A Preliminary Study on the Creation of a Covert Channel with HTTP Headers

Stefano Bistarelli[1], Michele Ceccarelli[3], Chiara Luchini[1,2], Ivan Mercanti[1] and Francesco Santini[1]

[1]*Dipartimento di Matematica e Informatica, Università degli Studi di Perugia, Via Vanvitelli 1, 06123 Perugia (PG), Italy*
[2]*Dipartimento di Matematica e Informatica "Ulisse Dini", Università degli Studi di Firenze, Viale Giovanni Battista Morgagni, 67/a, 50134 Firenze (FI), Italy*
[3]*Colacem S.p.A., Gubbio (PG), Italy*

Abstract

Steganography conceals confidential information within seemingly innocuous data and has evolved with technological advancements. In network steganography, data is hidden in packets exchanged at different levels (e.g., Ethernet, IP, TCP, etc.). This paper considers the HTTP protocol for setting up a covert channel between two endpoints: the main motivation is that creating ad-hoc HTTP packet headers does not require superuser privileges, while TCP segment headers, for example, require them. This simplifies the execution of tools implementing the channel. Moreover, HTTP/HTTPS traffic is usually allowed to flow to/from a local network and is often not modified (if not automatically proxied). Therefore, we propose a detailed exploration of a covert channel protocol by modulating standard fields in the HTTP headers for unidirectional communication, i.e., from a sender to a receiver.

Keywords

Network Steganography, Covert Channels, HTTP

1. Introduction

Steganography [1, 2] is an ancient technique used for centuries to hide sensitive information from public view in seemingly innocent material. Its development throughout time, spurred by technological breakthroughs, has produced a variety of steganographic methods that have increased its applicability in a wide range of fields. A pivotal moment in 2003 marked the introduction of "network steganography" [3], often referred to as *covert channels* [4], emerging as a widely implemented type in practical settings. In general terms, covert transfer of information always features the following elements regardless of its specific: a *covert sender*, the entity that sends secret information, and a *covert receiver*, the entity that receives secret information. Then, the *covert object*: is the data carrier in which the covert sender hides secret information. It must be selected so that it does not represent an anomaly but at the same time has enough embedding

ITASEC 2024: The Italian Conference on CyberSecurity, April 8-12 2024, Salerno, Italy
stefano.bistarelli@unipg.it (S. Bistarelli); chiara.luchini@collaboratori.unipg.it (C. Luchini); ivan.mercanti@unipg.it (I. Mercanti); francesco.santini@unipg.it (F. Santini)
https://bista.sites.dmi.unipg.it/ (S. Bistarelli); https://francescosantini.sites.dmi.unipg.it/ (F. Santini)
0000-0001-7411-9678 (S. Bistarelli); 0009-0001-6846-0922 (C. Luchini); 0000-0002-9774-1600 (I. Mercanti); 0000-0002-3935-4696 (F. Santini)

capacity. Finally, a *representation* specifies how secret information is embedded in the covert object.

Network steganography can be employed in various legitimate and potentially malicious applications. Some examples of possible applications include data exfiltration in espionage and Intelligence, confidential business communication, malicious activities (malware communication or command-and-control traffic to evade IDSs), anonymous communication and anti-censorship, and copyright protection (embedding information to identify the origin or ownership of digital content).

The creation of a thorough taxonomy for the scientific community was prompted by the widespread usage of covert channels over time, which addressed issues ranging from cyber-security to terrorism. When the "Information Hiding Project"[1] was first launched, it used a model-based classification system to classify different steganographic methods and suggested performance assessment indices to gauge how effective they were over a wide range of covered channels.

Practical implementations of network steganography may involve the manipulation of various communication protocols, such as IP [5], TCP [6], UDP [7], ICMP[2], DNS [8], and HTTP [9]. Researchers in this field continually work to develop new techniques that can evade monitoring and detection systems, leading to an ongoing evolution of concealment strategies. Network steganography poses significant challenges in the context of cybersecurity, requiring a constant effort to develop advanced detection methods. Its growing relevance is highlighted by the need to explore new approaches and countermeasures to protect digital networks from using steganography for malicious purposes.

Our proposal emphasises a detailed definition of an HTTP-level covert channel protocol. The prototype's client and server components use HTTP headers for one-way communication while prioritising particular features. The following sections make up the structure of this paper. In Section 2, we provide an overview of the HTTP protocol and some technical elements of network steganography, including nomenclature and taxonomy. Section 3 presents the most interesting works on HTTP steganography. Section 4 describes our prototype idea. We explore our findings and possible future directions for this study in Section 5.

2. Background

This Section presents an overview of the network steganography and HTTP protocol.

2.1. Network Steganography

The fundamental concept of network steganography is to hide messages or data within other seemingly innocuous data, ensuring that the act of concealment does not raise suspicions. Unlike traditional forms of steganography, which often target images or audio files, network steganography centres on manipulating data packets, frames, or communication protocols within a computer network. Krzysztof Szczypiorsky originally presented the idea of network

[1] https://patterns.ztt.hs-worms.de
[2] https://www.rfc-editor.org/rfc/rfc1256.

steganography in 2003 [10]. Compared to other well-known methods like picture steganography, this new kind of steganography had received little attention before its presentation. This modern branch has seen tremendous growth regarding communication concealment in the last few decades, bringing many innovative network steganography techniques to the scientific community.

Researchers use terms such as “information hiding" and “covert channel" to refer to the same technique, which involves concealing information in network protocols [11]. Differences between Lampson ’s [12] initial definition of a covert channel and a later definition by the *US Department of Defence* (*US DoD*) [13] have partly contributed to this. This paper refers to a *covert channel* as a hidden or secret channel intended to facilitate discrete data transmission between two peers by covertly exchanging information within a network protocol. On the other hand, a *overt channel* is a recognised channel where a sender and a receiver can legitimately communicate information. When discussing the sender and receiver, it is important to make clear distinctions. Specifically, an *Overt Sender* (*OS*) is defined as the individual who transmits data through a legitimate channel, while an *Overt Receiver* (*OR*) is the individual who receives the data. In contrast, a *Secret Sender* (*SS*) is defined as the entity sending data in a hidden channel, while the *Secret Receiver* (*SR*) is the entity receiving it. It is important to note that SS and SR may not always align with OS and OR, making the latter unaware that third parties are using their communication for other operations. Illegitimate communication in a covert channel involves two processes: embedding and extraction. The embedding process allows the sender to conceal secret data within legitimate communication, while the extraction process allows retrieving such data.

Traditionally, covert channels were categorised into *Covert Storage Channels* (*CSC*) and *Covert Timing Channels* (*CTC*), although there is no fundamental distinction between them [13, 14]. Storage channels involve the sender’s direct/indirect inscription of object values and the receiver’s direct/indirect reading of these values. On the other hand, timing channels entail the sender signalling information by modulating resource usage (e.g. CPU usage) over time, allowing the receiver to observe and decode the transmitted data. The methodologies employed in the construction of covert channels are numerous and diverse. However, they may be broadly classified into two categories: those that alter the bits of packets, thereby storing information directly in the traffic and those that modify the timing or behaviour of the flow, allowing the receiver to decode covert data by observing and interpreting the traffic. Recently, a third category, referred to as *hybrid channels*, has been introduced alongside storage and timing channels [15]. These techniques combine the utilisation of both storage and timing methodologies.

2.2. The HTTP Protocol Header Fields

Hypertext Transfer Protocol (*HTTP*) is a client-server protocol that is used to fetch resources such as HTML documents. It is the foundation of web data exchange and is reconstructed from sub-documents such as text, images, videos, and scripts [16]. HTTP requests are composed of headers[3] and a body. The headers convey essential information for processing the data in the

[3]HTTP headers: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

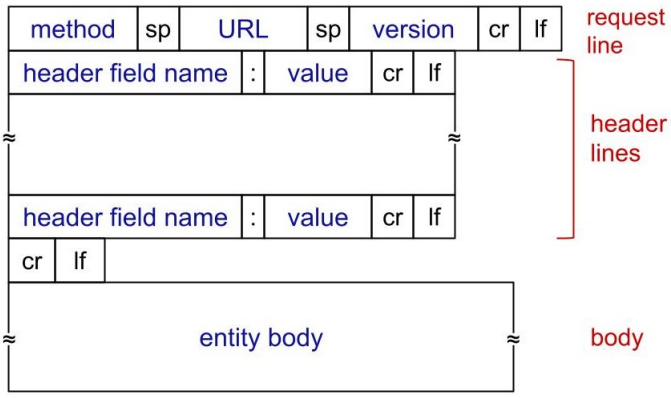

Figure 1: Structure of an HTTP request message [17].

body. Clients include fields in the requests to inform the server about their handling capabilities and preferences for receiving requested resources. For example, clients may specify their ability to process compressed resources, preferred language, or acceptance of older resource versions. However, it is essential to note that although clients express these preferences, servers may disregard them or respond based on their preferences if they cannot comply.

HTTP request headers in Figure 1, adhere to a consistent structure, comprising a case-insensitive string followed by a colon (':') and a value. The entire header, including the value, is on a single line, which may be lengthy. Requests may feature diverse headers categorised into groups [18]:

- *General headers* refers to an HTTP header that may be used in both request and response messages but is unrelated to the content.
- *Request headers*, such as *User-Agent* or *Accept*, further specify or provide context to the request. Examples include *Accept-Language* for language preference and *Referer* for contextual information. Some, like *If-None-Match*, conditionally restrict the request.
- *Representation headers*, including *Content-Type*, delineate the original format of the message data and any applied encoding. These headers are only present if the message includes a body.

Requests and responses feature standard and personalised fields, encompassing proxy settings, security configurations, and server-set parameters. Standard request fields communicate the client's characters, encodings, manipulations, and language capabilities. They also provide details about the request, such as date, user agent, or authentication-related data. Meanwhile, response headers convey data specifications like length, type, encoding, or hash, with additional fields expediting resource processing. Notably, the *Set-Cookie* header in responses communicates cookies the client should set for future interactions.

Non-standard headers cater to more resource-specific details, such as *X-Content-Duration*, indicating the duration of audio or video content in seconds. Given the likelihood of requests passing through various systems, including proxies, before reaching the server, these intermedi-

aries may modify or control parameters to optimise delivery. We will go over in detail each HTTP header field that will be employed in our model:

- *Accept:* This field contains the MIME types accepted for the response.
- *Accept-Encoding:* The Accept-Encoding field contains the encoding formats accepted for the response, which can be a value or a list of values.
- *Accept-Language:* This field allows clients to choose the language(s) they want to receive the requested resource. As a result, one or more values can be specified as well as the ":q=" which expresses a preference among several options.
- *Accept-Datetime:* This field contains the version date of the requested resource. Dates are written using the standard format "<day-name>, <day> <month> <year> <hour>:<minute>:<second> <time-zone>".
- *From:* The From field can contain the contact information of the person who submitted the request, which is useful if any issues need to be resolved by the server. The standard requires this contact to have a traditional email address.
- *If-Match*: The If-Match field determines whether a resource request has been altered when using the POST method. It has an identifier or list of them, and the request won't be handled until one matches the one saved in the server. The most popular method for generating resource identifiers is using a hash, such as SHA-256.
- *Range:* This field will only be included in the request if the If-Match field is also present. This is because, in most circumstances, they are used together to request (or modify) a specified resource portion. This field is used to provide the two hypothetical byte ranges.
- *TE:* This standard field in GET requests allows clients to specify the transfer encodings they accept. In HTTP protocol versions 2 and 3, this field is only permitted when set to *trailers*[4].
- *User-Agent:* The User-Agent value is necessary for communication and indicates the client version making the request. The server uses this information to determine which software it is communicating with. It helps to decide which data to handle and apply further optimisations on top of what is supplied in the other fields.

3. Related Work

The highest layers of the ISO/OSI stack, the application layer protocols, have also been utilised to suggest several hidden channels. At this level, the protocols we uncover can be client-server or peer-to-peer, where users share information collaboratively. The primary application-level protocol for information transmission on the Web is HTTP. Although a more secure TLS-based version (HTTPS) is available, almost all organisations still permit Internet surfing over HTTP. Dyatlov et al. [19] presented storage channels that exploit the HTTP request/response header and/or body. The amount of allowed headers changes depending on the web server version, making an accurate performance evaluation of these strategies impossible.

One way to transmit instructions and output them secretly over HTTP is using the Reverse WWW Shell tool [20]. On the other hand, Bowyer [21] encodes messages in URL parameters

[4]https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/TE

or after GET requests and utilises these hidden channels to connect with Trojans hidden behind firewalls. To build an anonymous overlay network, Bauer [22] suggests using common online user communications, such as headers, cookies, redirects, HTML components, and active content. The majority of these methods, along with the currently available HTTP covert channels analyses, are documented in the work by Brown et al. [23].

Heilman et al. [24] presents a covert channel that mimics stealthy behaviour by leveraging the base Linux shell and command language while relying minimally on system resources through the usage of the User-Agent string in the HTTP Request Header. The channel's usage of HTTP enables it to blend in with network traffic and propagate over wide-area networks, possibly expanding its reach and making it more accessible to Bash shell users.

Kwecka [25] proposes a method to embed covert data into HTTP headers by leveraging the protocol's treatment of various amounts of whitespace as a single character. For instance, tabulation can represent 1, while a standard space can represent 0. Additionally, the varying capitalisation of letters can be used for secret data transfer.

Ji et al. [26] introduced a technique based on the length of HTTP packets, achieving a recorded performance of 50 bytes, which includes 20 bytes for the TCP Header, 20 bytes for the IP Header, and 18 bytes for the Ethernet Header.

Alman [27] elucidates how a connection can be established through a proxy server by exploiting a vulnerability in the CONNECT method. Van Horenbeeck [28] developed the Wondjina tool, enabling a client to validate its cached copy using HTTP Entity tags. Similar concepts are applied in [29], incorporating LSB approaches on the Date and Last Modified fields.

4. A Prototype

Notably, real-world implementations have been articulated primarily for the IP, TCP, UDP, ICMP, DNS, and HTTP protocols. The prominence of the first three protocols is attributed to the substantial generation of network traffic at the upper layers of the ISO/OSI stack, enabling broader applicability to numerous data packages. However, the requisite root permissions for operations at these layers limit potential application scopes.

ICMP and DNS emerge as commonly used protocols, offering easily accessible payloads and the ability to initiate conversations without necessitating root privileges. Nevertheless, their widespread use has led to the proliferation of monitoring techniques leveraging artificial intelligence, which poses a challenge.

The HTTP protocol, operating at the Application level and associated with web browsing, dominates internet traffic within contemporary computer networks. Monitoring systems that comprehensively analyse HTTP packets are notably scarce, with a prevalent focus on communication characteristics. Anomalies, such as an imbalance in data transmission between client and server during a single connection, may indicate potential covert communication. Despite HTTP protocol communications generally circumventing the need for root access, most implementations deviate from pure network steganography, employing tunnelling methods within ostensibly innocuous HTTPS traffic.

Our objective encompassed the development of a protocol designed to operate seamlessly on most computers and evade detection by large-scale systems. The choice of the application level

was deliberate, considering the absence of root privileges and the widespread use of the HTTP protocol among users. The decision to focus on a unidirectional channel, akin to data exfiltration, aimed to enhance the methodology's versatility across scenarios. Nevertheless, the outcomes exhibit adaptability and can be easily extended to facilitate bidirectional communication, thus broadening its potential applications. The data integration approach within HTTPS packages was achieved through a unified solution using different header fields of an HTTP request.

After examining the several recommendations in the literature, we concluded that [30] email suggestion would be most helpful for our investigation. Initially, they proposed entering information in the "Message-ID" and "Content-Type" boxes. We modified the method to represent the data according to the characteristics of the different fields and expanded the methodology to HTTP request headers. Before discussing each component's operation in depth, we specified the modulation of HTTP headers for data exfiltration.

4.1. HTTP Request Modulation

There are numerous available fields for creating queries, including nonstandard ones. Some fields imply the presence or absence of others. We also consider the maximum number of bits that could be injected into a single field and avoid fields that might be modified. To construct our requests, we follow the format of requesting a portion of a previously passed resource using the GET method. In Section 2, we listed all the HTTP header fields included in a GET request.

This Section describes the technique used to represent bit strings for each field. Powers of 2 are used primarily to facilitate the representation of possible binary strings. The technique proposed in this paper falls under the PS11 category, as it involves *Value Modulation* and preserves the structure [31]. We describe a potential modulation of the GET request's HTTP header fields and specify the appropriate course of action for each. Table 2 shows the modulation of the Accept, Accept-Enconding and Accept-Language fields, while Table 3 displays the modulation of the Accept-Datetime, From, Range and TE fields.

Accept. The *Accept* field specifies the MIME types accepted for the response. We focus on the most common MIME types such as `text/plain`, `text/css` and so on. In this case, there can be 16 possible values, allowing hiding a sequence of up to 4 bits within this field.

Accept-Encoding. Even with the *Accept-Encoding* field, which lists the accepted encoding types for the response, the options should be limited to the eight most common types. This would allow for the hiding of 3-bit sequences.

Accept-Language. The client uses the *Accept-Language* field to indicate the preferred language(s) for receiving the requested resource. In our case, we considered three languages: Italian, American English, and British English. Italian was chosen because it is the language of the authors of this paper, while American and British English were selected due to their widespread usage. We also considered the preference value ' q' and the presence or absence of the considered languages to expand the representable data set. Unlike the other fields, the Accept-Language field can steganograph a maximum of three bits, but it can also be used for bit sequences of shorter lengths.

Accept-Datetime. In this context, the version date of the requested resource is denoted, employing a standard date format: " <day-name><day><month><year><hour>:<minute>:<second><time-zone>". We use all fields except the first and last to exploit this date format. The first field is omitted due to its dependency on others, and the last is disregarded because requests from the same machine cannot differ in time zone.

Like other fields, the approach involves considering the maximum power of two for each piece of data, with binary strings assigned to that number of elements. For instance, considering February the shortest month with 28 days, the largest power of two less than 28 is 16 (2^4). This logic modulates the day-name value, minutes, hours, and seconds. The first eight (2^3) months are considered, and the years from 1991 to 2022, resulting in 32 (2^5) valid values, avoiding dates before the internet's inception. The day of the week is determined once the date is generated, and the specified Italian CET time zone can be used. However, these two values are disregarded during the decoding process. In summary, it is feasible to conceal 26 bits, considering string lengths of 4 for the day and hour, 3 for the month, and 5 for the year, minutes, and seconds.

From. This field includes the contact information of the request submitter, providing valuable details for server issue resolution. The standard mandates that this contact information adhere to a traditional email address format.

To address this, we acquire databases containing the most popular DNS names in the US over recent years, merge them, and generate a dictionary comprising 32,768 names. Each entry in this dictionary corresponds to a 15-bit binary string (2^{15}=32768). Additionally, we identify eight of the most popular email domains and assign them their respective 3-bit binary strings.

If-Match. The *If-Match* field is employed to ascertain whether a resource request has undergone alterations when using the POST method. This field includes an identification or a list of identifiers, and the request is not processed unless it matches the identifier stored on the server. Typically, resource IDs are produced with a hash algorithm such as SHA-256. We assume that SHA-256 was used for digest computation and that the request includes two hashes. We transform 256 bits of the secret message into hexadecimal digits and enter them in the field. Given that SHA-256 yields a 256-bit digest, we may mask 512 bits in this field by entering two numbers. Notably, this field and its accompanying Range field (described in the next paragraph) are not used if less than 256 bits of data must be sent.

Range. As previously stated, the *Range* field appears in the request only when the *If-Match* parameter is included. These fields are commonly used to request or edit a specified portion of a resource. This parameter is used to provide two hypothetical byte ranges. Recognising that a resource may consist of several bytes, we consider it fair to utilise integers between 0 and 1023 (2^{10} values) to define two intervals. Each number is now assigned a 10-bit binary string, which allows for hiding 40 bits inside this field. The first value is extracted from the first ten bits, followed by the second value from the ten bits after that. The initial number is then added to the latter to avoid an unreasonable interval in which the end is smaller than the start. The same approach is used to compute the second interval.

TE. This standard field is present in GET requests and allows the client to declare which transfer encodings it is ready to accept. Recognising its fundamental role in requests and consistent presence, it was deemed suitable for concealing steganography information. Given the restricted choices, the four most regularly used values were assigned to 2-bit binary strings.

User-Agent. The *User-Agent* field is obligatory in communications and serves to identify the client version initiating the request. This information is important for the server to discern the program with which it interacts, enabling the selection of data handling and implementing optimisations beyond those offered in other fields.

In this context, modulation is eschewed, and instead, the identifying string of a Mozilla version is entered into each request. This choice is based on Mozilla being the most prevalent browser across various systems. Detecting requests from the same computer with multiple User-Agents in a brief period would likely raise concerns.

Request structure		
Field	**Example**	**Bit**
Accept:	text/…, image/…, video/…, application/…	4
Accept-Encoding:	gzip, deflate, compress, br, identity, *	3
Accept-Language:	it-IT,it;q=0.9,en-US;q=0.8,en;q=0.7	3
Accept-Datetime	Wed, 21 Oct 2015 07:28:00 GMT	26
From:	aristea@libero.it	18
If-Match:	(x2) hash da 256bit	512
Range:	bytes:1207-2367	40
TE:	compress, deflate, gzip, trailers	2
User-Agent	Mozilla/5.0 (Windows NT 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0	-
	Total bit sent	608 bit = 76 Byte

Table 1
Example of HTTP header structure.

Table 1 provides an example of HTTP request fields associated with the corresponding amount of bits that may be modulated.

4.2. Implementation Details

The proposed prototype has two primary components: *sender* and *receiver*. The first takes as input arguments the file or directory to steganograph inside headers and the URL to make requests to. Instead, the receiver is a web server listening on a specific port.

The client executable, denoted as the sender, primarily functions to embed steganographically or exfiltrate a file or directory via HTTP requests. Its workflow is as follows:

1. *Preliminary operations*: The sender prints the tool banner and generates the dictionaries for the Range and From fields.
2. *Parameters*: The sender takes the URL and the path to the file or directory to be sent as input parameters.

3. *Beginning of communication*: the communication begins with a request that has the value of the Accept field set to *application/zip*, with the remaining fields either left blank or, if necessary, defaulted.
4. *Sending phase*: sender manages path reading, file or directory identification, request creation, and send. The file is opened in binary read mode, and an estimate of the number of requests required is reported to the terminal. The process reads the bit blocks of the file, creating requests based on the binary string obtained from the file. Each field is assigned a dictionary, with the key being the binary string and the associated value representing the information to be written into the request. The file transfer process involves sending a new request when the input file's length is incompatible with the binary string. If this happens, a new request is created cyclically until the entire block is sent. The number of requests sent is updated at each step, providing an estimate of the completion percentage.
5. *Ending phase*: the server is informed of the end of exfiltration operations by sending a request with the *Accept* field set to *application/rtf*. The total number of requests used in the process is displayed on the screen, managed in a variable updated each time a request is sent.

Contrarily, the receiver is a web server listening on a designated port, mirroring the sender's structure.

1. *Preliminary operations*: the server's execution is based on a main function, which prints the terminal banner and creates dictionaries related to person names and ranges. The distinction from those the client uses is that the key is the information found in the request, while the binary string represents the related value. The port number and file names to be generated are prerequisites; otherwise, default values, specifically port 8000 and the name *output_file*, will be automatically assigned.
2. *Extraction phase*: the headers are initially stored in a variable upon receiving each GET request. Subsequently, the HTML page for the response is chosen, and the headers are parsed to extract relevant information. The receiver aims to simulate the functioning of a real web server, responding with HTML pages of varying sizes without emphasising their actual content. In response to each request, the receiver employs a random web page, concluding the communication to prevent anomalies. Post-response, the server analyses the previously saved headers for information. It performs a reverse process from the client, reading request data and employing dictionaries to derive binary strings, which are sequentially stored. Following the parsing of each request, the first byte of the constructed binary string is examined, representing the length of the block read and sent. The corresponding bytes are written to the destination file if the sum of other bits surpasses the previously read value. Throughout this process, the terminal is updated with the count of received requests.
3. *Ending phase*: once the server receives a request with the Accept field set to application/rtf, it writes the last bytes to the file and communicates that operations are complete.

Notably, such a prototype, operating at the bit level, can transmit any data stream. Table 1 depicts all of the header fields of an HTTP GET request issued; the file sent requires around eight queries, this being the sixth. A transmitted file of a size of 459 bytes produced a total of 9 requests.

5. Conclusions and Future Work

This paper reviewed network steganography and some related approaches in the literature by summarising the fundamental characteristics of this research field. Steganography comprises the science and art of hiding information transfer and storage. It is not to be confused with cryptography: while they both share the ultimate goal of protecting information, the former attempts to hide it to make it "difficult to notice".

Many existing works call this type of communication a covert channel, referencing a concept first introduced by Lampson in 1973 [12]. The latter part of this paper outlines our proposed implementation of a covert channel. Section 4 elucidates the rationale behind critical development decisions and provides a comprehensive account of the implementation details. Our prototype is rooted in HTTP/HTTPS requests, where we modulate header values to embed our targeted information steganographically.

While not explicitly addressed, a fundamental consideration for the technique's development is its behaviour in the presence of HTTP proxies along the packet path. In this case, requests for web resources are routed via the proxy server rather than directly to the destination server. After retrieving the response from the destination server, the proxy server relays the request back to the client. HTTP proxies can modify several fields in the HTTP header as they process requests and responses between clients and servers. For example, proxies might modify the Accept-Encoding header to perform content compression.

Given this, we think a future direction could be to adapt channels to different contexts, i.e., by assembling a portfolio of them, for example, with the proposal in [32] by some of the authors of this work. Instead of relying solely on one steganography method, a general parametric framework could switch between several potential alternatives based on information gathered on the target network. While there is still no guarantee that this channel would evade the defender's countermeasures, it has two benefits over a channel that employs a single steganography approach. The first benefit is that it drives up costs for the defender because it is probably necessary to deploy more tools (or, at the very least, configure the ones already there more capillary) to secure the network against different exfiltration strategies. The second benefit is that as the defender's setup gets more intricate, there is a greater chance of human error; if even one exfiltration method works, the attacker could finally prevail.

Acknowledgments

The authors are members of the Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM). This work has been partially supported by:

- GNCS-INdAM, CUP_E53C22001930001 and CUP_E53C23001670001;
- European Union - Next Generation EU PNRR MUR PRIN - Project J53D23007220006 EPICA: "Empowering Public Interest Communication with Argumentation";
- University of Perugia - Fondo Ricerca di Ateneo (2020, 2021, 2022) - Projects BLOCKCHAIN4FOODCHAIN, FICO, AIDMIX, "Civil Safety and Security for Society";
- European Union - Next Generation EU NRRP-MUR - Project J97G22000170005 VITALITY: "Innovation, digitalisation and sustainability for the diffused economy in Central Italy";

- Piano di Sviluppo e Coesione del Ministero della Salute 2014-2020 - Project I83C22001350001 LIFE: “the itaLian system Wide Frailty nEtwork” (Linea di azione 2.1 “Creazione di una rete nazionale per le malattie ad alto impatto” - Traiettoria 2 “E-Health, diagnostica avanzata, medical devices e mini invasività”).

References

[1] D. Kahn, The history of steganography, in: R. Anderson (Ed.), Information Hiding, Springer Berlin Heidelberg, Berlin, Heidelberg, 1996, pp. 1–5.

[2] O. I. Abdullaziz, V. T. Goh, H.-C. Ling, K. Wong, Network packet payload parity based steganography, in: 2013 IEEE Conference on Sustainable Utilization and Development in Engineering and Technology (CSUDET), IEEE, 2013, pp. 56–59.

[3] J. Lubacz, W. Mazurczyk, K. Szczypiorski, Principles and overview of network steganography, IEEE Communications Magazine 52 (2014) 225–229. doi:10.1109/MCOM.2014.6815916.

[4] S. Bistarelli, M. Ceccarelli, C. Luchini, I. Mercanti, F. Santini, A survey of steganography tools at layers 2-4 and HTTP, in: Proceedings of the 18th International Conference on Availability, Reliability and Security, ARES 2023, Benevento, Italy, 29 August 2023- 1 September 2023, ACM, 2023, pp. 81:1–81:9.

[5] A. Shamir, IP = PSPACE, J. ACM 39 (1992) 869–877.

[6] J. S. Chase, A. Gallatin, K. G. Yocum, End system optimizations for high-speed TCP, IEEE Communications Magazine 39 (2001) 68–74. doi:10.1109/35.917506.

[7] L.-A. Larzon, M. Degermark, S. Pink, UDP lite for real time multimedia applications, Hewlett-Packard Laboratories, 1999.

[8] J. Jung, E. Sit, H. Balakrishnan, R. Morris, DNS performance and the effectiveness of caching, in: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, IMW ’01, Association for Computing Machinery, New York, NY, USA, 2001, p. 153–167. URL: https://doi.org/10.1145/505202.505223.

[9] S. Lederer, C. Müller, C. Timmerer, Dynamic adaptive streaming over HTTP dataset, in: Proceedings of the 3rd Multimedia Systems Conference, MMSys ’12, Association for Computing Machinery, New York, NY, USA, 2012, p. 89–94. URL: https://doi.org/10.1145/2155555.2155570.

[10] K. Szczypiorski, Steganography in TCP/IP networks. state of the art and a proposal of a new system-HICCUPS, Warsaw University of Technology, Poland Institute of Telecommunications, Warsaw, Poland (2003).

[11] S. Zander, G. J. Armitage, P. Branch, A survey of covert channels and countermeasures in computer network protocols, IEEE Commun. Surv. Tutorials 9 (2007) 44–57.

[12] B. W. Lampson, A note on the confinement problem, Communications of the ACM 16 (1973) 613–615.

[13] C. S. C. (US), Computer Security Requirements: Guidance for Applying the Department of Defense Trusted Computer System Evaluation Criteria in Specific Environments, Dod Computer Security Center, 1985.

[14] U. D. National Computer Security Center, Trusted Computer System Evaluation Criteria,

Technical Report, DOD 5200.28-STD, National Computer Security Center, Dec 1985. http://csrc.nist.gov/publications/history/dod85.pdf.

[15] A. Ganivev, O. Mavlonov, B. Turdibekov, et al., Improving data hiding methods in network steganography based on packet header manipulation, in: 2021 International Conference on Information Science and Communications Technologies (ICISCT), IEEE, 2021, pp. 1–5.

[16] mnd web docs, An overview of HTTP, Technical Report, Mozilla, 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview.

[17] J. F. Kurose, K. W. Ross, Computer Networking: A Top-Down Approach, Pearson, 2020.

[18] mnd web docs, HTTP Messages, Technical Report, Mozilla, 2022. https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages.

[19] A. Dyatlov, S. Castro, Exploitation of data streams authorized by a network access control system for arbitrarydata transfers: Tunneling and covert channels over the HTTP protocol, Zugriff am unter http://dl.packetstormsecurity.net/papers/protocols/covert_paper.txt (2003).

[20] V. Hauser, Placing backdoors through firewalls, WindowsSecuriy.com (1999).

[21] L. Bowyer, Firewall bypass via protocol steganography, Network Penetration (2002).

[22] M. Bauer, New covert channels in HTTP: adding unwitting web browsers to anonymity sets, in: Proceedings of the 2003 ACM workshop on Privacy in the electronic society, 2003, pp. 72–78.

[23] E. Brown, B. Yuan, D. Johnson, P. Lutz, Covert channels in the HTTP network protocol: Channel characterization and detecting man-in-the-middle attacks, Journal of Information Warfare 9 (2010) 26–38.

[24] S. Heilman, J. Williams, D. Johnson, "covert channel in HTTP user-agents", in: 11th Annual Symposium on Information Assurance, ASIA'16, 2016, pp. 68–73.

[25] Z. Kwecka, Application layer covert channel analysis and detection, Undergraduate Project Dissertation, Napier University (2006).

[26] L. Ji, W. Jiang, B. Dai, X. Niu, A novel covert channel based on length of messages, in: 2009 International Symposium on Information Engineering and Electronic Commerce, IEEE, 2009, pp. 551–554.

[27] D. Alman, Http tunnels through proxies, SANS Institute (2003).

[28] M. Van Horenbeeck, Deception on the network: thinking differently about covert channels, Australian Information Warfare and Security Conference (2006).

[29] R. Duncan, J. E. Martina, Steganographic message broadcasting using web protocols, in: proceedings of: Simposio Brasilerio de Seguranca (SBSeg 2010), Fortaleza, Brasil, 2010, pp. 61–70.

[30] A. Castiglione, A. d. Santis, U. Fiore, F. Palmieri, E-mail-based covert channels for asynchronous message steganography, in: 2011 Fifth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 2011, pp. 503–508. doi:10.1109/IMIS.2011.133.

[31] S. Wendzel, L. Caviglione, W. Mazurczyk, A. Mileva, J. Dittmann, C. Krätzer, K. Lamshöft, C. Vielhauer, L. Hartmann, J. Keller, et al., A generic taxonomy for steganography methods, TechRxiv (2022). URL: http://dx.doi.org/10.36227/techrxiv.20215373.v2.

[32] S. Bistarelli, A. Imparato, F. Santini, A tcp-based covert channel with integrity check and retransmission, in: 20th Annual International Conference on Privacy, Security and Trust,

A. Appendix

HTTP field	Value	Bit
Accept	text/plain	0000
	text/html	0001
	text/css	0010
	text/javascript	0011
	image/gif	0100
	image/png	0101
	image/jpeg	0110
	image/webp	0111
	video/mpeg	1000
	video/webm	1001
	video/ogg	1010
	video/mp4	1011
	application/octet-stream	1100
	application/javascript	1101
	application/xml	1110
	application/pdf	1111
Accept-Enconding	gzip	000
	compress	001
	deflate	010
	identity	011
	gzip, compress	100
	gzip, deflate	101
	gzip, identity	110
	gzip, br	111
Accept-Language	it	0
	it, *	1
	it, en-US	00
	it, en-US:q=0.8	01
	it:q=0.9, en-US	10
	it:q=0.9, en-US:q=0.8	11
	it, en-US, en-GR	000
	it, en-US, en-GR:q=0.7	001
	it, en-US:q=0.8, en-GR	010
	it, en-US:q=0.8, en-GR:q=0.7	011
	it:q=0.9, en-US, en-GR	100
	it:q=0.9, en-US, en-GR:q=0.7	101
	it:q=0.9, en-US:q=0.8, en-GR	110
	it:q=0.9, en-US:q=0.8, en-GR:q=0.7	111

Table 2
Modulation of Accept, Accept-Enconding and Accept-Language values.

HTTP field	Value	Bit
Accept-Datetime	Thu, 01 Jan 1991 00:00:00 CET	00000000000000000000000000
	...	...
	Wed, 05 Feb 2003 12:24:13 CET	01000010110011001100001101
	...	...
	Thu, 16 Aug 2022 16:32:32 CET	11111111111111111111111111
From	gmail.com	000
	outlook.com	001
	yahoo.com	010
	proton.me	011
	virgilio.it	100
	libero.it	101
	email.it	110
	mail.com	111
Range	0	0000000000
	...	...
	673	1010100001
	...	...
	1023	1111111111
TE	compress	00
	deflate	01
	gzip	10
	trailers	00

Table 3
Modulation of Accept-Datetime, From, Range and TE values.