Journal of Universal Computer Science, vol. 24, no. 7 (2018), 864-891
submitted: 24/2/17, accepted: 24/7/18, appeared: 28/7/18

SOMSteg - Framework for Covert Channel, and its Detection, within HTTP

Waldemar Graniszewski, Jacek Krupski, Krzysztof Szczypiorski
(Warsaw University of Technology
Warsaw, Poland
w.graniszewski@ee.pw.edu.pl, krupskij@ee.pw.edu.pl, ksz@tele.pw.edu.pl)

Abstract: Due to high efficiency and relatively ease of use, application-layer covert channels, especially HyperText Transfer Protocol (HTTP), have been extensively studied in recent years. This paper extends a new steganographic method where the covert channel is created within the HTTP protocol header, i.e., trailer field [1]. HTTP is the most popular protocol for browsing the Internet and gives the possibility of information sharing. The popularity of HTTP traffic is one of the requirements for undetectable message exchange. This paper presents SOMSteg - a framework for a covert channel, and its detection as a countermeasure, within HTTP. The server's and client's parts are implemented in the JavaScript language and based on the Node.js. Several machine learning techniques can be used for anomaly detection. We tested the detection possibility of such hidden communication by Self Organizing Maps (SOMs). SOMs were also used for tuning the parameters of the covert channel settings within the HTTP trailer. The results of the performed studies are also presented.
Key Words: Information hiding, covert channels, network steganography, HTTP, SOM, machine learning
Category: C.2.0, D2.11, D4.6, I.2, I.5.3, K6.5

1 Introduction

During the First (official) International Workshop on Information Hiding, Kahn revealed that people have being used steganography[2] from the very beginning [Kahn, 1996]. For over 60 years we have been using electronic media to transfer our messages. As a result, the steganographic methods have also changed. In the 1990's, techniques for data hiding commonly use digital pictures as the medium [Bender et al., 1996, Petitcolas et al., 1999]. Almost parallel to this, information camouflaging methods were introduced, which utilized vulnerabilities and unused fields in protocols of ISO/OSI reference model, e.g., possibilities of manipulating a jamming signal by CSMA/CD in the Ethernet or to apply the two unused least significant bits in the *type-of-service* byte in the Network Layer [Handel and Sandford, 1996].

[1] This is the extended version of the paper "The covert channel over HTTP protocol", presented during the IEEE–SPIE conference: Photonics Applications in Astronomy, Communications, Industry, and High–Energy Physics Experiments 2016 [Graniszewski et al., 2016]

[2] "Steganography - The practice of concealing messages or information within other non–secret text or data"[Steganography, 2016]

Some methods protect communication just by encryption, as presented in Figure 1. However, the usual processes of encrypting a message with a special key have attracted the attention, not only of the casual observer but, in particular, the attention of potential intruders, who want to break the cipher.

Figure 1: Communication protected by encryption

Figure 2: Steganographic communication

Furthermore, sufficient encryption is not always available or allowed, e.g., the USA treated cryptography like a munition and regulated the export of cryptography [United States Department of State, 1996, Schwartzbeck, 1997]. Moreover, in some countries, the usage of cryptographically strong methods, like VPN, has been constricted or VPN software was removed from repositories, like the App Store for China [ExpressVPN, 2017]. In other states, like in Russia, some legislation amendments have been passed that are linked to backdoors in communication applications [Szoldra, 2016, Baxter, 2017]. Other countries, like the

UK [Wakefield, 2015, Haynes, 2017], tried to gain access to encrypted communications and work on prohibition of strong encryption or/and implementation of backdoors in software.

For every action there is always a reaction. Citizens who want to protect their privacy and may not use strong cryptography, have always been looking for other allowed methods to avoid censorship. Therefore, it is reasonable just to hide the message in regular data exchanges with steganographic methods, to overcome restrictions on cryptography – as shown in Figure 2. The problem with such a method is known as the prisoner's problem [Simmons, 1984].

On the other hand, terrorists can use steganography for sending secret instructions to their groups [Conway, 2003]. Moreover, intelligence can be sent using covert channels by spies to their state agencies, as proved by the FBI agent Kachhia-Patel in a complaint against Russian spies in 2010 [Kachhia-Patel, 2010].

In addition, steganography can be used to steal information from computer systems. This issue was addressed by Lampson and named as data leakage in 1973 [Lampson, 1973]. Later, in 1985, the Department of Defense described and formulated this problem in Trusted Computer System Evaluation Criteria, also known as the Orange Book [United States Department of Defense, 1985].

Structure of the paper. We have divided our paper into six parts. The rest of this paper is organized as follows: Section 2 provides a description of cover channels within HTTP. In 2.1 we describe HTTP and its way of cooperation with other protocols, within the TCP/IP protocol stack model. In 2.2 we present an overview of related steganographic techniques, which also uses HTTP as a covert channel. A description [3] of the architecture and implementation of the proposed system is presented in section 3. Next, Section 4 describes the steganalysis methods for annomalies detection in network traffic. We use a Self Organizing Map for testing usual HTTP requests and responses with HTTP traffic created by our system. Section 5 shows the test results for anomalies detection using the implemented SOMs. We provide conclusions and possible future work in Section 6.

2 Covert channels within HTTP

2.1 HTTP - introduction

HyperText Transfer Protocol (HTTP) is the most popular Application layer protocol (see Figure 3) for browsing the Internet and gives the possibility of sharing information. This protocol is stateless, i.e., it does not save information about previous transactions with a client. This feature is an advantage: HTTP requests do not have any history records so they are smaller and they do not

[3] We used the structure to describe hidden methods proposed by [Wendzel et al., 2016].

overload a server. To save any important information, for instance session data, one can use cookies or data in the URL address [Barth, 2011].

HTTP works in the client–server model and functions as a request–response protocol. When the client connects to the server, its browser sends a HTTP request to see or to get selected data. The server replies with a HTTP response. This also includes information about accepted encodings and the type of content. Sometimes in the response header, one can also find a date of last modification or the Accept–Charset [Fielding et al., 1999].

Before sending, the application's data are segmented into pieces and encapsulated by lower layer protocols, as shown in Figure 3. On the right hand side of the picture there are the layers of the TCP/IP network model. During the encapsulation, protocols headers are added to the pieces of data.

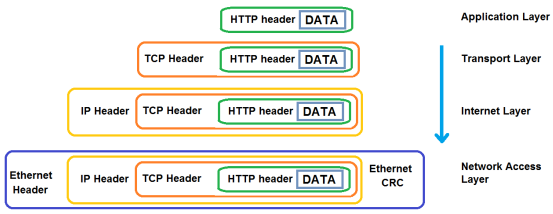


Figure 3: Data encapsulation in TCP/IP network model

As an example, a HTTP message from the application layer is wrapped by a TCP header in the transport layer. This layer is responsible for the transport reliability. Then a TCP segment is wrapped into an IP datagram in the Internet layer. Later an addressed IP datagram is wrapped in the network access layer and creates an Ethernet frame. This layer is responsible for communication between computer network devices. The Network Access layer sends each data frame to the medium [Braden, 1989].

Figure 3 presents how HTTP co–operates with other protocols, especially with TCP and IP. While traveling across a computer network each packet can go through different intermediary network devices, and each of them checks the packet's address. Routers, which work in the Internet layer, check the IP address of the datagram and forward it to the appropriate interface.

When the encapsulated information reaches the destination, starting from the network access layer and finishing at the application layer, the decapsulation process begins. In the end, the HTTP message arrives at the destination's computer application, for example, a browser.

2.2 Related steganographic techniques using HTTP protocol

As mentioned in Subsection 2.1, HTTP is one of the most popular protocols for information exchange. For this reason it seems to be a good choice as a potential steganographic medium [4].

In several countries, free use of Web content is prohibited. Therefore, early published works use these techniques to avoid censorship in the Internet. Feamster et al. [Feamster et al., 2002] used image steganography to send hidden data.

Dyatlov and Castro [Dyatlov and Castro, 2003] hid data inside HTTP requests and responses. They took into consideration various design aspects of HTTP client–server covert channel communication:

- type of server model which can be implemented,
- how to design a tool to add confusion from a traffic watcher point of view,
- types of functionality that can be applied to the covert channel.

Among the types of the server model they studied:

- httpd–like server model,
- Proxy–like server model,
- CGI–like server model.

Van Horenbeeck [Horenbeeck, 2006] discussed HTTP Entity Tag tunneling. He developed this technique during a penetration test of a more sensitive network. Though only HTTP was available to transfer data out, he checked a number of connections through the proxy using different headers, which are valid in RFC for HTTP, e.g., *general-header* (section 4.5 of RFC 2616), *request-header* (section 5.3 of RFC 2616), *response-header* (section 6.2 of RFC 2616), and *entity-header* (section 7.1 of RFC 2616) [Fielding et al., 1999]. He tested the possibility of leaking an Excel spreadsheet that contained a set of sensitive numbers.

Schear et al. proposed Glavlit [Schear et al., 2006]. They developed and evaluated a system for preventing exfiltration of data over HTTP protocol responses. They performed on–the–fly parsing of the protocol, and verified the content of the structured fields. In some cases, RFC loosely defines the header syntax and does not explicitly require presence / lack or a certain order of the response header. This can lead to undetectable usage of HTTP steganography. The developed system can restrict the usage of HTTP communication in cases of data leakage, where necessary.

[4] Good systematic reviews of all steganography and steganalysis techniques can be found in [Zander et al., 2007, Lubacz et al., 2014, Mileva and Panajotov, 2014, Mazurczyk et al., 2016].

Blasco et al. presented a framework for avoiding steganography usage over HTTP [Blasco et al., 2012]. In their paper, they analyzed the steganographic possibilities of this protocol and then developed StegoProxy as a tool for preventing steganography. They proposed an active warden model to eliminate any covert communication channel.

Wang et al. [Wang et al., 2017] presented the possibility to exploit content delivery networks (CDN) for covert channel communication. Among other aspects they also explored the possibility of applying HTTP-based covert channel attacks under the CDN environment. They constructed a proof-of-concept covert channel on Amazon CloudFront.

Schen et al. [Shen et al., 2018] discussed behavior-based covert channels in HTTP. They decided to deal with application-layer protocols because of the relatively rigorous formats of lower-layer protocols (e.g., TCP, UDP, IP). They first proposed a basic behavior-based covert channel, Lost in HTTP Behaviors (LiHB). Due to some limitations of LiHB they presented an enhanced secure HTTP Behavior-based Covert Channel (HBCC) that takes the advantage of the request-flow distributions. They exploited the natural behavior of browsers to design covert channels in which the distribution relationships between HTTP requests and flows are dynamic.

3 Proposed covert channel system

3.1 Architecture of covert channel system

The application that was developed is an example of a covert channel program. It is a client–server system that uses HTTP headers as the data carrier. To communicate, actors should have two parts of the program: a server that only sends data and a client's part that only receives messages.

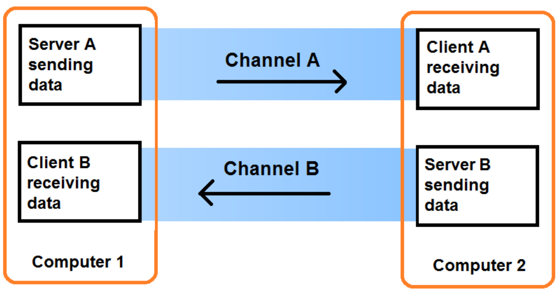


Figure 4: Two way communication used in application

The communication could be held one way (one sender and receiver) or two ways where each of the users has two parts of the application. In the first case, the server, only sends data and the client's part, only receives messages. In the second case there are two different servers sending data so there are two different communication channels, as shown in Figure 4. Such a solution gives the possibility to use two ports, which seems more secure than one port communication. In general, there could be more than one receiver that is waiting for information from the sender. There are not any intermediaries in the communication, so the covert channel is a direct one [Wendzel et al., 2016].

According to Figure 6, which presents the UML use–case diagram, there are two actors: a receiver and a sender [Booch et al., 1999]. The receiver starts the communication. If the receiver node is not started, the sender is not currently listening to the receiver's socket – the communication would not be initiated. This is based on a HTTP request and response idea. The server sends HTTP responses only when it gets HTTP requests from the receiver's side. Sending messages includes editing HTTP headers.

The sender is working on the server's side of the application that builds HTTP headers and saves a secret message in the protocol's field named *trailer*. It is an optional field. *Trailer* is used for connections based on series of chunks. In Figure 5 the communication is chunked as the Transfer-Encoding field value indicates, so a *trailer* occurs. Its presence gives the opportunity to transmit additional fields with metadata [Fielding and Reschke, 2014], for instance message integrity check.

In [Wendzel et al., 2016] one can find a description of the crucial attributes of the steganographic algorithm, which should help the reader to understand those techniques. It is worth mentioning the hidden pattern, as it standardizes hiding methods. The technique we utilize in the application, according to [Wendzel et al., 2015], is an example of an *Add Redundancy Pattern*. This pattern's full path of the hierarchy is as in the following:

```
Network Covert Channels
'--Covert Storage Channels
   '--Modification of Non-Payload
        '--Structure Modifying
           '--Add Redundancy
```

Covert Channel camouflages data in the HTTP header field so it is a Covert Storage Channel. During the communication the payload is not modified, as the data carrier is in the *trailer* field (Modification of Non-Payload). In the process of tucking secret data an optional field is added to the header, so the structure is not conserved (Structure Modifying). As the *trailer* field is not used by default, we utilize an *Add Redundancy Pattern*.

The sender's part of the application can run in five different modes. The first

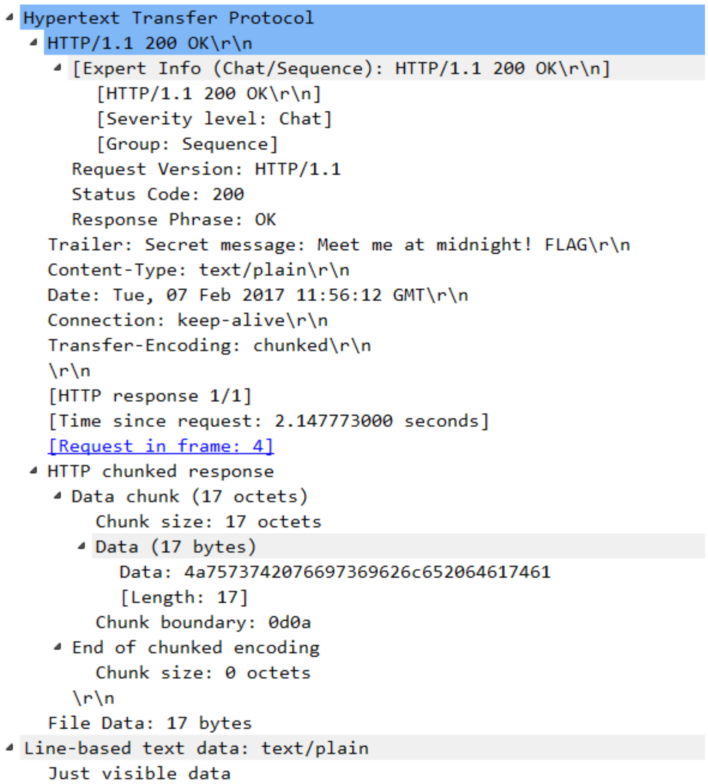


Figure 5: Caught HTTP response with covert steganographic communication made by program

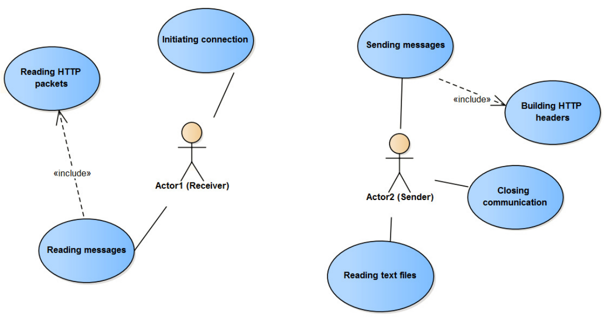


Figure 6: Use case diagram of application

one opens the program and sends the number of messages that was set. The next possibility is to send a given text in a random number of messages to the receiver. The third option is to change the port number on which the server is working [5]. The fourth mode is to read the text from a given file, and the last – default possibility, sends one HTTP message to the receiver. The receiver can reply via the second instance of the server using the same application but opened parallel. In this case, two-channel communication is utilized (see Figure 4).

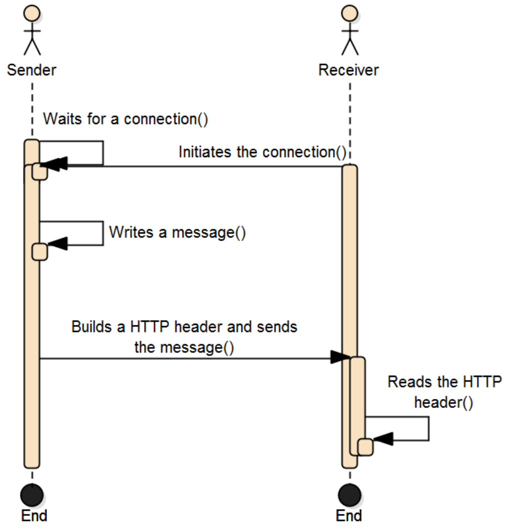


Figure 7: Sequence diagram of application

Figure 7 presents the order of sequences during the communication. First, the server starts working and the sender sets the number of messages that will be transferred to the receiver. Then the server starts listening and it enables the sending of the data. In the last message there is a special flag that closes the receiver's application. While working the receiver can restart the application, and then the remaining messages will be delivered, but the previous information will be lost.

Before communication starts the client and the sender should exchange the staganographic key. The key contains the sender's IP address, port number and data carrier. The last is built in the program. One may use the following port numbers: 80, 8008, 8080. The secrets that are encapsulated in the key have to

[5] A few ports could be used for HTTP, e.g., 80 (so called *well known* port number) or alternative ports, like 8008, 8080.

be sent via a different secured channel.

Another possibility is not to use the receiver's part of the application, but instead take advantage of a protocol sniffer. This program analyses the network traffic. Sniffers capture each packet and give a possibility of reading it. One can capture all packets and look for HTTP responses and through them find the secret message. The information about the exact time of communication makes it possible to find the HTTP response edited by the application.

3.2 Implementation

The steganographic application takes advantage of Node.js – an open–source, cross–platform JavaScript runtime environment. The framework was published in late 2009 in Berlin [Surhone et al., 2010] by Ryan Dah – a member of the Joyent group and is currently developed by the Node.js foundation. This runtime environment enables the JavaScript language to make the whole server side code independently. In this the application the HTTP servers have been used.

Listing 1: The steganographic application default mode pseudocode

```
1  server = new http.Server()
2
3  server(function (request, response){
4
5        add server Listener{
6          function(data){
7            data <- text given by the user in standard input
8              texttosend <- data, flag
9                response: write HTTP Head(
10                 in a field server's answer write "200"
11                 in a field trailer write "texttosend")
12
13           print("Message sent")
14           response: write "Just visible data" in a field
                 reserved for GET data
15         }
16       }
17 }) Server listens to the set port
```

The pseudocode in Listing 1 shows the default mode of the steganographic application, which will run when opened without any other parameters. In the first line the object server is created as an instance of the class HTTP server. Then in line 3 there is a function that sends the response only when given the request. The function in line 6 reads the input data from the user and assigns it to the variable *texttosend*. At the end of the *texttosent* variable there is a *Flag*. It is only in the last message that there is an order to stop the communication. Then in line 9 the HTTP header is created. In the field server's answer code 200 is written, in the *trailer* field, the variable *texttosend*. Then one is informed about the sent data. In line 14 the server adds to the response an additional text for a field supposed to transfer the text. Line 17 of the listing informs us that the server listens to the selected port.

To capture traffic between the server and the client the *Wireshark* sniffer was used [Lamping et al., 2014]. Figure 5 shows a caught HTTP message, with a secret communication made by the application. One can learn that the packet gives a HTTP response with server code 200. Then there is information that the HTTP version is 1.1. After that there is some data about the HTTP request, for which this HTTP message is an answer. Below that, there is a *trailer* field in which one can find the clandestine message.

Listing 2: Application open file mode pseudocode

```
1  filename <- name of the a file given by the user in standard
       input
2  Read file "filename"{
3        function(error, data){
4                if (error) throw error
5                text <- data
6                print "In the file there is a text: "
7                print text
8                length <-text.length
9
10               while (i < length){
11                       ...
12               }
13
14               server = new http.Server()
15
16               server(function (request, response){
17                       ...
18               })Server listens to the set port
19
20       }
21 }
```

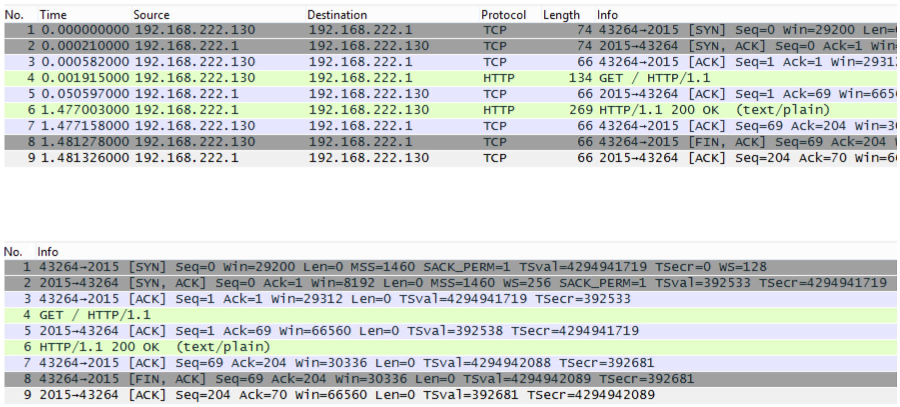
Listing 2 is the application's open file mode. In the beginning the file name is taken from the user. Then the reading file function checks whether there are any errors. In line 5 the data typed by the user is assigned to the *text* variable. Then the console shows the loaded text. In the *while* loop the text is cut into smaller pieces. In line 14 of the listing a new instance of a class HTTP server is created. This is the algorithm for creating an application server similar to that shown in Listing 1.

In Listing 3, at the beginning of the receiver's side of the application, the basic HTTP request options are set. The most important setting is the GET method in which data would be sent. Then the data is processed. Next in line 12 the secret message is read. Then the existence of the flag is checked. If there is a flag, this means that the last part of the message is processed. After printing the text, the process is closed. In lines 22 and 23 there is a condition that shows errors only if they occur.

Listing 3: Receiver's side of steganographic application pseudocode

```
create http request {
host <- "192.168.50.50"
port <- "2015"
url <- "/"
method <- "get"}

function (response){

        response: read text from GET data field

        response(function(){
                data <- text from trailer field
                end <- last 4 characters of the data
                if (end == "flag"){
                        data <- data without last 4 characters
                        print data
                        exit
                }
                print data
        })

        response(function(error){
                print error
        })

}
```

In Figure 8 one can notice communication from the perspective of the Wireshark packet catcher. In this example the application sends only one message. There are nine datagrams that contain seven TCP and two HTTP protocols.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	192.168.222.130	192.168.222.1	TCP	74	43264→2015 [SYN] Seq=0 Win=29200 Len=
2	0.000210000	192.168.222.1	192.168.222.130	TCP	74	2015→43264 [SYN, ACK] Seq=0 Ack=1 Win
3	0.000582000	192.168.222.130	192.168.222.1	TCP	66	43264→2015 [ACK] Seq=1 Ack=1 Win=2931
4	0.001915000	192.168.222.130	192.168.222.1	HTTP	134	GET / HTTP/1.1
5	0.050597000	192.168.222.1	192.168.222.130	TCP	66	2015→43264 [ACK] Seq=1 Ack=69 Win=665
6	1.477003000	192.168.222.1	192.168.222.130	HTTP	269	HTTP/1.1 200 OK (text/plain)
7	1.477158000	192.168.222.130	192.168.222.1	TCP	66	43264→2015 [ACK] Seq=69 Ack=204 Win=3
8	1.481278000	192.168.222.130	192.168.222.1	TCP	66	43264→2015 [FIN, ACK] Seq=69 Ack=204
9	1.481326000	192.168.222.1	192.168.222.130	TCP	66	2015→43264 [ACK] Seq=204 Ack=70 Win=6

No.	Info
1	43264→2015 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=4294941719 TSecr=0 WS=128
2	2015→43264 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1 TSval=392533 TSecr=4294941719
3	43264→2015 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=4294941719 TSecr=392533
4	GET / HTTP/1.1
5	2015→43264 [ACK] Seq=1 Ack=69 Win=66560 Len=0 TSval=392538 TSecr=4294941719
6	HTTP/1.1 200 OK (text/plain)
7	43264→2015 [ACK] Seq=69 Ack=204 Win=30336 Len=0 TSval=4294942088 TSecr=392681
8	43264→2015 [FIN, ACK] Seq=69 Ack=204 Win=30336 Len=0 TSval=4294942089 TSecr=392681
9	2015→43264 [ACK] Seq=204 Ack=70 Win=66560 Len=0 TSval=392681 TSecr=4294942089

Figure 8: Packets caught during one–message communication made by application

Node.js servers use, like all other web applications, the TCP protocol as the Transport Layer Protocol. The Node.js HTTP server is inherited from the Node.js TCP server. This state of affairs is not a coincidence. When users send data via the Internet, they want to be sure that the data will be delivered. The TCP protocol gives users this assurance. Three TCP segments are sent from the sender to the receiver, and they establish the connection. To have reliable communication there is a three–way handshake. After that the fourth segment is the HTTP request with the information. Then the sender transmits the TCP segment and a secret message hidden in the HTTP header. The last TCP segment informs the server and then the client about the end of the communication. An interesting fact is that the TCP segments have approximately 50 bytes, whereas HTTP messages are two to four times bigger. In this simulation the secret data was very short: four characters plus a flag.

4 Steganalysis tool for covert channel detection

The most practical method is to check whether there are any patterns or anomalies in the sizes of the protocol's messages. Since their introduction, machine learning techniques (formerly known as Artificial Intelligence - AI), have been successfully implemented. Among them we can distinguish [Mohd et al., 2016]:

- unsupervised learning, e.g., Support Vector Machine,
- supervised learning, e.g., Self Organizing Maps (SOM).

4.1 SOMs for anomaly detection

In our tests for anomaly detection we selected Self Organizing Maps (SOMs), which were developed by Kohonen [Kohonen, 1982]. They can be used also for classification of the given objects. Kohonen Network is a special example of neural networks that during learning do not need any kind of supervision and have been successfully implemented in many network anomaly detection system [Rhodes et al., 2000, Feyereisl and Aickelin, 2009].

4.2 Learning and testing of SOM

The basic part of SOM is a neuron. All neurons create a grid. The topology of the map is established by a neighborhood relation of the neurons. After all iterations, the grid gains its structure, i.e., contiguous neurons do not significantly stand out as their weight vectors are similar [Vesanto et al., 2000].

The network operation consists of two phases: learning and testing. The tests use the Matlab program and SOM Toolbox for Matlab 5 made by Juha Vesanto, Johan Himberg, Esa Alhoniemi and Juha Parhankangas [Vesanto et al., 2000].

In the learning phase, Kohonen Networks use clustering algorithms such as k–means. In each iteration, the random vector of the input data is selected and the distances between it and the SOM weight vectors are calculated. After finding the most similar neuron's weight vector to the input data (called *Best Matching Unit – BMU)*, the locations of all weight vectors are updated. They are moved towards the input vector. The map created by the tool presents a unified distance matrix, which consists of the distances between SOM units.

After each iteration, weight of the BMU is updated accordingly:

$$W_v(s+1) = W_v(s) + \Delta W_v(s+1) \tag{1}$$

$\Delta W_v(s+1)$ is the change in the unit's weight, given by formula (2):

$$\Delta W_v = \theta(u,v,s)\alpha(s)(D(t) - W_v(s)), \tag{2}$$

where:

- s - the current iteration,
- λ - the iteration limit,
- t - the index of the target input data vector in the input data set $\mathbf{D}$,
- $D(t)$ - a target input data vector,
- v - the index of the node in the map,
- W_v - the current weight vector of node v,
- u - the index of the best matching unit (BMU) in the map,
- $\theta(u,v,s)$ - a restraint due to distance from BMU, usually called the neighborhood function,
- $\alpha(s)$ - a learning restraint due to iteration progress.

A graphical representation of the matrix reveals the cluster build of SOM. In Figure 9 one can notice an orange dot that represents the location of an input data vector and a set of neurons with highlighted green BMU. The picture shows one iteration, when the whole set is being updated. A considerable advantage of these maps is to search for any kind of structure or pattern that an analyst could not identify.

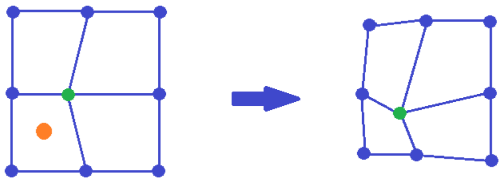


Figure 9: One iteration in SOM learning phase

5 Test results of SOM usage for anomaly detection

The results of these tests are devoted to finding any anomalies in the network traffic made by the application presented in Section 3. The simplest way to find something different in a big stream of data is to compare each stream's object.

5.1 Test data set

In the application, the communication is based on one type of Application Layer protocol – HTTP. It is important that the packets with edited headers, i.e., with secret data, do not stand out when compared with typical HTTP traffic. In the tests there will be 10 samples with 1000 HTTP protocols each. Samples come from:

1. browsing the Acer company webpage (A),
2. browsing the Cisco company webpage (C),
3. browsing the Dell company webpage (D),
4. browsing the Faculty of Electrical Engineering at Warsaw University of Technology webpage (E),
5. browsing the HP company webpage (H),
6. browsing the Intel company webpage (I),
7. browsing the Logitech company webpage (L),
8. browsing the Node.js framework webpage (N),
9. browsing the Oracle company webpage (O),
10. using our steganographic application with different settings(S).

5.2 Substring preparation

To check the usefulness of the application in tests, a long message will be sent. When one wants to send just a few short messages, this will not be as abnormal as sending a much bigger piece of text. In tests the server will send a 1000–character text from a file.

Listing 4 shows the algorithm that cuts a given text into random parts. In the *while* loop random variables are made, and they are drawn between variables named minimum and maximum. For every substring a new variable is drawn, which cuts out the next group of characters from the text. As long as i is smaller than the length, a new random variable is created. For security reasons, the *if* statement in line 7 checks whether the length of a substring is bigger than the whole text. Then in the message array substrings are saved. The counter i becomes the pointer to the last letter of the text saved in the message array.

Listing 4: Pseudocode of algorithm that cuts text from file into substrings

```
1  j <-0;
2  i <-0;
3
4  while (i < length){
5        random <- a random number between minimum and maximum
6
7        if (i+random > length)
8                random <- lenght -i;
9
10       message[j]<- substring of text between i and random
11       j<- j+1;
12       i<- i+ random;
13 }
```

5.3 Anomalies pattern detection in HTTP traffic

The tests will check the impact of the variables minimum and maximum for the detection of any patterns in HTTP messages made by the program. For every test a text consisting of 1000 characters was loaded to the steganographic application, and then the text was sent in parts of random length. During communication the packet sniffer was opened, and it recorded every packet. The presence of the HTTP protocol in each IP datagram was checked. Thereafter, for the analysis, 1000 HTTP protocols were taken. Then these packets' lengths were put into the matrix with nine other samples of 1000 HTTP protocols each, making the matrix 10x1001, where there were labels in the last column.

5.3.1 Sizes of HTTP messages in test

Table 1 contains the minimum and maximum values for each of the six tests. The first column, named *Message length*, shows the range of drawing a random length of the substring of the given 1000 character text. The next column presents the same numbers but as a percentages of the 1000 character text that is being sent. The third column shows the range of the ballast in every HTTP request. This kind of extra data was introduced due to the conclusions from the test presented in Subsection 5.3.2.

Table 1: Sizes of HTTP messages in tests

Message length	% of text	HTTP request ballast
1 10 to 200	1% to 20%	—
2 10 to 200	1% to 20%	10 to 20
3 10 to 200	1% to 20%	10 to 100
4 10 to 400	1% to 40%	10 to 200
5 100 to 500	10% to 50%	100 to 500
6 50 to 300	5% to 30%	50 to 300

Tests 1, 2 and 3 have the same range of substrings when sending HTTP responses. The difference between them is that in the first case extra data are not added to the receiver's side. In test number 2, one adds from 10 to 20 ballast signs to the HTTP request. The number of characters 10 and 20 stand for 1% and 2% of the 1000 characters text that is being sent through the HTTP response.

In tests 1 to 4 the range of random substrings of the text is huge. It means that a message (1000 characters in length) will be cut into only a few pieces. This special concept's aim is to result in disorder in the HTTP message sizes. Only test number 5 has a very big minimum variable for both sides.

From test 2 until the last one, random data will also be added to the HTTP request. On the receiver's side, the random variables in the range are also wide. This method has to hide the trend of identical HTTP messages made by the receiver's side of the program. Half of all HTTP messages made by the application are HTTP requests made by the client.

5.3.2 Test for random variables in range 1%–20% without additional characters

Figure 10 shows the result of the test. One can notice that each label stands for the 1000 sample set. Two labels are not visible. This means that for the SOM only eight samples are unique. Matlab shows only one label if the others are similar to it. That means that samples O and N are hidden. After removing visible labels, N and O show up. N is very similar to the A label, because it was in the same cell as A before and O in its turn is similar to the I label.

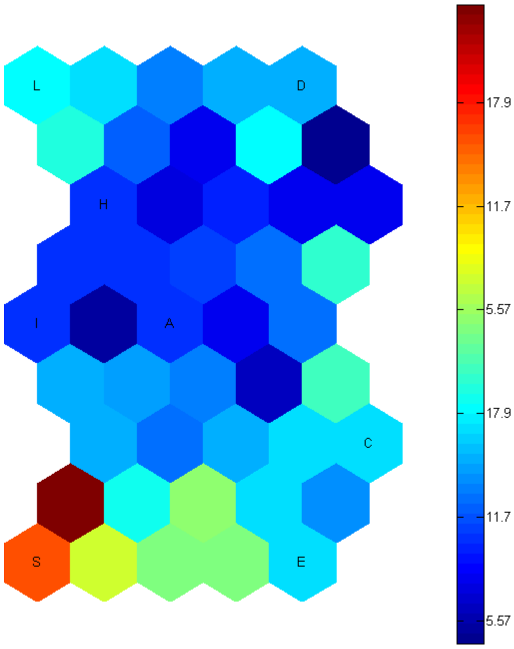


Figure 10: Test 1: Result of SOM for random variables in range 1%–20% without additional characters

The colored scale on the right side of the map stands for all possible subspecies in the data set. This test showed that steganographic packets are in a red subspecies only. All other HTTP messages are in blue subspecies. In addition, the S label is very far away in the diagram from the other labels, the majority of which are in the center of the SOM. This result means that steganographic messages do stand out from the rest of the samples.

The main reason for the fact that in test 1 steganographic packets were not like the other messages is the structure of the program. The covert channel operates on the HTTP request and response mechanism. In test 1 all HTTP responses from the server were differentiated. On the other hand, all HTTP requests from the client were equal to 99 bytes. To change this trend there should be added random strings to the HTTP request on the client's side.

5.3.3 Test for random variables in range 1%–20% with additional 10 to 20 characters

Figure 11 shows the results of the second test. One can notice that the S label is still in a different color subspecies. There is a slight difference between the results from the first test, and the S sample is located in a slightly lighter color. This means that steganographic HTTP messages do assimilate to the rest of the HTTP messages. The idea of adding random data to the HTTP request improved the program's detection by other applications.

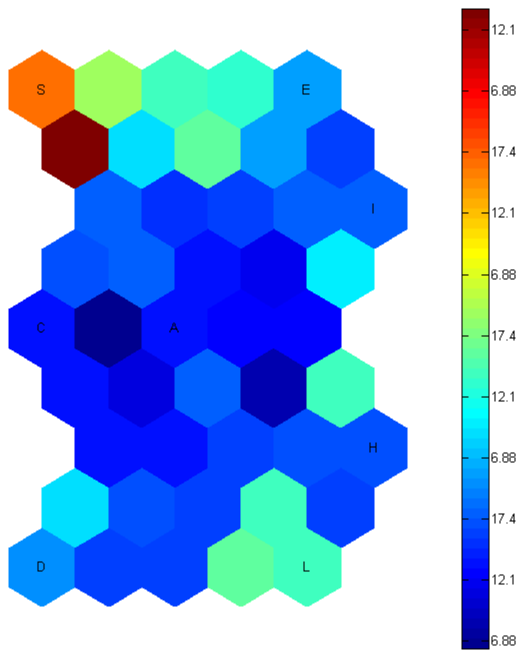


Figure 11: Test 2: Result of SOM for random variables in range 1%–20% with additional 10 to 20 characters

The graphic arrangement of the labels is also unquestionably crucial. Labels S, D, E and L are located at the corners of the map. The A and C labels are very close to the right in the center. There is not just one agglomeration of labels.

This tendency means that every sample of packets is a little different and the S label does not stand out in this comparison. As in the first test, label N is very similar to label A and O to I. That is why N and O are hidden.

5.3.4 Test for random variables in range 1%–20% with additional 10 to 100 characters

Figure 12 presents the results of the next test. The label symbolizing the packets of the steganographic application lies on an orange cell, like the D and H labels. This is a satisfactory result. S is still different from the majority of the samples but it is more similar to them than in the previous test. The interesting thing is that the label L is now an outlier. In addition, the I label lies in a more intensive colored cell than the steganographic samples. The SOM of the result of the test 3 has no labels in blue colors, which means the S label is much more similar to the whole group when compared to the previous tests. Only two labels, A and C, are light green.

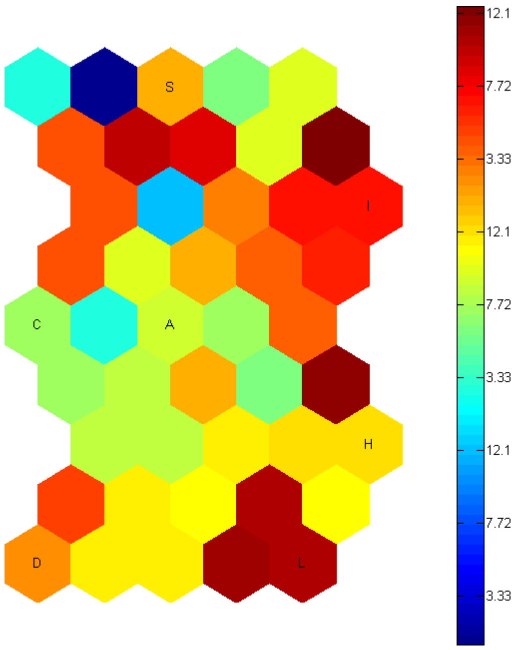


Figure 12: Test 3: Result of SOM for random variables in range 1%–20% with additional 10 to 100 characters

The SOM in Figure 12 is similar to the previous (Figure 11). There is neither an agglomeration nor small groups of samples. In this test label E is hidden, and after disabling all other labels, the E character shows up in the same place as the S label. This is a promising development in the HTTP message comparison tests. However, in this test the N label is still hidden under A and O in the same place as I.

5.3.5 Test for random variables in range 1%–40% with additional 10 to 200 characters

Figure 13 shows the SOM that is the result of the next test. There are no considerable differences between this result and the SOM from Figure 11. In this test the only difference from the previous one is the maximum range of random drawings for the sizes of the HTTP request and HTTP response. It seems that the changes in these variables do not affect the Kohonen's Map. This idea just rescaled the map. In the previous SOM, the S label was at the top while now it is at the map's bottom.

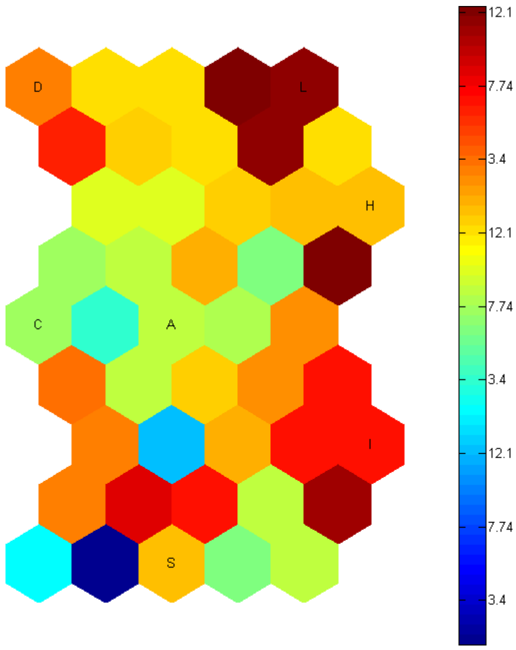


Figure 13: Test 4: Result of SOM for random variables in range 1%–40% with additional 10 to 200 characters

One can notice that still labels N and O are hidden under the same labels as before. The E label is under S. All labels are still separately located.

5.3.6 Test for random variables in range 10%–50% with additional 100 to 500 characters

Figure 14 shows the results of the next test. There is a rapid change in the colors of the subspecies. S is now in the sea–green color. The analysis shows that it is generally a very close subspecies to A and the hidden N and E. This test shows that the bigger range in the random variables for the client's HTTP request really camouflages the steganographic communication. In this SOM the HTTP messages from the L sample differ considerably from the rest.

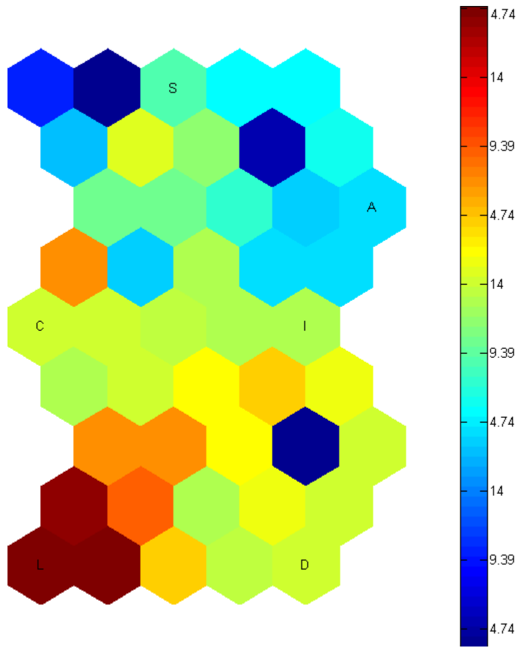


Figure 14: Test 5: Result of SOM for random variables in range 10%–50% with additional 100 to 500 characters

The labels in the SOM presented in Figure 14 are scattered. This tendency means that the samples differ from each other. This can be caused by the different servers they use or by the variety of data that each website has. The result presented in Figure 14 is satisfactory, because if normal website HTTP messages are not identical then the S label should also be a little different from the others.

5.3.7 Test for random variables in range 5%–30% with additional 50 to 300 characters

Figure 15 shows the results of the final test. In this test the range of random variables is wide but not as big as in the previous example. This moderation in choosing the range of the borders resulted in the best outcome. S is in the subspecies where the majority of labels are. Only one label, N, really differs from the others. At this point one can say that the steganographic messages are hidden well.

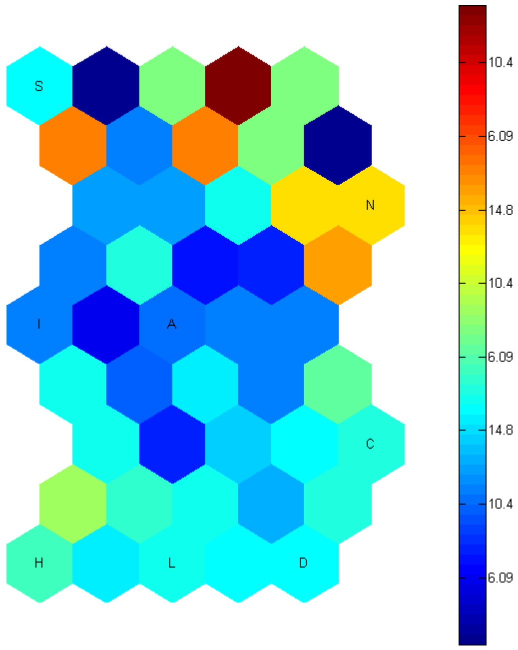


Figure 15: Test 6: Result of SOM for random variables in range 5%–30% with additional 50 to 300 characters

Like in the previous test, all the labels are scattered around the map. The E label is still very similar to the S label and also the O label is under I.

These tests show that when sending a longer text, approximately 1000 +/- 500 characters, one has to cut it into substrings of at least 5% to 30% of the text length. It is also important to add random data to the HTTP requests in the same random range as the server does.

5.4 Results analysis

It is also very important to hide the patterns in the message sizes as well as to check the average size of a packet. From the previous tests it follows that the average HTTP packets sizes were:

- while browsing the Acer company webpage (A): 684 bytes,
- while browsing the Cisco company webpage (C): 700 bytes,
- while browsing the Dell company webpage (D): 717 bytes,
- while browsing the Faculty of Electrical Engineering at Warsaw University of Technology webpage (E): 531 bytes,
- while browsing the HP company webpage (H): 712 bytes,
- while browsing the Intel company webpage (I): 687 bytes,
- while browsing the Logitech company webpage (L): 852 bytes,
- while browsing the Node.js framework webpage (N): 648 bytes,
- while browsing the Oracle company webpage (O): 715 bytes.

The range is from 531 bytes to 852 bytes. While, the average message sizes from the steganographic communication were:

- from the first test: 199 bytes,
- from the second test: 425 bytes,
- from the third test: 462 bytes,
- from the fourth test: 597 bytes,
- from the fifth test: 931 bytes,
- from the sixth test: 678 bytes.

In the aspect of average message size outcomes from tests 4 and 6 are located in the range of normal network traffic. In the test where SOM classified the samples, the best result was in the sixth test. The average HTTP length result firmly confirmed this. It means that the range 5% to 30% for random substrings and ballast in the HTTP requests is the best to hide the covert channel communication.

The results obtained are convincing evidence that SOMs are a very supportive steganalysis tool. The SOMs exposed the patterns in the sizes of the protocol headers in the initial version of the proposed program. They could identify any outstanding sequence in a set of data, furthermore they do not need supervised learning. A significant merit of visualizing these neural networks is the intuitive map, which could be examined by a non-expert.

Usually if a computer is connected to the Internet, it is open to exchange data. The SOM test revealed the order in sizes of the HTTP request. This order could be detected by special steganoanalysis programs. The idea of cutting a longer text into substrings worked. On the other hand, sending short texts or just a few messages does not require special camouflage.

6 Conclusion

In this paper we extended the staganographic method [Graniszewski et al., 2016] by adding a part dedicated to steganalysis of the generated traffic. For anomalies detection we used SOMs, and, therefore, we named the framework SOMSteg. The steganographic part uses the HTTP header field for hiding the data transfer. The proposed Covert Channel utilizes the idea of hiding data in the *Trailer* field. This method is classified as steganographic *Add Redundancy Pattern*. We developed a special test environment written in the JavaScript language. The developed program is a two channel steganographic communicator. It works based on the HTTP Node.js server. This server has many helpful features. In each turn of the communication not only the HTTP but also the TCP packets are sent. This technique tones down the accumulation of the HTTP messages, which are the data carriers.

A series of steganoanalysis tests were performed to determine the program's communication noticeability by third parties. Profund analysis of the HTTP messages by SOM led to a hardening of the camouflagic algorithm.

The conducted tests showed that without any changes the HTTP messages may be relatively easy to recognize as outliers. By manipulating the sizes of the strings in the HTTP header field one can hide the steganography technique, which will be not so easily detected by anomalies detection, like the SOM.

According to [Wendzel et al., 2015] *Add Redundancy Pattern* has a few countermeasures. Traffic normalizers can restore the protocol's headers to the default values. More complex traffic normalizers focus on batches of protocols, so long text sent by our program in a few HTTP messages could be detected. Machine learning and statistical approaches could relatively easily discover the covered channel based on the *Add Redundancy Pattern*; nevertheless, we applied SOM to harden the developed application.

6.1 Future work

Intrusion Detection Systems (IDS) are utilized to secure network devices, and some of them are enriched with Deep Packet Inspections (DPI). DPI investigates application traffic, i.e., protocols that are being sent, e.g., Application Control Engine (ACE) [Cisco, 2007]. Examining their impact on our program could be a compelling approach for the future work.

Another interesting topic would be to extend our system and test it with HTTP/2. Dmitrova and Mileva [Dimitrova and Mileva, 2017] have already studied some possibilities of covert channels in this protocol, which could be a good starting point for an extension of our framework. Although HTTP/2 is mainly utilized with TLS as HTTPS it can also be used without encryption.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable comments.

References

[Barth, 2011] Barth, A. (2011). Http state management mechanism. RFC 6265, RFC Editor. `http://www.rfc-editor.org/rfc/rfc6265.txt`.

[Baxter, 2017] Baxter, A. (2017). Did russia really ban vpns? `https://www.expressvpn.com/blog/did-russia-ban-vpns/`.

[Bender et al., 1996] Bender, W., Gruhl, D., Morimoto, N., and Lu, A. (1996). Techniques for data hiding. *IBM Syst. J.*, 35(3-4):313–336.

[Blasco et al., 2012] Blasco, J., Hernandez-Castro, J. C., de Fuentes, J. M., and Ramos, B. (2012). A framework for avoiding steganography usage over http. *Journal of Network and Computer Application*, 35(1):491–501.

[Booch et al., 1999] Booch, G., Rumbaugh, J., and Jacobson, I. (1999). *The Unified Modeling Language User Guide.* Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

[Braden, 1989] Braden, R. (1989). Requirements for internet hosts - communication layers. STD 3, RFC Editor. `http://www.rfc-editor.org/rfc/rfc1122.txt`.

[Cisco, 2007] Cisco (2007). *Cisco 4700 Series Application Control Engine Appliance Security Configuration Guide.*

[Conway, 2003] Conway, M. (2003). Code wars: Steganography, signals intelligence, and terrorism. *Knowledge, Technology & Policy*, 16(2):45–62.

[Dimitrova and Mileva, 2017] Dimitrova, B. and Mileva, A. (2017). Steganography of Hypertext Transfer Protocol Version 2 (HTTP/2). *Journal of Computer and Communications*, 5.

[Dyatlov and Castro, 2003] Dyatlov, A. and Castro, S. (2003). Exploitation of data streams authorized by a network access control system for arbitrary data transfers: tunneling and covert channels over the http protocol. Technical report, http://www.gray-world.net.

[ExpressVPN, 2017] ExpressVPN (2017). Apple removes vpn apps from china app store. `https://www.expressvpn.com/blog/china-ios-app-store-removes-vpns/`.

[Feamster et al., 2002] Feamster, N., Balazinska, M., Harfst, G., Balakrishnan, H., and Karger, D. (2002). Infranet: Circumventing web censorship and surveillance. In *Proceedings of the 11th USENIX Security Symposium*, pages 247–262, Berkeley, CA, USA. USENIX Association.

[Feyereisl and Aickelin, 2009] Feyereisl, J. and Aickelin, U. (2009). Self-organising maps in computer security. *CoRR*, abs/1608.01668.

[Fielding and Reschke, 2014] Fielding, R. and Reschke, J. (2014). Hypertext transfer protocol (http/1.1): Message syntax and routing. RFC 7230, RFC Editor. `http://www.rfc-editor.org/rfc/rfc7230.txt`.

[Fielding et al., 1999] Fielding, R. T., Gettys, J., Mogul, J. C., Nielsen, H. F., Masinter, L., Leach, P. J., and Berners-Lee, T. (1999). Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor. `http://www.rfc-editor.org/rfc/rfc2616.txt`.

[Graniszewski et al., 2016] Graniszewski, W., Krupski, J., and Szczypiorski, K. (2016). The covert channel over http protocol. In *Proc. SPIE 10031, Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2016*, volume 10031, pages 100314Z–100314Z–8.

[Handel and Sandford, 1996] Handel, T. G. and Sandford, II, M. T. (1996). Hiding data in the osi network model. In *Proceedings of the First International Workshop on Information Hiding*, pages 23–38, London, UK, UK. Springer-Verlag.

[Haynes, 2017] Haynes, J. (2017). Backdoor access to WhatsApp? Rudd's call suggests a hazy grasp of encryption. `https://www.theguardian.com/technology/2017/mar/27/amber-rudd-call-backdoor-access-hazy-grasp-encryption`.

[Horenbeeck, 2006] Horenbeeck, M. V. (2006). Deception on the network: Thinking differently about covert channels. In *in: Proceedings of 7th Australian Information Warfare and Security Conference*.

[Kachhia-Patel, 2010] Kachhia-Patel, A. (2010). United States of America vs Anna Chapman and Mikhail Semenko. `https://www.justice.gov/sites/default/files/opa/legacy/2010/06/28/062810complaint1.pdf`.

[Kahn, 1996] Kahn, D. (1996). The history of steganography. In *Proceedings of the First International Workshop on Information Hiding*, pages 1–5, London, UK, UK. Springer-Verlag.

[Kohonen, 1982] Kohonen, T. (1982). Self-organizing formation of topologically feature maps. *Biological Cybernetics*, 43(1):59–69.

[Lamping et al., 2014] Lamping, U., Sharpe, R., and Warnicke, E. (2014). *Wireshark User's Guide: For Wireshark 2.1.*

[Lampson, 1973] Lampson, B. W. (1973). A note on the confinement problem. *Commun. ACM*, 16(10):613–615.

[Lubacz et al., 2014] Lubacz, J., Mazurczyk, W., and Szczypiorski, K. (2014). Principles and overview of network steganography. *IEEE Communication Magazine*.

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

[Mileva and Panajotov, 2014] Mileva, A. and Panajotov, B. (2014). Covert channels in tcp/ip protocol stack - extended version-. *Central European Journal of Computer Science*.

[Mohd et al., 2016] Mohd, R. Z., Zuhairi, M. F., Shadil, A. Z., and Dao, H. (2016). Anomaly-based nids: A review of machine learning methods on malware detection. *2016 International Conference on Information and Communication Technology (ICICTM)*, pages 266–270.

[Petitcolas et al., 1999] Petitcolas, F. A. P., Anderson, R. J., and Kuhn, M. G. (1999). Information hiding - a survey. *Proceedings of the IEEE, special issue on protection of multimedia content*, pages 1062–1078.

[Rhodes et al., 2000] Rhodes, B. C., Mahaffey, J. A., and Cannady, J. D. (2000). Multiple self-organizing maps for intrusion detection.

[Schear et al., 2006] Schear, N., Kintana, C., Zhang, Q., and Vahdat, A. (2006). Glavlit: Preventing Exfiltration at Wire Speed. In *HotNets-V*. ACM SIGCOMM.

[Schwartzbeck, 1997] Schwartzbeck, M. (1997). The evolution of US Government Restrictions on Using and Exporting Encryption Technologies (U). `https://www.cia.gov/library/readingroom/docs/DOC\_0006231614.pdf`.

[Shen et al., 2018] Shen, Y., Yang, W., and Huang, L. (2018). Concealed in web surfing: Behavior-based covert channels in http. *Journal of Network and Computer Applications*, 101:83 – 95.

[Simmons, 1984] Simmons, G. J. (1984). *The Prisoners' Problem and the Subliminal Channel*, pages 51–67. Springer US, Boston, MA.

[Steganography, 2016] Steganography (2016). *Oxforddictionaries.com*. Oxford University Press.

[Surhone et al., 2010] Surhone, L. M., Tennoe, M. T., and Henssonow, S. F. (2010). *Node.Js*. Betascript Publishing, Mauritius.

[Szoldra, 2016] Szoldra, P. (2016). Isis' favorite messaging app may be in jeopardy. http://www.businessinsider.com/russia-anti-encryption-telegram-2016-6?IR=T.

[United States Department of Defense, 1985] United States Department of Defense (1985). Trusted Computer System Evaluation Criteria (Orange Book). Technical report, United States Department of Defense. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[United States Department of State, 1996] United States Department of State (1996). International traffic in arms regulations (itar) (22 cfr 120-130). http://www.dtic.mil/dtic/tr/fulltext/u2/a312382.pdf.

[Vesanto et al., 2000] Vesanto, J., Himberg, J., Alhoniemi, E., and Parhankangas, J. (2000). Som toolbox for matlab 5. Technical report, Helsinki University of Technology.

[Wakefield, 2015] Wakefield, J. (2015). Can the government ban encryption? http://www.bbc.com/news/technology-30794953.

[Wang et al., 2017] Wang, Y., Shen, Y., Jiao, X., Zhang, T., Si, X., Salem, A., and Liu, J. (2017). Exploiting content delivery networks for covert channel communications. *Computer Communications*, 99:84 – 92.

[Wendzel et al., 2016] Wendzel, S., Mazurczyk, W., and Zander, S. (2016). Unified description for network information hiding methods. 22(11):1456–1486. http://www.jucs.org/jucs_22_11/unified_description_for_network.

[Wendzel et al., 2015] Wendzel, S., Zander, S., Fechner, B., and Herdin, C. (2015). Pattern-based survey and categorization of network covert channel techniques. *ACM Comput. Surv.*, 47(3):50:1–50:26. https://dl.acm.org/citation.cfm?id=2684195.

[Zander et al., 2007] Zander, S., Armitage, G. J., and Branch, P. (2007). A survey of covert channels and countermeasures in computer network protocols. *IEEE Communications Surveys and Tutorials*, 9(1-4):44–57.