research fellow at the Institute for Applied Mathematics and Information Technologies of the National Research Council of Italy. Within the European Project SIMARGL (Secure Intelligent Methods for Advanced RecoGnition of malware and stegomalware), he investigates novel detection methods for steganographic malware exploiting both network and local covert channels. His main research interests are the use of in-kernel methodologies (e.g., the extended Berkeley Packet Filter) to collect information on software/network components, and the design of mechanisms for detecting malicious communications in an efficient, scalable and extensible manner.