SAPIENZA
UNIVERSITÀ DI ROMA

Multi-Channel Exfiltration Techniques

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Dipartimento di Informatica
Corso di laurea in Networks & Security

Mahdi Akil
Matricola 1720976

Relatore
Prof. Luigi V. Mancini

Correlatore
Prof. Daniele Venturi

A/A 2017/2018

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Prof. Luigi Mancini. The door to Prof. Mancini office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right direction whenever he thought I needed it.

I would also like to acknowledge Prof. Daniele Venturi as the second reader of this thesis, and I am forever grateful for his very valuable comments on this thesis.

Contents

List of Figures

List of Tables

1 INTRODUCTION

Data exfiltration, data extrusion or data exportation is the unauthorized, illegal and unapproved removal of data from a computer or a network. We have two types of data exfiltration, it could be manual where the attacker has physical access to the computer or it could be automated by a malicious program running over the network. As it requires the transfer of data inside and outside a company's network, it frequently mimics typical network traffic, allowing valuable data loss events to go unnoticed until data exfiltration has already been completed. And when your company's most precious information is in the hands of hackers, the damage is limitless.

There are more data exfiltration ways than there are roads to Rome. but still, the most sufficient way to exfiltrate data is by using covert channels.

Lampson first described covert channels (Lampson, 1973) as "Channels not intended for information transfer at all". Since then the definition of covert channels has been developing. In 1993 Virgil Gligor defined them (V. Gligor, 1993) as "A communication channel that allows a process to transfer information in a manner that violates the systems secu-

rity policy". While the most recent definition was given in 2010 by Eric Couture (Couture, 2010), he described a covert channel as "A mechanism for sending information without the knowledge of the network administrator or other users". To sum all this up we can describe covert channels as "A Communication technique that is used to transfer information in a secretive and unauthorized manner. Hence, it's simply an extra way for information to leave a network".

Data Exfiltration using a covert channel can be expressed as a bag with a secret section that a spy is using to slip a weapon past security guards into or out of a guarded building. An attacker can use covert channels to transmit sensitive documents unobserved. In our case, rather than bypassing security guards we need to bypass network security standards to implement a covert channel. And just as a spy can use that same secret section to sneak a weapon from security guards when entering a guarded building, an attacker can use a covert channel to conceal a cyber weapon. For example, download a malware from an external server into the victim machine within an organization's private network.

1.1 Impact of Data Exfiltration

It is a known fact that stealing consumers data can create problems but in a way this only affects individuals so the

damage is limited, while stealing data from corporations and governments can lead to more disastrous effects. As stealing information such as trading secretes or marketing strategies from a company and sharing these information with a rival company might cause a considerable financial loss for the victim company and possibly lead to its closing. While this has huge impact its no where close to the impact of stealing from governmental targets, the consequences could be huge not only financially but this could lead to threatening the nations safety. The catastrophic effects data exfiltration could have on a nation can be observed by what Edwin Snowden has caused. (von Solms & van Heerden, 2015).

1.2 Data Exfiltration operation

There are certain steps that an attacker must go through to successfully exfiltrate data. The attacker must first breach the private network by bypassing the Intrusion Detection Systems (IDS), then gain access to a host machine, and then he should not have a big trouble installing a malware that will read from a directory where the sensitive files are located and this malware will send the data into the outside server that is controlled by the hacker.

1.3 Covert Channels

Secrete forms of communication have been used throughout the history to deliver messages from one side to another, beginning with tattooed slave scalps and wax tablets to hiding information in the packets headers. Covert channels are a special class of secrete communication which seeks to deny that a communication is taking place between two hosts over the internet. Network covert channels pose a significant security concerns and challenges. Since nowadays covert channels are not only used to leak sensitive information to unauthorized people but they are also used to implement malicious code to the victims machines.

1.3.1 Covert Channels Structure

The structure of a covert channel specifies how the channel is constructed and what does it do to hide and carry information. The hiding mechanism is what differentiates each covert channel from one another. Therefore, the structure often influences covert channel characteristics such as capacity, detection, prevention etc.

1.3.2 Classification of Covert Channels

The most common way to classify covert channels was proposed by (Brown, Yuan, Johnson, & Lutz, 2010) which classifies the channels into:

1. Storage.
2. Timing.
3. Behavioural.

Storage based covert channels hide data in the spatial representation of the shared medium. Storage covert channels encode data in a medium that is shared between the end−points, where data is often not expected to be present. An example of this would be the use of reserved fields in various packet headers/footers to conceal the data.

Timing based covert channels manipulate the delays of the communication stream between two hosts to encode hidden data instead of directly modifying the data stream by inserting data in the stream itself. An advantage of a timing covert channel is that it is connected to the actual content of the communication stream.

Behavioural based covert channels are defined as communication channels that are resultant from observing the

characteristics, actions and other behavioural heuristic of a given subject. As these channels can exist in inter−Automated Information System (AIS) that can be implemented over any packet switched communication. The idea is that the encoding of secrete data stream by modifying the behaviour of the packet sizes generated and observing the behaviour at the opposite end of the channel (Yuan & Lutz, 2005).

1.4 Motivation & Significance of the work

We hope that this work will encourage the cybersecurity community to pay greater attention to the security of the network and especially to the traffic that is leaving the network and not only the one that is arriving in order to provide a better and more robust techniques to prevent data exfiltration.

This thesis shows that by using multiple covert channels which work in parallel, we can create new efficient techniques to exfiltrate data that are hard to detect and prevent using the conventional data exfiltration prevention techniques.

1.5 Research Objectives & Contributions

Man In The Middle attacks (MITM) are a common type of cybersecurity attacks that allow attackers to eavesdrop on

the communication between two targets. The attack takes place in between two legitimately communicating hosts, allowing the attackers to "listen" to or "disrupt" a conversation they should normally not be able to listen to, hence the name "Man-In-The-Middle."

The same attack could be applied while exfiltrating data using covert channels. Assume that Alice (Attacker) is trying to Ex-filtrate sensitive information from a company (Victim) to her server, she managed to gain access to a computer owned by Bob who is an employee in that company. Furthermore, Alice was able to install a malware on Bob's computer. This enabled Alice to open a covert channel between her server and Bob's computer which is a malicious computer now in the company. Using the covert channel Alice now can exfiltrate data from Bob's computer or she can use the malicious computer to access other computers in the company and exfiltrate more data. Eve (Watcher) is the network administrator in the company. As covert channels are hard to detect and see, to protect the network Eve can act as a MITM by adding noise to random channels. This will lead to the disruption of the information on those channels. So, if Alice is using a covert channel and it happens that Eve is adding noise to that channel, in this case Alice will receive the data she's exfiltrating plus the noise that Eve added and

then she will never be able to decode the data to have the correct files.
Note that Eve will not be able to add noise to all the channels since some channels could be used legally by the employees.

To solve this problem, we came up with a new technique that Alice could use to successfully exfiltrate the data even if Eve is adding noise to the channels.
Instead of using one covert channel, with our technique Alice could use n covert channels that will work in parallel to exfiltrate the data. We used some cryptographic techniques that enabled us to differentiate between authentic (not noisy) and non authentic (noisy) data. With this new technique, it's enough to have only one good channel and Alice will be able to receive the complete authentic data.

To measure the efficiency of this technique we introduced the concept of round complexity. A round is calculated every time we send n packets over the n channels. To prove the efficiency we were able to send a $200MB$ file in 92 rounds over 30 channels.

1.6 General objective of the thesis

This work shows that it is possible for an attacker to implement multiple exfiltration covert channels using the same

port or different ports on the victims' machine to ex−filtrate data. The attacker will be able to use these channels in parallel to ex−filtrate data to his server. And in the case where a portion of channels are detected and the victim is trying to corrupt the data by adding noise to the channels our protocol will 100% guarantee the integrity and the authenticity of the information.

1.7 Dissertation Outline

In this dissertation, we first provide an introduction on the latest data exfiltration techniques in Chapter 2. In Chapter 3, we provide the security model and properties of our proposed protocols. After that in chapter 4 we introduce our innovative protocols that use multiple channels working in parallel in order to steal information from the victims' machine. Moreover, in chapter 5 we present the Mathematical Logic behind our protocols. In chapter 6 we provide the experiments that were done to prove the efficiency of the proposed protocols. And finally we present our conclusions in chapter 7 along with the insight on future directions that could be followed to continue this work.

2 Related Work

Network security is one of the most crucial and frequently neglected aspect in a computer network. A network security breach could impact all the actors in that network, from consumers, companies, and as well as governments. For consumers, networks could include their online identities leaving them victims of attackers who are interested in stealing information such as, credit card numbers, passwords, and other data that can cause a tremendous amount of damage if it was in the wrong hands. For companies and corporations, the damage could be much worse from losing sensitive data to another competitor. Furthermore, a weak network security could influence governments, as attackers could gain access to confidential military documents as well as to the financial records which could be used against that government itself.
Network attacks are expected and are more common than you think. And it is obvious now that a new course of action must be taken to prevent them.

Most of the cybersecurity research that has been, or is being done direct its attention on preventing attackers from breaching into the network. And the most commonly practiced method for that is the Intrusion Detection System

(IDS) (Liu & Kuhn, 2010). The goal of IDS is to give attackers several security layers to beat before breaching the network. Of course, this defensive tool is quite necessary but it is even more essential to look at other lines of defence as well. Since these protective methods cannot examine all network traffic without costly hardware or high overhead that will result in network slowdowns. Plus, in reality, these defensive tools can never absolutely protect the network as the most eager and enthusiastic hackers will not stop trying at the sign of a powerful defence especially if they are trying to breach a military or governmental network that will give them a great advantage in the times of war and political activities. Hence, eventually, they will find a way around these tools to breach the network. Based on the above, it is not only important to look into the incoming traffic for abnormalities that could be an evidence of network breach but also to examine the outgoing traffic for the effect of a network breach. Promptly, in order to earn from a network breach, a hacker must be able to find a way to transfer data from the attacked machine to his server and this is known as Data Exfiltration. Some tools do exist to block sensitive data from leaving the network. These tools are referred to as Data Leak Prevention (DLP) tools. However, these tools are targeted and are an expression or keyword–based algorithms, which an intruder can easily overcome.

In the following sections we will describe existing covert channel techniques and mechanisms and since there are countless number of methods for data exfiltration, we will describe some of the most used and feasible methods.

2.1 Data Exfiltration Methods

2.1.1 Exfiltration using File Transfer Protocol

File transfer will use the FTP protocol (Postel & Reynolds, 1985) which relies on Transmission Control Protocol (TCP) (Morandin, 2010). In order to transfer data to a server which will be running FTP. An authentication step first must be done followed by the transfer of the file and finally the closing of the connection. Now, in order to exfiltrate data using FTP from an infected machine, the data (files) usually must be encapsulated in a binary formate (possibly Encrypted). The authentication details (Username and password) could be static or dynamically given to the malicious program by the attacker. The data will be sent over a TCP channel, and the port that will be used can be a non standard port specified by the attacker. (Van Antwerp, 2011)

Now, in order for the attacker to receive the data he must configure his FTP server and provide the username and password on the victim machine. The attacker may configure the server to accept connections only during the time of exfiltration and this may reduce the probability of detection.

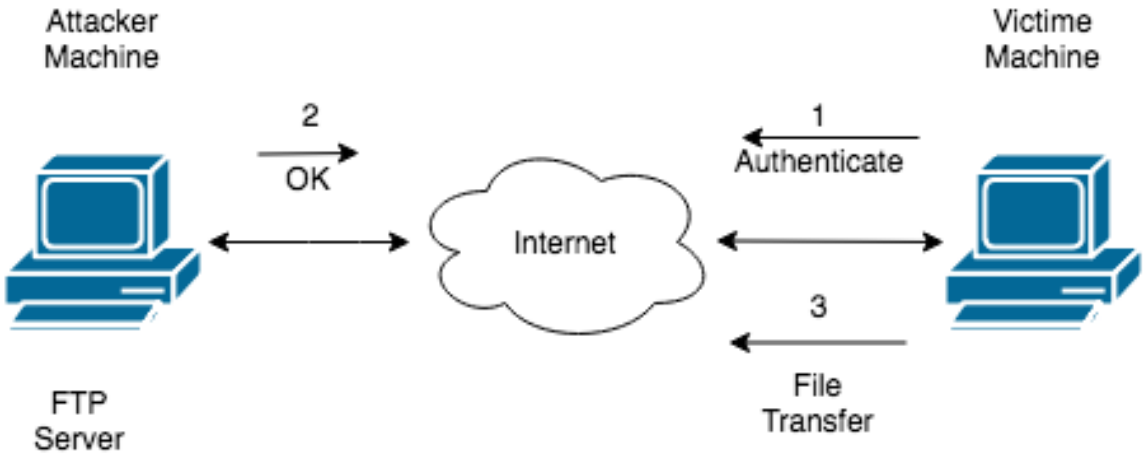

Figure 1: Exfiltration using File Transfer Protocol (FTP).

The best way to detect if FTP is being used for exfiltration, packet headers of FTP traffic could be examined to check the destination server and then compare it with the servers that are in the whitelist. For a demonstration on this technique refer to (Fig. 1).

2.1.2 Exfiltration using Secure Copy

The secure copy protocol (SCP) uses the Remote Cope Protocol (RCP) in combination with SSH in order to provide a copy method that is encrypted and authenticated. RCP uses a TCP channel provided by an SSH tunnel that allows the transfer of files. SSH provided encryption of the entire channel, hiding the internal message that is being transferred and therefore limiting the detection algorithms. Now, to use SCP for data exfiltration, the client must initiate an

SSH session to a remote server along with the username and the password combination (Van Antwerp, 2011).Once the server authenticates the provided username and password the transfer of the file will start using the RCP and at the end the connection is closed when the transfer of the file is closed. In order for the server to receive the file it must properly configure an SSL service with the corresponding username and password. And as with the FTP the attacker here also have the chance to use a non standard port to avoid suspicion. The way to detect if SCP is being used for data exfiltration, is also by examining packet headers for SSH authentication attempts to non−trusted servers through the use of a whitelist also. For a demonstration on this technique refer to (Fig. 2).

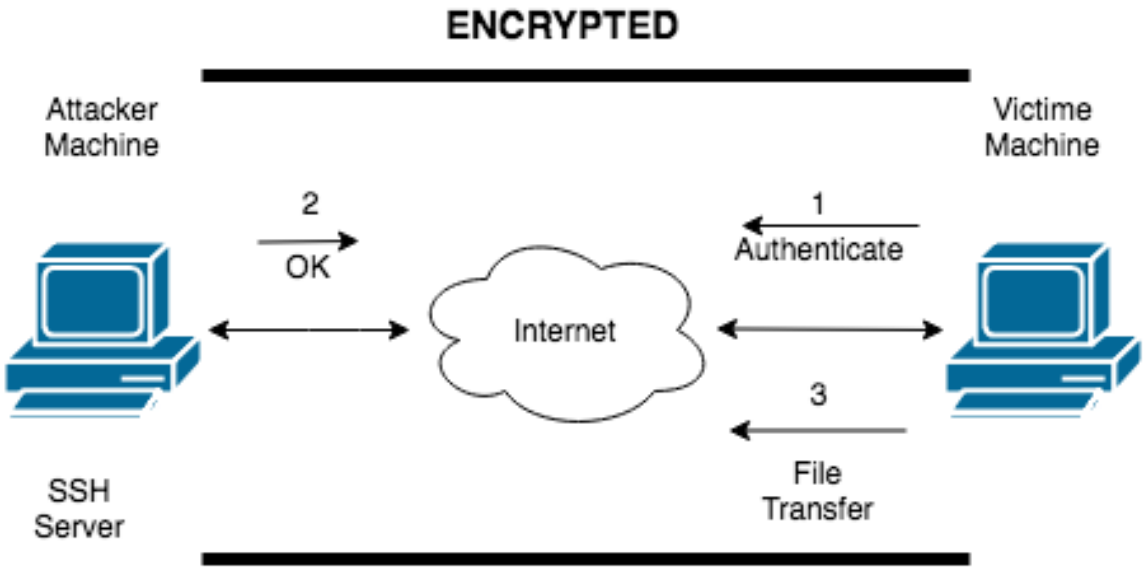

Figure 2: Exfiltration using Secure Copy Protocol (SCP).

2.1.3 Exfiltration using HTTP POST

HTTP POST method is a method consists of an HTML object that contains a form and a POST action. This method uses the Hyper Text Transport Protocol "HTTP" (Fielding & Reschke, 2014). The POST action will send the specified text in the web form to the server specified by the HTML. This method is unencrypted and originally was not intended for large amount of data. Later, modifications were made to the method to accommodate the transfer of file uploads to a web server. Now, to exfiltrate data using the HTTP POST multiple POST requests might be needed to satisfy the transfer of large files (Van Antwerp, 2011). The POST requests are needed to be split up in order to behave like normal POST requests. Note that, the attacker must consider that the majority of the web applications that accept file uploads have a limit usually few megabytes. To receive the exfiltrated data a properly configured HTTP server should be able to handle the expected POST requests. Typically, POST method requires the client to request an HTML object from a server, followed by the client submitting a POST request. But when exfiltrating data, the first step will probably be skipped to minimise unnecessary traffic and this could be used as a flag to detect irregular traffic. For a demonstra-

tion on this technique refer to (Fig. 3).

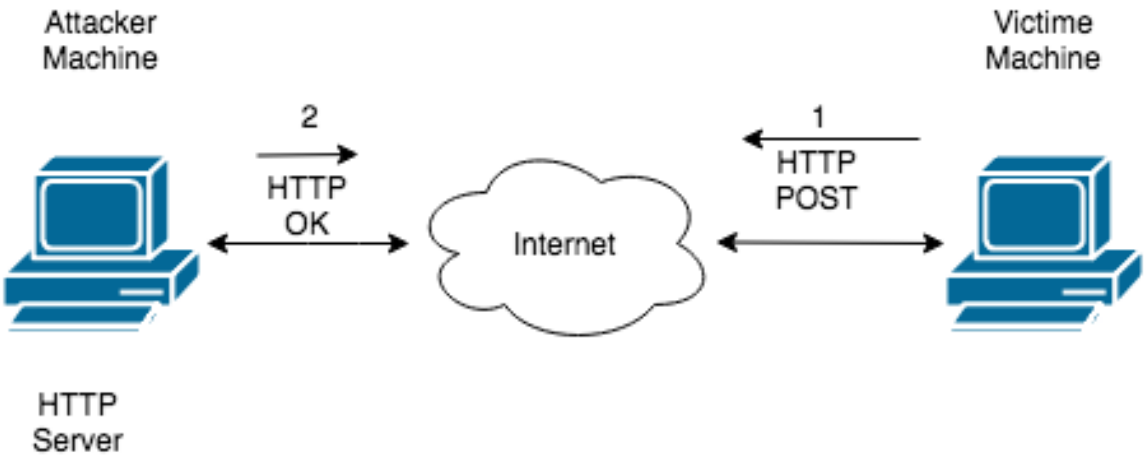

Figure 3: Exfiltration using HTTP POST.

2.1.4 Exfiltration using SSH Tunnel

An SSH tunnel will work by encrypting a channel using SSH session. Traffic will be tunnelled to the SSH host through the use of a local proxy. The data channel will be encrypted and closed when finished. This way will make it possible for any underlying protocol to be used through an encrypted channel which makes it very appealing to the attackers. Now, to exfiltrate data using SSH Tunnel (Van Antwerp, 2011), the first step to be followed is to create an SSH session with the remote host. SSH needs a secure hand-shake which can be coded using sockets. A proxy will then be initiated over SSH and the traffic will be sent over the specified port. And just like with "Secure Copy", the only requirements for accept-

ing this type of data would be a properly configured SSH service. There will be two key issues that will arouse suspicious activity in this case, the first issue will be the connection to a remote SSH server, which a normal user will only do rarely. Therefore, a whitelist can be used and any connection to outside this list will notify the admin. The other issue will be the use of a local proxy. So any packets sent to 127.0.0.1 will hint at the existence of a local proxy and therefore could notify the administrator as well. For a demonstration on this technique refer to (Fig. 4).

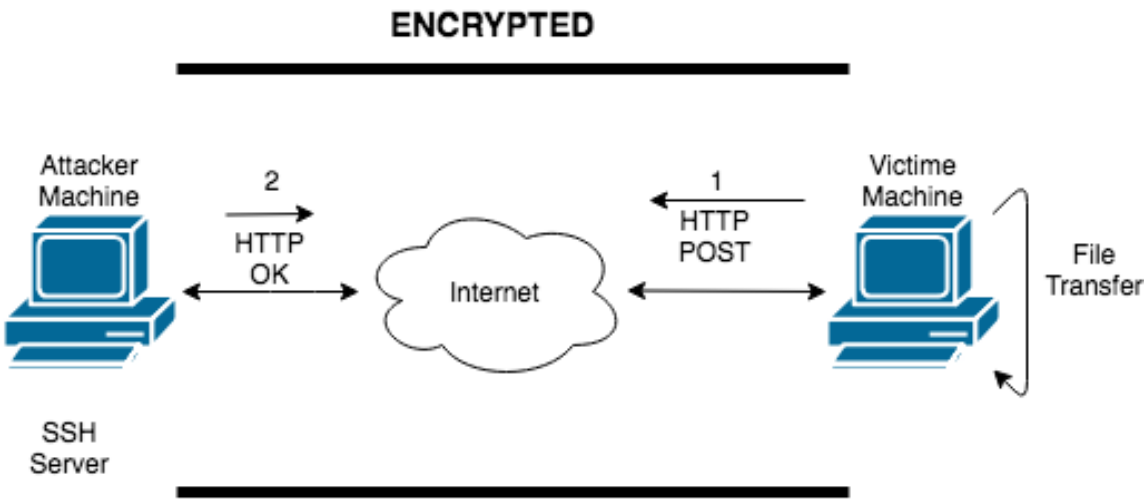

Figure 4: Exfiltration using SSH Tunnel.

2.1.5 Exfiltration using Peer−2−Peer

Peer−2−Peer (P2P) traffic can be described as using a centrally located server for each client to communicate with

each other. Once the clients find each other, they will initiate a direct connection and communicate accordingly. If a user is actively using P2P applications, specifically legitimate ones such as those used for software distribution, there will be more traffic and therefore this traffic could be used to hide the data. Now, to exfiltrate using P2P the client must contact a third party server (Van Antwerp, 2011). This third party server can be an Internet Relay Chat (IRC) server or writable web server, for example. Once the clients have communicated, they can open a communication channel and transfer data.

In this technique, the server will be almost the same as the

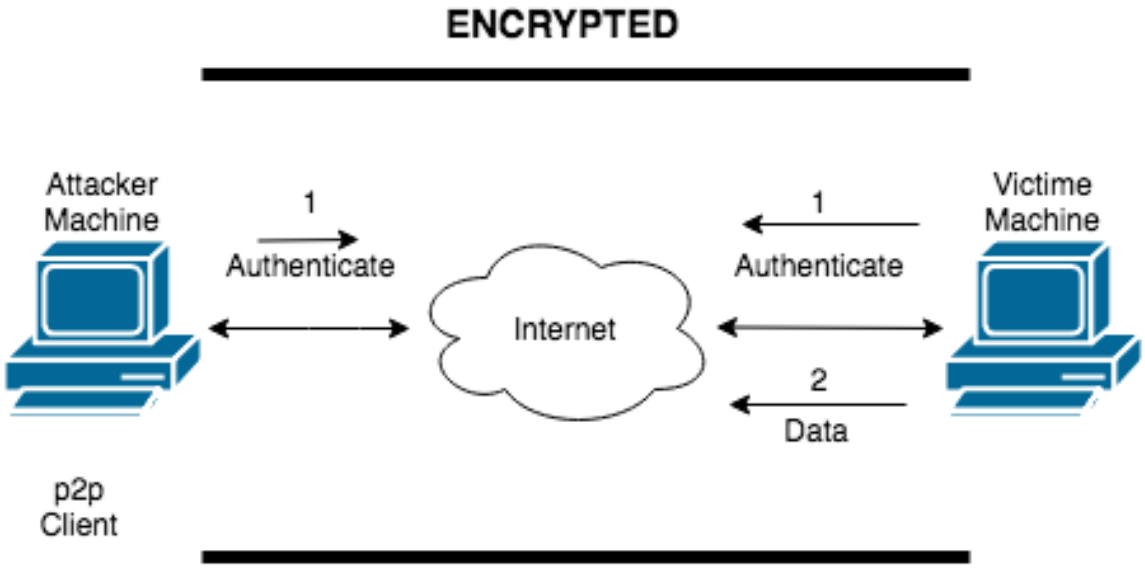

Figure 5: Exfiltration using Peer-2-Peer.

client. The server will probably have open ports so that the

sending side can initiate connections, bypassing firewalls in the process. This technique could be easily detected in case there is no other P2P traffic, nevertheless in the presence of P2P activity, choosing the P2P activity is suspicious would be extremely difficult. For a demonstration on this technique refer to (Fig. 5).

2.1.6 Exfiltration using ICMP

An Internet Control Message Protocol (ICMP) echo request packet is often used by a user to determine if a remote host is reachable. The host receiving the echo request packet will respond with an echo response with the packet payload being identical to the packet payload received. If the client does not wish to receive a reply, the exfiltration could be more covert by crafting the echo request packet to have an incorrect source address. One of the overlooked details about this transaction is that the contents of the ping packet are rarely inspected as they are largely irrelevant to determining if a host is available or not. For this reason, the packet payload can contain data to be exfiltrated and the receiving host can simply record the payloads of all ICMP echo request packets it receives. To Exfiltrate data using this technique, raw sockets can be used to implement an ICMP echo request. There is some flexibility in terms of setting the payload size. Standard echo request packets are

56 bytes, however they can be larger (Fig. 7). Making the packet size larger will likely make this method easier to detect. In this case, the attacker must balance the packet size against transfer bandwidth so as to not get detected. Detection Algorithms could be developed to inspect ICMP packets for keywords or regular expressions. For a demonstration on this technique refer to (Fig. 6).

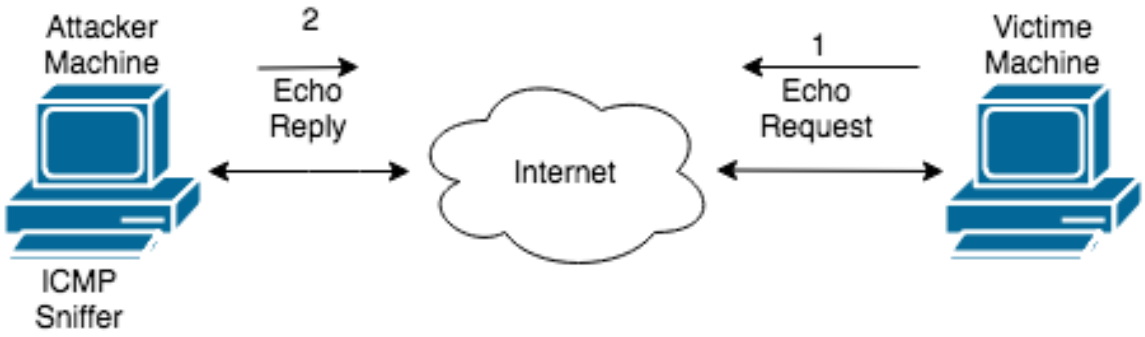

Figure 6: Exfiltration using ICMP Echo Request packets

	Bits 0–7	Bits 8–15	Bits 16–23	Bits 24–31
Header (20 bytes)	Version/IHL	Type of service	Length	
	Identification		*flags* and *offset*	
	Time To Live (TTL)	Protocol	Header Checksum	
	Source IP address			
	Destination IP address			
ICMP Header (8 bytes)	Type of message	Code	Checksum	
	Header Data			
ICMP Payload (*optional*)	Payload Data			

Figure 7: ICMP packets Diagram

2.1.7 Exfiltration using TCP

The Transmission Control Protocol (TCP) can be used to simply transmit messages to and from a server. To send a file, a server could wait for a client to connect on a certain port and assume that the transmission is finished (The file has been completely sent) when the client closes the connection. Now, to exfiltrate data using TCP (Schear, Kintana, Zhang, & Vahdat, 2006), sockets could be used to transmit the exfiltrated data, in order to ensure that the data is sent correctly "Acknowledgment" messages (ACKs) could be used. Due to the enormous amount of data that uses TCP this could be one of the most covertly methods. The detection of this exfiltration technique is slightly difficult due to the vast amount of other TCP traffic that exists in the network.

2.1.8 Exfiltration using UDP

Like TCP, the User Datagram Protocol (UDP) could be used to send data also. But unlike TCP, UDP doesn't have any feature to ensure the reliability of the channel. The client−server should account for duplicate packets, missing packets and as well as out of order packets. Now, to exfiltrate data using UDP (Van Antwerp, 2011), sockets could be used to send UDP packets. The application must be properly coded in order to ensure that the data is received com-

pletely and in the desired order, it is highly recommended that the server should be designed in a way to ensure that the file is completely transferred correctly due to the unreliability of the UDP packets. Detecting UDP is a little bit easier that TCP since services that use UDP are far less than the services that use TCP and therefore the examination of all the UDP traffic may be possible.

2.2 Covert Channels Techniques

Frequently, it is believed that the adoption of encryption is adequate to ensure the safety of the message. Though, encryption only restricts unapproved individuals from decoding the message. Whereas in many circumstances, the presence of communication or variations in communication patterns, such as an increased message rate, is sufficient to raise doubt and initiate alerts. the best way to bypass these problems is by using covert channels. The main goal of covert channels is to intend to hide the presence of the communication. Typically, covert channels use means of communication not commonly designed to be used for communication, making them quite baffling.

Lampson originated the concept of covert channels (Lampson, 1973) in the meaning of unified systems as a mechanism by which a process at a high−security level leaks information

to a process at a low–security level that would otherwise not have access to it. While covert channels are a dangerous threat even for single hosts, their potential in computer networks is greatly extended. In computer networks, overt channels such as network protocols are used as transports for covert channels.

The powers of covert channels in computer networks have considerably been enhanced because of modern high–speed network technologies (Zander & Armitage, 2008), and this course is expected to advance. If we consider that only one bit per packet can be transmitted using a covert channel, an attacker with a high–speed connection will be able to leak approximately 26 GB of data per year.

Steganography is the technique of hiding information in the audio, video, or text (Febryan, Purboyo, & Saputra, 2017). Steganography requires some kind of content that will be used as a means of hiding information, while covert channels need network protocols as carriers. The fact that using an internet protocol is the only way to transfer information over the networks makes covert channels broadly available. And this gives them the advantage in circumstances where steganography cannot be employed.

Common employment of covert channels are of a malicious or rejected nature, and hence posing a severe threat

to network security (Wendzel & Mazurczyk, 2016). Moreover, the usage of network covert channels has increased because of the strict rules that are followed by important organizations against the use of open channels, specifically the physical channels such as printing a large number of papers during a short period of time or the carrying of memory sticks in or out of the organization. Grasping the knowledge of existing covert channel methods is essential in generating countermeasures. The discovery, removal, and detecting the capacity of covert channels are challenging. But to ensure the safety of computer networks, these challenges need to be addressed.
This section presents a summary of existing covert channels in network protocols.

2.3 The Prisoner Problem

The prisoner problem was first introduced by (Simmons, 1984) and is the actual model for covert channel communication. Consider Two people, Alice and Bob, that are in a prison and are planning to escape. In order to make an escape strategy they need to communicate first, but Eve the warden monitors all their messages. If Eve finds any signs of unusual messages, she will put Alice and Bob in solitary jail, making it impossible for them to escape. Alice and Bob

must exchange harmless messages carrying hidden information that hopefully Eve will not notice. Consider the following three types of wardens:

1. A passive warden can only watch the channel, but cannot modify any messages.
2. An active warden who is able to insignificantly alter the messages, but without changing the meaning.
3. A malicious warden who adjusts the messages and converts the entire meaning.

Applying this scenario to computer networks, Alice and Bob use two networked computers for communication. They communicate using a harmless overt channel between their computers and covert channel is hidden inside the harmless overt channel that they are using. Assume that Alice and Bob use a different overt channel (e.g, A meeting in a coffee shop) to share a secret code which will be beneficial to encode the messages that are being sent using the covert channel. Keep in mind that it is possible for Alice and Bob to be one person, maybe a hacker who is trying to exfiltrate sensitive data from a company that he gained access to before. Eve manages the network and can monitor the passing traffic for overt channels or alter the passing traffic to eliminate or disrupt any chance of covert channels. Figure 8 below describes the model (Alice sending to Bob in the presence of

Eve).

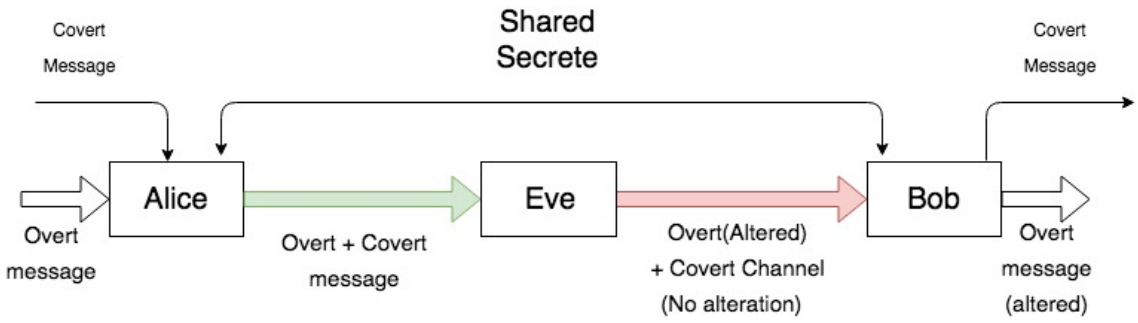

Figure 8: communication using overt and covert channels

2.4 Communication Situations

It is possible to use covert communication in many situations depending on whether Alice and Bob are using a harmless channel and they are trying to hide information in that channel or if they are acting as Man In The Middle (MITM) in a connection that is being used by two normal users and they are trying to manipulate that channel to covertly exchange information between them.
Considering that the overt sender is the same person who wishes to send information using a covert channel then he will be able to manipulate the overt channel as he aspires to hide a covert channel in there. But to have better secrecy, the sender can operate as a Man In The Middle by embedding a covert channel into an existing overt channel that is being used by an honest user.

The covert receiver could be the same person who is receiving via the overt channel, but to enhance stealth the receiver can also be a Man In The Middle deriving the covert information from an overt communication designed for an honest user.

2.5 Internet Protocols Suits

It is essential to get a grasp on the structure of network protocols, to better understand how covert channels really work. The major building blocks of the data transmission structure are the network protocols in any network environment.

Communication protocols that serve as an underlying model for network communications are included in the term Internet Protocol Suite (IPS) (Faloutsos, Faloutsos, & Faloutsos, 1999). The modifying of the IPS is the fundamental direction for network hiding techniques. As the two major protocols that are included in the IPS design are the TCP and the IP then it is common to refer to IPS as the TCP/IP. Note that, IPS uses a quadruple-layered protocol stack design to delineate between component levels of transmitted data. Next designs of internet protocol architecture, such as the Open Systems Interconnection (OSI) Model, further portray component layers within the standard four TCP/IP

layers.
IPS is described in (Fig. 9) and it includes:

- Link Layer.
- Internet Layer.
- Transport Layer.
- Application Layer

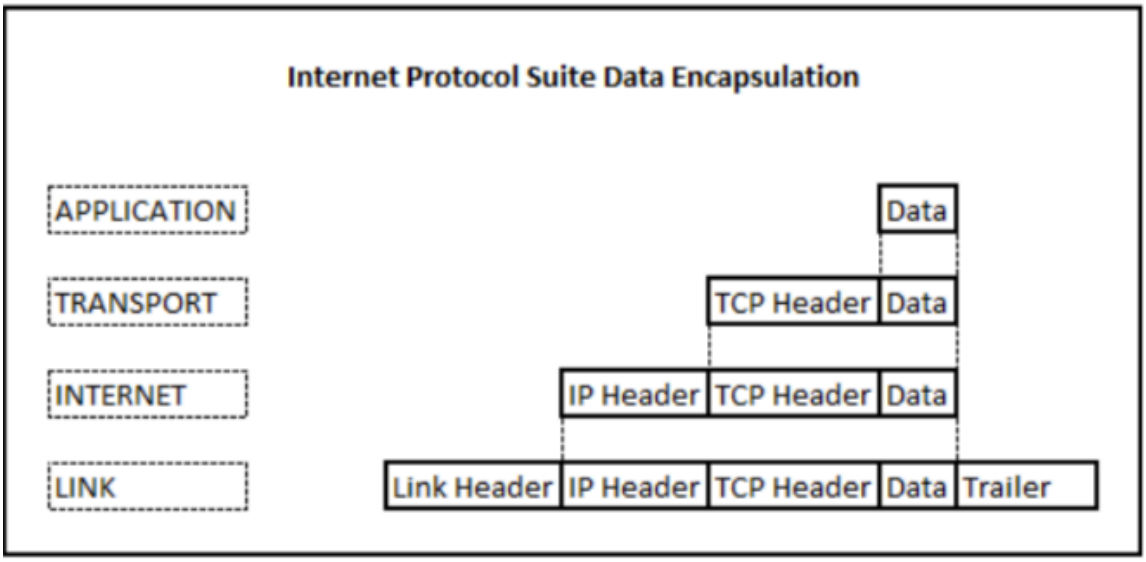

Figure 9: IPS Data Encapsulation

1. *Link Layer:* Is necessary to define the topology of the local network as it contains the needed protocols for that. It is the first and considered as the most basic layer in the IPS protocol design. Examples of link layer protocols used to define the type of local network transmission include:

- Ethernet
- The IEEE 802 Local Area Network standard.

2. *Internet Layer:* This is the second layer in the IPS protocol stack design. It is a group of inter–networking methods, protocols, and specifications in the Internet Protocol Suite that are used to transport datagrams (packets) from the originating host across network boundaries to the destination host specified by an IP address. Its function is facilitating internetworking, which is the concept of connecting multiple networks with each other through gateways. Examples of internet layer protocols used to define network to network communication include the addressing mechanisms IPv4 and IPv6.

3. *Transport Layer:* This is the third layer in the IPS protocol stack design and contains the protocols necessary to open and maintain connections to the appropriate application or process on the target host. The best known transport protocol is the Transmission Control Protocol (TCP). It is used for connection–oriented transmissions, whereas the connectionless User Datagram Protocol (UDP) is used for simpler messaging transmissions. TCP is the more complex protocol, due to its stateful design incorporating reliable transmission and data stream services.

4. *Application Layer:* This is the fourth layer in the IPS protocol stack design and includes the protocols required for process−to−process communications over networked devices. Examples of Application Layer protocols include Hypertext Transfer Protocol (HTTP) for web traffic, File Transfer Protocol (FTP) for a host to host file transfers, and the cryptographic protocols Secure Shell (SSH) and Secure Sockets Layer (SSL).

A packet is completed when all the data is appropriately encapsulated into the IPS protocol design. A packet can be simply defined as a formatted data unit that is able of traveling across the network. Keep in mind that forming a complete packet does not really require all the four layers. The application layer, for example, is not required by many types of packets. Furthermore, methods of network steganography depend on several modes of packet manipulation, but may or may not need all layers of the protocol stack design.

Most of the next timing and storage channel procedures adopt the IPv4 explicitly. IPv4 is the fourth generation of the Internet Protocol addressing system; It uses a $32-$bit address commonly identified by the dotted decimal address format, such as the $172.16.1.1$ address. Every component of the four blocks contains numbers normally selected be-

tween 1 and 255, which translates to 8 bits, or 1 byte, per component number. As IPv4 only contains a 32-bit address, the total number of unique addresses is limited to 2^{32}; computationally speaking, the number of possible IPv4 addresses has been outrun by the arrival of personal network devices that each requires a unique IP address. Although techniques have been exercised to briefly decrease the IPv4 address deficiency, a good solution is found in the IPv6 addressing scheme. IPv6 is the sixth generation of the Internet Protocol currently in early stages of unified approval. Given the size of network−connected devices worldwide, IPv6 attempts to resolve the lack of unique network addresses by assigning a hexadecimal $128-$bit address; This development of addressing describes a space of 2^{128} unique addresses. The resulting construction change will demand adapted network steganographic techniques for those covert channels relying specifically on IPv4 architecture to change also.

Note: Channel capacity, and bandwidth, differ depending on the kind of the channel implemented. In storage channels, bandwidth is decided by the number of bits or byte modifications made inside the manipulated network protocol. In timing channels, bandwidth is restrained by the amount of data crossing the network that is influenced by the scheme of manipulation.

When originally proposed by TCSEC (Trusted Computer System Evaluation Criteria) (V. D. Gligor, 1994), the Department of Defense declared that from a security viewpoint, covert channels with low bandwidth represent a lower threat than those with high bandwidth. When defining data transfer rates, faster speeds or higher bandwidth may seem acceptable. Nevertheless, several intrusion or extrusion detection systems used over a target network may have rule–based detection algorithms intended to prevent questionable data leaks. If Alice (Attacker) chooses to apply a covert channel with remarkably fast transfer rates or higher bandwidth, she risks exposure or undesired attention from Eve (Watcher). In the case of overall efficiency, it would, therefore, be recognized more efficient to transfer data at slower and less noticeable speeds; while individual communication transfer may be slower, the communicating parties stand to gain persistence of use of a particular channel and thus increase overall efficiency.

2.6 Covert Channels Classification

2.6.1 Storage Channels

Storage channel communication designs provide a process to alter transmitted network protocols to covertly transmit data. For Alice and Bob to communicate throughout storage channels, they need to agree on a communication

scheme. In a network environment, this can be achieved by a diversity of means including straight adjustment of acknowledgment frames, header extensions, and padding, or unused data fields contained within a protocol. By construction, storage channels influence communication by transmitting the message itself within the bounds of the altered protocol. Because of naturally higher capacity than timing channels, this increases the detection probability and therefore, it makes storage channels the less useful covert channels. However, the smoothness of creating storage channels makes them an impressive method of covert information hiding, especially in a poorly monitored network environment. As storage channels depend on straight packet adjustment and injection designed to carry a covert message, they are acknowledged to have an aggressive impact generated on the flow of digital traffic within the network(Wang & Lee, 2005).

2.6.2 Timing Channels

Timing channel communication methods depending on the inflection of system resources. For Alice and Bob to communicate over timing channels, they must first agree on a communication scheme. In a network environment, this can be fulfilled by a diversity of methods including modification of packet transmission timing, packet volume, or

packet ordering. By structure, timing channels carry data by packet transmission rates. It is consequently, essential for the sender and receiver to agree on timing parameters, and oftentimes perform synchronization, in order to achieve communication. Because of naturally lower capacity than storage channels, and the corresponding increase in detection difficulty, timing channels are the most effective class of techniques.

Timing channels perhaps can be more classified as active or passive depending on the sort of influence they yield on the flow of digital traffic within the network. Active timing channels need the injection of packets to covertly transmit information, while passive timing channels temper with present packet traffic. Active packet injection is naturally faster but higher bandwidth is not regularly rewarded. The complexity of detecting passive packet modulation makes passive timing channels a conceivably more desired way.

While implementing storage channels may appear straightforward, the implementations of timing channels is complicated and can encounter depraved performance due to accidental transmission delays or intentional interception techniques. Mark that the efficiency of timing channels is subjected to network conditions, handling abilities over the channel, algorithm complexity, and coding portability. Network jam during peak hours of use may cause defective packet

reordering, packet loss, or alterations in round−trip time between packet transmissions and replies. Likewise, both the sender and receiver must lessen the external impact on system resources to counter processing obstructions. Lastly, the sender and receiver should select adequate communication algorithms and assure that coding functionality is not influenced by differences in system environments. Recognizing these limitations have a double goal: software developers can promptly build stronger robust code, but network administrators can likewise implement targeted procedures to increase timing irregularities over their networks (Wang & Lee, 2005).

2.7 Covert Channels Methods

In this section, we will present a survey of existing covert channel techniques that could be used in data exfiltration.

2.7.1 Reserved Header Bits

By exploiting protocols, such as TCP/IP, it is possible to encode a covert channel using reserved or unused bits in the packet headers. This technique is especially used if the receiver does not check the standard values of the packet, then hidden data could be transmitted covertly (e.g. in type of service field of the IP header)($Fig.10$).

Keep in mind that to carry covert data any unused header

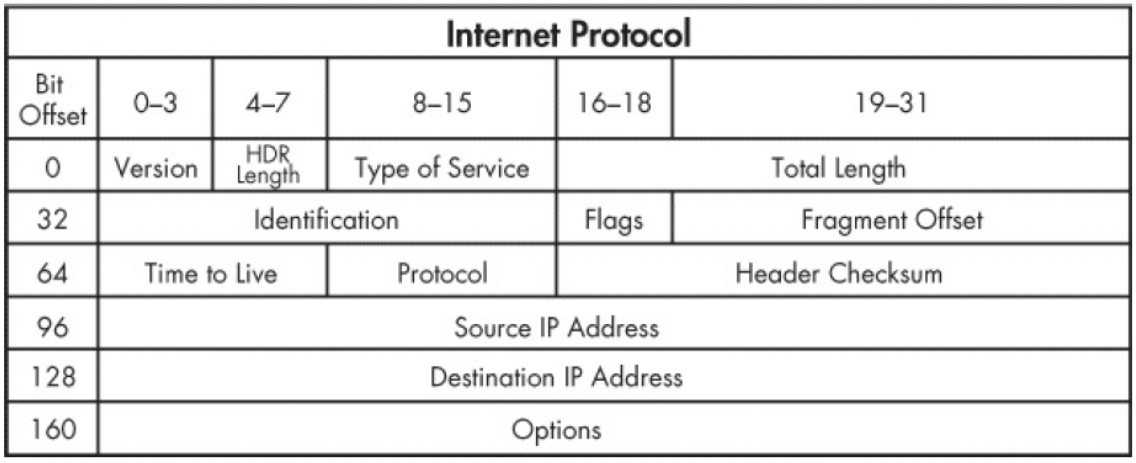

Figure 10: The Ip header structure

field could be used. For example, inside the IP header field, the Don't Fragment "DF" bit could be used to carry hidden information (Fig. 10).
Moreover, padding bits present a way to encode covert data also (Fig. 10). For example, Ethernet frames must be padded to a minimum length of 60 bytes. But if the protocol standard does not enforce specific values for the padding, then any data can be encoded there to be covertly transferred (Zander, Armitage, & Branch, 2007a).

2.7.2 Optional Header Fields

Regular headers can be extended by several protocols. These extensions are usually used to carry non−mandatory or not predicted data in the initial designation when needed, and thus increasing the abilities of the designed protocol. Us-

ing existing or new header extensions, covert data could be encoded. For example, covert data can be hidden as IP addresses in IP route record option headers. Another approach is to use the presence or absence of optional header fields as a covert channel, in a way where the presence of a specific header field could be translated as 1 while the absence of that field could be translated to 0 by the receiver(Zander, Armitage, & Branch, 2007b).

2.7.3 Semantic Overloading Of Header Fields

By modifying the semantics of the overt channel a covert channel could be implemented. For example, by modifying the TCP sequence numbers it is possible to encode covert data, the sequence number is normally used to coordinate which data has been transmitted and received (Fig.11). The initial sequence number (ISN) is the first sequence number that is selected by the client. The covert sender must be careful in choosing the ISN of the new packets so that they won't overlap with the sequence numbers of the previous packets. Based on this by directly encoding covert data in the most significant byte of each ISN and setting the last three remaining bytes to zero a very modest covert channel can be constructed. (Zander et al., 2007b).

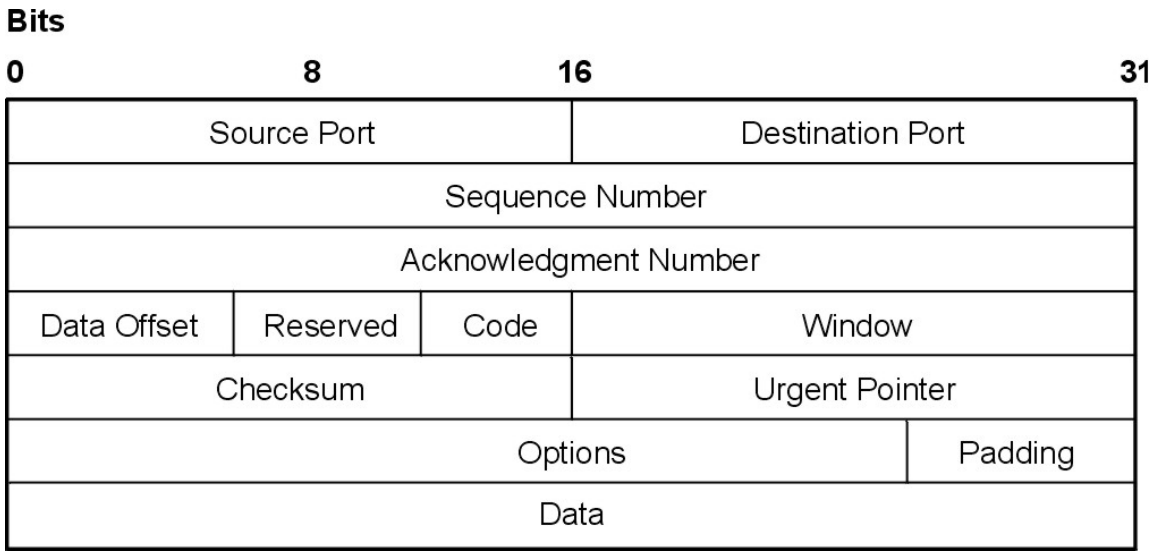

Figure 11: The TCP header structure

Although these previous encoding schemes are simple and can be easily implemented, they are easily detected because they are encoded in the header fields of the packets. Therefore, simple algorithms could detect that there is a defect with the header fields if they are carrying covert data. Ergo, more complicated schemes are proposed to make detection harder.

To have more encoding options, covertly sending data using protocols that work on higher layers, especially those that are text–based are introduced. For instance, by changing the use of lower or upper case letters or by varying the number of spaces between words. The Hypertext Transfer Protocol (HTTP) could be exploited to hide data in its header fields, more ways could be used to exploit the HTTP pro-

tocol.

2.7.4 Modulating Header Fields

Using subsequent transmissions by tampering with the order of the valid destination addresses or by encoding data in the destination address fields covert data can be hidden. Destination IP address or UDP/TCP destination port numbers can be good examples of usable address fields. In the case where the source address can be changed, it will be possible to hide covert information there as well.
Header extensions, length of headers and messages are fields that can be found in most TCP and IP communication protocols. By changing the length of the message or the header length it will be possible to send covert information. Keep in mind that instead of changing the lengths freely or without any rules a more experienced covert sender could choose from a set of lengths to resemble sensible messages.
Moreover, using the low order bits of timestamp fields, covert information could be embedded. For instance, using the TCP timestamp header options, covert data can be encoded, note that this proposed method slows down the TCP stream so the timestamps are valid on packets when they are sent. To achieve this algorithm, for every TCP timestamp produced by the system the least significant bit (LSB) will be compared with the bit to be sent covertly. If the (LSB) and

the covert bits are equal then the packet will be immediately sent and thus encoding a (0) otherwise we will have a small delay in the packet and therefore encoding a (1). (Murdoch & Lewis, 2005).

2.7.5 Message And Packet Sequence Timing

Another way of varying interpacket times is varying the packet rates which could be also used to carry covert information. This can be implemented when the sender alters its packet rates at each time interval. The receiver then has to measure the rates of the time intervals of the packets and then decode the covert information. A special channel which is called on/off channel could be used where one rate is zero and other rates are one and this makes it easier to decode the information. In order to be able to use the packet timing channels, a synchronization procedure for time intervals must be followed by both the sender and the receiver.

Nevertheless, an alternative way exists that does not require synchronization since the covert information is encoded in the interpacket delays of consecutive packets. This covert channel makes it possible to leak information from a high−security host to a low-security host in the timing of acknowledgments necessary for reliable communication.

The timing of protocol operations can be modulated also

in a similar way of packet timing. For example, a receiving station can recognize each message individually or wait till two messages arrive before recognizing the first.(Zander et al., 2007b)

2.7.6 Packet And Field Ordering

Packet reordering could be used to implement a covert channel. Consider having n packets then these packets can be arranged in n ways, then a maximum of $\log_2 n!$ bits can be transmitted at a time. In order to determine the original order of the packets, a pre−packet sequence number is required. And this already included in a number of protocols, including TCP.
The same technique can be applied to reorder header fields in protocols where the order of the header fields is not fixed. For example, covert information can be encoded in the order of HTTP header fields. (Zander et al., 2007b)

3 Security Model

3.1 Interactive Encryption & Message Authentication

Public Key Encryption (PKE) and Public Key Message Authentication (PKMA) schemes are two of the most significant and well studied cryptographic primitives. Traditionally, both notions are defined as non-interactive (i.e., used

for single messages). Both the sender and receiver need not keep any state, such non-interactive "one−and−done" philosophy is also crucial in many applications. Joined with the fact that we now have many efficient candidates for both signature and encryption schemes, it might appear that there is limited benefit in prolonging the syntax of signature/encryption schemes to allow for possibly interactive realizations.
The biggest disadvantage of non-interactive signatures is the fact that there is no interaction between the sender and the receiver. Hence, this works for a single message only which is very limited. To avoid this, a new scheme was proposed by (Dodis & Fiore, 2014) that assumes the presence of interaction between the sender and the receiver which is logical especially in the case where there is a conversation. This new scheme is "Interactive Encryption and Message Authentication" (Dodis & Fiore, 2014)

This interactive scheme offers some advanced security properties which are impossible to be achieved with the non-interactive schemes, Deniability and Forward Security.

- ***Deniability:*** A Public Key Message Authentication protocol is deniable if the sender S can authenticate a message m to the receiver R in such a way that R cannot use the transcript of their conversation as evidence to later convince third parties about the fact that S took part in

the protocol and authenticated m.

- ***Forward Security:*** Forward security guarantees that any leak of secret information at some time t should not affect the security of protocol runs that occurred in the past, i.e., at any time t' < t. Forward security is a desirable property that is not known to be achieved by standard non-interactive public key encryption.

We consider a problem that consists of an attacker "Alice" who already has breached a network and gained access to a machine controlled by a victim "Bob". Alice intends to exfiltrate data from Bob's machine to her machine via multiple n channels. We assume that there is a watcher "Eve " who can read all the n channels and modify b out of n channels, where $b \leq (n-1)$.

To define our interactive security Model, we start by describing a ***Measured Matrix*** consisting of two matrices.

- ***Round Complexity:*** Which is the number of iterations that we should do to send p packets over n channels.
- ***Communication Complexity:*** Which is how many bits we transmit with respect to the size of the file that is intended to be transferred.

3.2 Security Properties

We define the security properties, ***completeness***, and ***unforgeability***.

- ***Completeness:*** Is the property which guarantees that Alice will receive the complete file that she is trying to exfiltrate from Bob's machine.
- ***Unforgeability:*** Is the property that ensures that it's not possible for the adversary to forge a packet and send it to Alice as if it were coming from Bob. In other words, ***Unforgeability*** guarantees that Bob will not be fooled by the adversary.
- ***Authenticity:*** Is the property which guarantees that Alice will receive messages from Bob only.

The watcher Eve changes the content of the communication channels with various objectives that include:

1. Deleting a packet so that Bob never receives it. Thus, preventing communication between Alice and Bob, censorship;
2. Inserting a new packet invented by Eve and send it to Alice wishing she will consider it authentic, forgeability;
3. Changing the order of packets sent through the channels.

3.3 Protocol Proposal

The proposed protocol is an innovative exfiltration technique that uses n covert channels to exfiltrate files from a controlled machine to a controlled server. The aim of this protocol is to guarantee that the attacker will receive complete and unforgeable data that he is trying to exfiltrate even in the presence of a Man In The Middle who's trying to modify or corrupt the data.
This proposed technique is different from other exfiltration techniques since we don't send all of our bandwidth over 1 channel which could raise alarms and notify system administrators. Instead, we divide the bandwidth over n channels. That means, over each channel, a small amount of bandwidth will be sent and that could aid in covertness. Moreover, by using n channels we avoid having a single point of failure.

The figures below $(12, 13)$ show the difference in bandwidth distribution between our protocol and the normal protocols that use a single channel.
We can notice that by using n channels we can clearly send more packets/second e.g if we're able to send 1 packet/second then by using a single channel to send 8 packets we need 8 seconds however with 4 channels we will be able to send $(\frac{8}{n})$ packets/second. So it would take us 2 seconds to send the

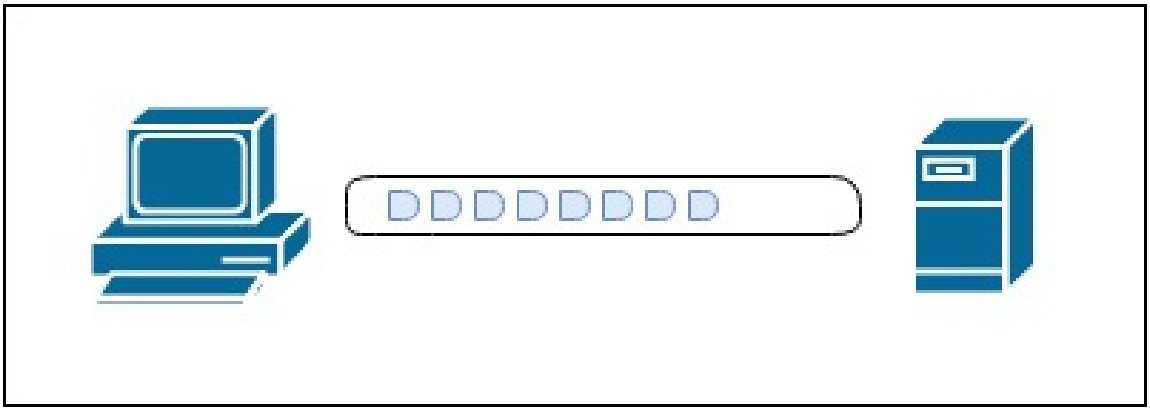

Figure 12: bandwidth distribution with a single channel

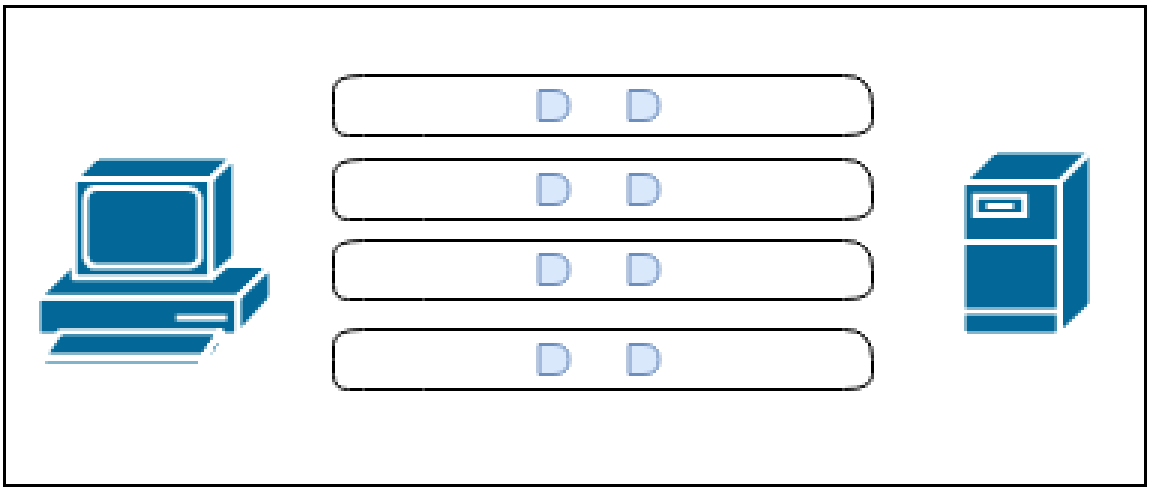

Figure 13: bandwidth distribution with multiple channels

same amount of packets.

4 Protocols

We define two protocols which calculate the number of rounds and the time it takes Alice to exfiltrate a file from Bob's machine that she illegally controls, using n covert chan-

nels while we assume that Eve can read all the n channels but can modify b out of these n channels where $b \leq (n-1)$.

From here on we will refer to Alice as the Server, Bob as Client and Eve as watcher.

Round: Each iteration over the available n channels is defined as a round.

Through out this protocol, to guarantee data authentication and integrity we used HMAC which is the abbreviation for "key-hashed Message Authentication Code". HMAC is a particular type of "Message Authentication Code" that uses a "Cryptographic Hash Function" with a "Secrete Key". With our protocol, to calculate the HMAC for the entire file we used the cryptographic function SHA-512 with a key K while to calculate the HMAC for the chunks of the file we used the cryptographic function SHA-1 with a key k.

Assumption: Since the client and the server are controlled by the same person we assume the hashing keys and the cryptographic functions that are used to calculate the $MAC(key, Message)$ can be safely applied to both the client and the server.

Summary of Protocol #1:
We compute the HMAC $T = MAC(K, M)$ of the entire file. After that, we divide the file into chunks and compute the HMAC $t_i = MAC'(K', m_i)$ for each chunk of the file. First, we create a packet of type #1 containing the calculated HMAC $T' = MAC(K, M')$ where M' consists of all the attributes inside the packet, the form of type#1 packet is $(T', andOtherAttributes)$, it will broadcast over all the available n channels. Then, we construct packets of type #2 that are formed of $(ti, andOtherAttributes)$ and send each packet in parallel over the n available channels. Repeat until all packets are sent. Meanwhile, at the beginning, the server will try to verify one of the type #1 packets, by computing the HMAC of the extracted attributes of each packet and then compare the calculated HMAC with the received HMAC T'. After that, for each type #2 packet received, it will compute $t_m = MAC'(K', m_i)$ and compare it with t_i. If they are equal, the server will accept the packet otherwise it will drop it. And when it receives the entire packets, it will compute $T_M = MAC(K, M_{received})$ of the entire file and compare it with T. If they are equal it will accept the entire file otherwise it will drop it.

Summary of Protocol #2:
We compute the HMAC $T = MAC(K, M)$ of the entire

file. After that, we divide the file into chunks and compute the HMAC $t_i = MAC'(K', m_i)$ for each chunk of the file. First, we create a packet of type #1 containing the calculated HMAC $T' = MAC(K, M')$ where M' consists of all the attributes inside the packet, the form of type#1 packet is $(T', andOtherAttributes)$, it will broadcast over all the available n channels. Then, for each chunk of the file, we construct a packet of type #2 that contains $(ti, andOther\ Attributes)$. The client picks the first n packets and send a different packet over each of the n channels, then the client waits for an ACK from the server that contains the packets which were delivered correctly, the server will deduct the correctly received packets and will retransmit the remaining packets, this will be repeated until all the n packets are received correctly and then the client will move to the next n packets and so on. Meanwhile, the server at the beginning will try to verify one of the type #1 packets, by computing the HMAC of the attributes inside each packet and then compare the calculated HMAC with the received HMAC T'. After that for each packet of type #2 received the server will compute the HMAC $t_m = MAC'(K', m_i)$ and compare it with its corresponding t_i. If they are equal, the client will accept the packet and add it to the list of correctly received packets otherwise the packet will be dropped, when the server is done from verifying the n packets it will broad-

cast an ACK over all the n channels to the client containing the list of sequence numbers of the packets that were correctly received asking it to resend the rest of the packets and repeat until it correctly receives the entire n packets. So, when the server receives the complete file it will compute $T_M = MAC(K, M_{received})$ of the entire file and compare it with T that was sent in a packet of type #1. If they are equal it will accept the file otherwise it will drop it.

For Both Protocols: T protects the file as a whole, whereas each t_i protects a chunk of the file and tells us which chunk was corrupted. Tagging the index along with the chunk takes care of an adversary permuting the chunks. This protocol will work even by assuming that the adversary can corrupt the channels adaptively. i.e, at each run, the corrupted channels are chosen before Bob sends stuff. In other words, even if the watcher decides to corrupt always the same channels the protocol will converge since, after each run, some packets are duplicated or sent over a different channel.

4.1 Protocol #1 :

Assumes that the adversary can modify an unknown number of channels b out of n channels where $b < n$. s is the packet size used.

Suppose we have a file of size f that the server needs to exfiltrate from the client's machine. we divide this file into p packets where

$$p = \frac{f}{s}$$

Protocol #1 in general, will calculate the HMAC of the entire file and then form a packet containing:

- The HMAC of the complete packet calculated from the attributes::
  1. Packet type.
  2. Number of packets to be sent.
  3. Length of the file name.
  4. The file name that the client will send.
  5. HMAC of the entire file.
- Packet type.
- The file name that the client is going to send.
- The length of the file name.
- The number of packets to send.
- The HMAC of the entire file.

This packet will broadcast over the all n channels at the beginning of the exfiltration process to notify the server that

a file will be sent now. Meanwhile, the server will start receiving these packets. For the first packet received it will, extract the attributes, calculate the HMAC of the extracted attributes and then compare it with the received HMAC of the complete packet if they verify it will create a file, save other attributes for later comparisons and drop all other packets of the same type. Otherwise, it will drop the packet and move on to the next one. Then the client divides the complete file into chunks. For each chunk, we form a packet p_i containing:

- The HMAC of the complete packet calculated from the attributes:
  1. Packet type
  2. Sequence number.
  3. Chunk of data from the file.
- Packet type.
- Sequence Number.
- Chunk of data from the file.

We will broadcast each p_i packet over all the n channels in parallel. E.g, packet 1 will be sent over n channels simultaneously, then packet 2 will be sent also over all the available channel and so on, until we send all the p packets.

Meanwhile, the server is receiving these packets, it extracts the data inside them, then calculate the HMAC of the attributes:

- Packet type.
- Sequence Number.
- Chunk of data from the file.

The server will compare the calculated HMAC with the HMAC that was extracted from the packet, if they are equal it will drop all other packets that have the same sequence number otherwise it will drop it and then check the next packet and so on until it will have the complete file. After having the complete file it will calculate the HMAC of the entire file and compare it with the HMAC that was saved from the packet of type #1.

We define two types of packets in protocol #1 that will be explained later in this section.
The first step is done when the client broadcasts a packet of type #1 that contains:

We call this packet the "File Transfer Request", it is broadcast over all the n channels at the beginning. Its job is to

Complete HMAC	Op Code = 1	Number of Packets	Length of the File Name	File Name	HMAC of the file

Table 1: Type #1 packet of protocol #1.

prepare the server to receive a new file. When the server receives this packet it will:

- Extract the attributes:
  1. OP Code (Type of the packet).
  2. Number of Packets.
  3. Length of the File Name.
  4. File Name.
  5. Hash of the File.
- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the complete HMAC that is received within the packet.
- If they are equal it will:
  1. Create a file with the name “File Name”.
  2. Save the complete HMAC of the file.
  3. Save the number of packets of the entire file.
  4. Drop all other packets of the same type.
- If they are not equal it will move to the next packet.

This packet is broadcast only one time at the beginning of the transmission of each file and the server will accept the first packet where the HMAC verifies and it will drop the rest.

Then we have packet of type 2 that contains:

HMAC	Op Code = 2	sequence no.	Chunk of Data from the File

Table 2: Type #2 packet of protocol #1.

We call this packet the "Data packet", each data packet is also broadcast over all the n channels. Its job is to send the data $(chunks)$ of the file. When the server receives this packet it will:

- Calculate the HMAC of the extracted attributes.
  1. OP Code (Type of the packet).
  2. Sequence Number.
  3. Chunk of Data from the File.
- Compare the calculated HMAC with the received HMAC.
- If the HMACs are equal it will accept the packet and drop all other packets with the same sequence number.
- If the HMACs are not equal it will drop the packet and move to the next one.

Note: The final "Data packet" will be padded with (char $= 0$) by the client in case it is not equal to the defined packet size, and then it will be stripped by the server when it is received.
The server will keep reading these "Data Packets" until the total number of packets accepted equals the number of packets attribute that was received in the "File Transfer Request" packet. Then it will calculate the HMAC of the entire file received and compare it with the HMAC that was received in the "File Transfer Request" packet, if they are equal it will accept the file otherwise, it will drop it. After that, it will get ready to receive the next file.

This protocol guarantees the delivery of the complete file even in the presence of a watcher who is trying to modify the packets between the client and the server. Although, this protocol does not use acknowledgment packets from the server to specify which packets were correctly delivered or no. It has high costs in terms of bandwidth and number of rounds as each packet is being broadcast over every channel each time.

4.2 **Protocol** #2 :

Suppose we have n channels out of which the watcher can corrupt at most an unknown number of channels b where

$b \leq (n-1)$. Further, Suppose we have a file of size f that the server needs to exfiltrate from the client's machine. we divide this file into p packets where

$$p = \frac{f}{s}$$

Protocol #2 in general, will calculate the HMAC of the entire file and then form a packet containing:

- The HMAC of the complete packet calculated from the attributes:
  1. Packet type.
  2. Number of packets.
  3. Length of the file name.
  4. The file name that the client will send.
  5. HMAC of the entire file.
- Packet type
- The file name that the client is going to send.
- The length of the file name.
- The number of packets to send.
- The HMAC of the complete file.

This packet will broadcast over the all n channels at the beginning of the exfiltration process to notify the server that

a file will be sent now. Meanwhile, the server will start receiving these packets. For the first packet received it will, extract the attributes, calculate the HMAC of the extracted attributes and then compare it with the received HMAC. If they verify it will create a file, save other attributes for later comparisons and drop all other packets of the same type. Otherwise, it will drop the packet and move on to the next one. After that, the client will divide the complete file into chunks and for each chunk, we form a packet p_i containing:

- The HMAC of each chunk of the file calculated from the attributes:
  1. Packet type
  2. Sequence number.
  3. chunk of data from the file.
- Packet type
- Sequence Number.
- Chunk of data from the file.

Protocol #2 will choose the first n packets and then send each packet p_i over a different channel. E.g, over channel #1, packet #1 will be sent, over channel #2, packet #2 will be sent and so on. The packets will be sent in parallel (At the same time). Meanwhile, the server is receiving the packets,

it extracts the data inside each packet, then calculate the HMAC of the attributes:

- Packet type.
- Sequence Number.
- Data of the chunk.

Then it will compare the calculated HMAC with the HMAC that was received within the packet. If the HMACs verify it will accept the packet and add it to the list of accepted packets. Otherwise, it will drop it. After inspecting the n packets the server will use an ACK packet containing:

- The HMAC of the complete packet calculated from the attributes::
  1. Packet type
  2. Sequence numbers of the accepted packets.
- Packet type.
- Sequence Numbers of the accepted packets.

This Ack packet will be broadcast over all the n channels to inform the client about the accepted packets then the client will retransmit the packets that were not correctly delivered. This will be repeated until all the n packets are successfully received by the server and then the client will

start sending the next n packets and the same process will be repeated till all the p packets are correctly received by the server. When all the p packets are received, the server will calculate HMAC of the entire file and then compare it with the HMAC that was received in the first broadcast packet, if they verify, it will accept the file. Otherwise, it will drop it.

We define three types of packets in protocol #2 that will be explained later in this section. The first step is done when the client broadcasts a packet of type 1 that contains:

Complete HMAC	Op Code = 1	Number of Packets	Length of the File Name	File Name	HMAC of the entire file

Table 3: Type #1 packet of protocol #2.

As with protocol #1, we call this packet the "File Transfer Request", it will broadcast over all the n channels in the beginning. Its job is to prepare the server to receive a new file. When the server receives this packet it will:

- Extract the attributes:
    1. OP Code (Type of packet).
    2. Number of Packets.
    3. Length of the File Name.

4. File Name.
5. HMAC of the Data.

- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the complete HMAC that is received within the packet.
- If they are equal it will:
  1. Create a file with the name “File Name”.
  2. Save the complete HMAC of the file.
  3. Save the number of packets of the entire file.
  4. Drop all other packets of the same type.
  5. Prepare to receive the chunks of the file.
- If they are not equal it will move to the next packet.

This packet is broadcast only one time at the beginning of the transmission of each file and the server will accept the first packet of these where the hashes verify and then it will drop the rest.

Then we have packet of type 2 that contains:

We call this packet the “Data packet”, each data packet is sent over one available channel. Its job is to send the data of the file. When the server receives this packet it will:

Complete HMAC	Op Code = 2	sequence no.	Chunk of data from the file.

Table 4: Type #2 packet of protocol #2.

- Calculate the HMAC of the attributes:
  1. Packet type.
  2. Sequence Number.
  3. Chunk of data from the file.
- Compare the calculated HMAC with the received HMAC.
- If the HMACs are equal it will accept the packet and add the packet sequence number to a list of accepted packets.
- If the HMACs are not equal it will drop the packet.

Note: The final "Data packet" will be padded with $(\text{char} = 0)$ by the client in case it is not equal to the defined packet size, and then it will be stripped by the server when it is received.
The server will keep reading these "Data Packets" until the total number of packets successfully received equals the number of packets received in the "File Transfer Request" packet. Then it will calculate the HMAC of the entire file received and compare it with the HMAC that was received in the

"File Transfer Request" packet, if they are equal it will accept the file otherwise, it will drop it. Then it will get ready to receive the next file.

Then we have packet of type 3 that contains:

HMAC	Op Code = 3	sequence no. of the received packets

Table 5: Type #3 packet of protocol #2.

We call this packet the "Data Packet Acknowledgment", This packet type is broadcast to the client from the server after inspecting the n packets that were received. Its job is to notify the client which packets were correctly received. When the client receives this packet type it will:

- Extract the attributes:
  1. Op Code (Packet type).
  2. sequence numbers.
- Calculate the HMAC of the extracted attributes.
- Compare the calculated HMAC with the received HMAC.
- If they are equal it will:

1. Extract the sequence numbers of the received packets.
2. Deduct these sequence numbers from the pool of the available packets to be sent.
3. Drop the other packets of the same type.
4. Send the remaining packets over the n channels again.

- If they are not equal it will:
  1. drop the packet.
  2. start analyzing the next packet.

This ACK packet is broadcast over all the channels, we can afford that since the size of these packets is relatively small.

This protocol guarantees the delivery of the file even in the presence of a watcher who is trying to modify the packets between the client and the server. It has low costs in terms of bandwidth and number of rounds in comparison with protocol #1 as each packet is sent only one time over a channel and will only be retransmitted in case of a failed delivery. Although this protocol uses ACKs packets. These packets are relatively small in comparison of the data packets so in a way we conserved the low bandwidth.

5 Mathematical Logic

To understand better how the protocol is working, we derived two equations that will explain the ***security models***:

1. First equation will explain the ***round complexity***.
2. Second equation will explain the ***communication complexity***.

Round Complexity: Is defined as the number of rounds it takes the protocol to send p packets over n channels.
Let $n =$ total number of channels
Let $b =$ number of bad/corrupted channels
Let $p =$ number of packets to be transmitted
Let $q =$ max number of rounds
Consider $p = n$ for simplicity
Assume always $b < n$
General form of $b : b = c \times n$ Where c is the fraction of corrupted channels out of n.

Examples of the form of b:

$$b = \frac{2}{3}n, \qquad b = \frac{3}{4}n, \qquad b = \frac{1}{2}n$$

As an example, assume the total number of channels $n = 10$, the total number of packets to send $p = 10$, the number of corrupted channels is $b = 5$. So, $b = \frac{1}{2}n$.

- In round 1, we expect $\frac{1}{2}$ of the total packets to be delivered.
- In round 2, we expect $\frac{1}{2}$ of the remaining packets to be correctly delivered.
- In round 3, we expect $\frac{1}{2}$ of the remaining packets to be correctly delivered.

Based on this, we deduce the #packets that are correctly received in each round to be:

- ***Round*** #1: $(n - nc)$ packets, which translates to half of the packets are correctly delivered.
- ***Round*** #2: $(nc - nc^2)$ packets, which translates to half of the remaining packets will be correctly delivered.
- ***Round*** #3: $(nc^2 - nc^3)$ packets, which translates to half of the remaining packets will be correctly delivered.
- ***Round*** #q: $(nc^{(q-1)} - nc^q)$ packets, as this is the final round, all the remaining packets will be delivered in this round.

According to that, we can derive the equation:
#packets correctly delivered after q rounds =

$$\sum_{i=1}^{q}[(1-c)\times c^{(i-1)}\times n] \quad (1)$$

Now, in the case where $p > n$. Then, the general form of p will be: $p = m \times n + r$
Where m is the coefficient and r is the constant.
Examples of the form of p:

$$p = 4n + 2, \qquad p = 3n + 5, \qquad p = 17n + 2$$

In this case, the general form of the equation will become:
#packets correctly delivered after q rounds =

$$m[\sum_{i=1}^{q}[(1-c)\times c^{(i-1)}\times n]] + \sum_{i=\frac{n}{r}}^{q}[(1-c)\times c^{(i-1)}\times n] \quad (2)$$

In the case where $r = 0$. Then the equation will become:
#packets correctly delivered after q rounds =

$$m[\sum_{i=1}^{q}[(1-c)\times c^{(i-1)}\times n]] \quad (3)$$

Note: The above equation assumes that the watcher will always corrupt the maximum number of packets. In other words, this equation calculates the #packets of packets correctly delivered over each round in the worst case scenario.

5.1 proof

The above equation can be proved by Mathematical Induction:
$(n - nc) + (nc - nc^2) + ... + (nc^{(n-1)} - nc^n) = \sum_{i=1}^{q}[(1 - c)c^{(i-1)}n]$
To prove this by Induction we need to prove:

1. Prove that $P(1)$ is true.
2. Assume that $P(k)$ is true, then prove $P(k + 1)$ is true for every natural number k.
3. If the above conditions are true, then $P(n)$ is true for all natural numbers n.

Proof:

1. For $P(1)$:

$$(n - nc) \stackrel{?}{=} \sum_{i=1}^{q=1}[(1 - c)c^{(i-1)}n] \tag{4}$$

$$(1 - 1(c)) \stackrel{?}{=} (1 - c)c^{(1-1)}(1) \tag{5}$$

$$(1 - c) \stackrel{?}{=} (1 - c)(1)(1) \tag{6}$$

$$(1 - c) = (1 - c) \tag{7}$$

This proved that $p(1)$ is true.
Reminder:$P(K) = (kc^{(k-1)} - kc^k) = \sum_{i=k}^{q}[(1 - c)c^{(k-1)}k]$

2. For $P(k+1)$:

$$((k+1)c^{((k+1)-1)} - (k+1)c^{(k+1)}) \stackrel{?}{=} (1-c)c^{((k+1)-1)}(k+1) \tag{8}$$

$$((k+1)c^{k} - (k+1)c^{(k+1)}) \stackrel{?}{=} (1-c)c^{k}(k+1) \tag{9}$$

$$(k+1)[c^{k} - c^{(k+1)}] \stackrel{?}{=} (1-c)c^{k}(k+1) \tag{10}$$

$$(k+1)[c^{k} - c^{(k+1)}] = (k+1)[(c^{k} - c^{(k+1)}] \tag{11}$$

3. By Mathematical Induction:

$$P(n) : (n - nc) + (nc - nc^2) + ... + (nc^{(n-1)} - nc^n)$$

$$=$$

$$\sum_{i=1}^{n}[(1-c)c^{(i-1)}n]$$

Is true for all natural numbers n.

Communication Complexity: Is defined as the total number of bytes we're sending in order to deliver the complete file with respect to the number of bytes of the file.
Let $n =$ The total number of channels.
Let $s =$ The size of the packet.
Let $r =$ The total number of rounds.
let $f =$ The file size.

The Communication complexity equation is:

$$cc = n \times s \times r \tag{12}$$

Also, we define the efficiency of the protocol which is:

$$\epsilon = \frac{f}{cc} \times 100 \tag{13}$$

6 Experiments

As mentioned before, the provided equation calculates the maximum number of rounds needed to receive a file. To mimic this situation in the experiments, we randomly changed the position of the corrupted channels after every round.

To test the efficiency of these protocols, we implemented them simulating a real word scenario. Note that we implemented the protocols using normal TCP channels since it doesn't make any difference if they were covert channels or normal channels. Through out the experiments we consider our computer running $(macOS10.13.4)$ positioned in Rome as the victim having these characteristics:

- CPU: 4.
- Processor: Intel Core i5.
- Clock Speed: 2,5 GHz.
- Memory: 8GB.

While the attacker server is a rented machine running $(ubuntu-xenial-16.04)$ from amazon located in Sydney "Australia" with the characteristics below:

- vCPU: 1.
- Processor: Intel Xeon.

- Clock Speed: 2,5 GHz.
- Memory: 1GB.

To check how the number of corrupted channels with respect to the number of total available channels affects the efficiency and the number of rounds it takes the protocol to send a file. We conducted the following experiments with a file of size $6MB$ that will be divided into 93 packets:

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #14). In addition in (Fig. #15) we compared them with protocol #1 by using 10 channels.

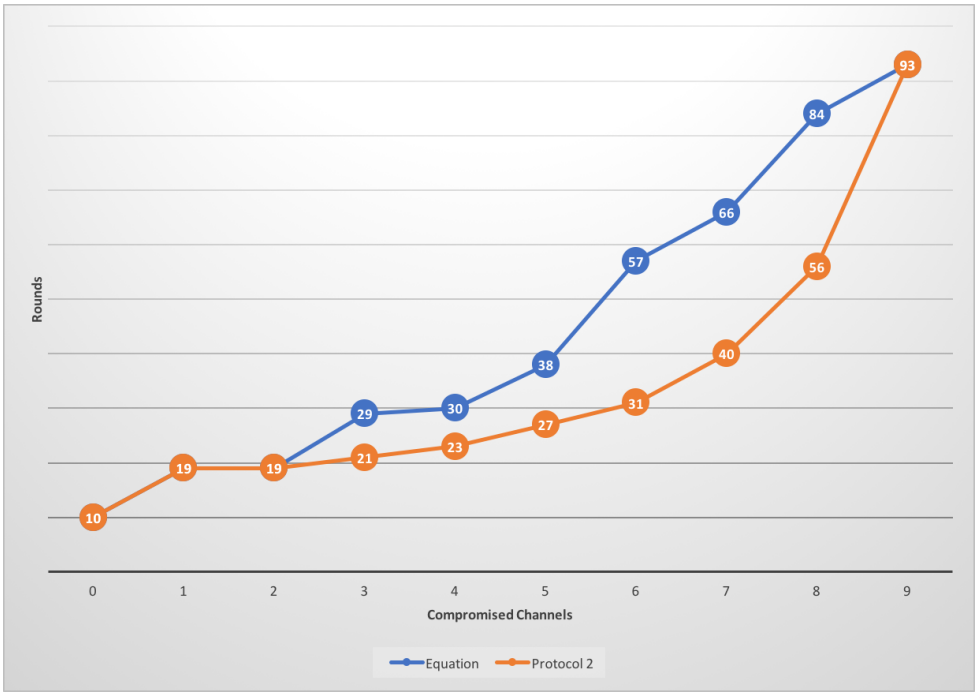

Figure 14: The difference in number of rounds taken protocol #2, and the equation

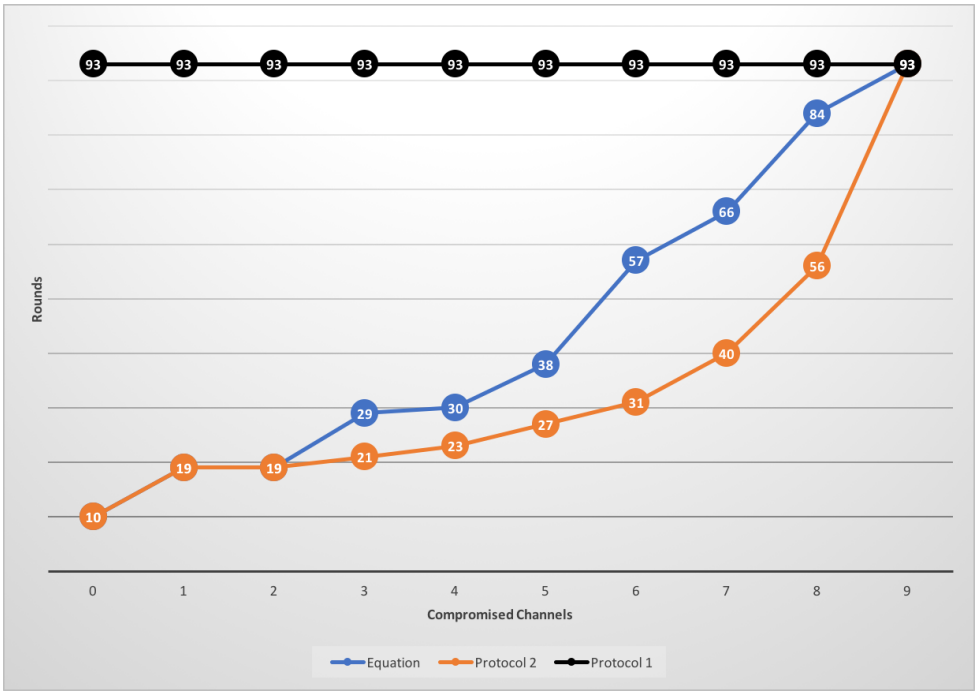

Figure 15: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #16). In addition in (Fig. #17) we compared them with protocol #1 by using 20 channels.

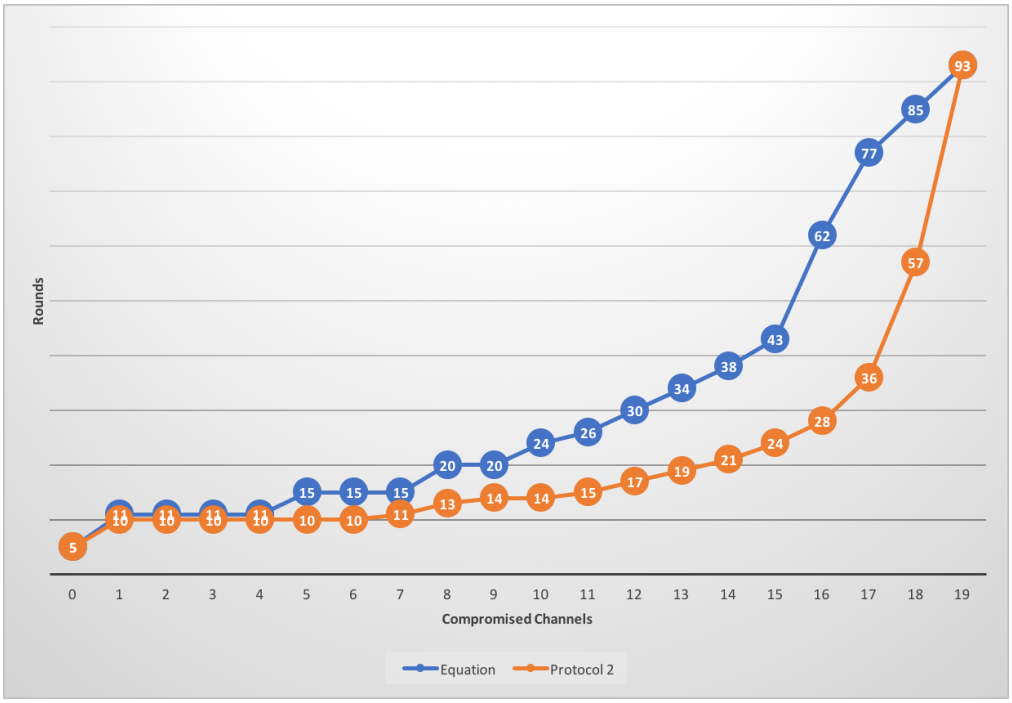

Figure 16: The difference in number of rounds taken protocol #2, and the equation

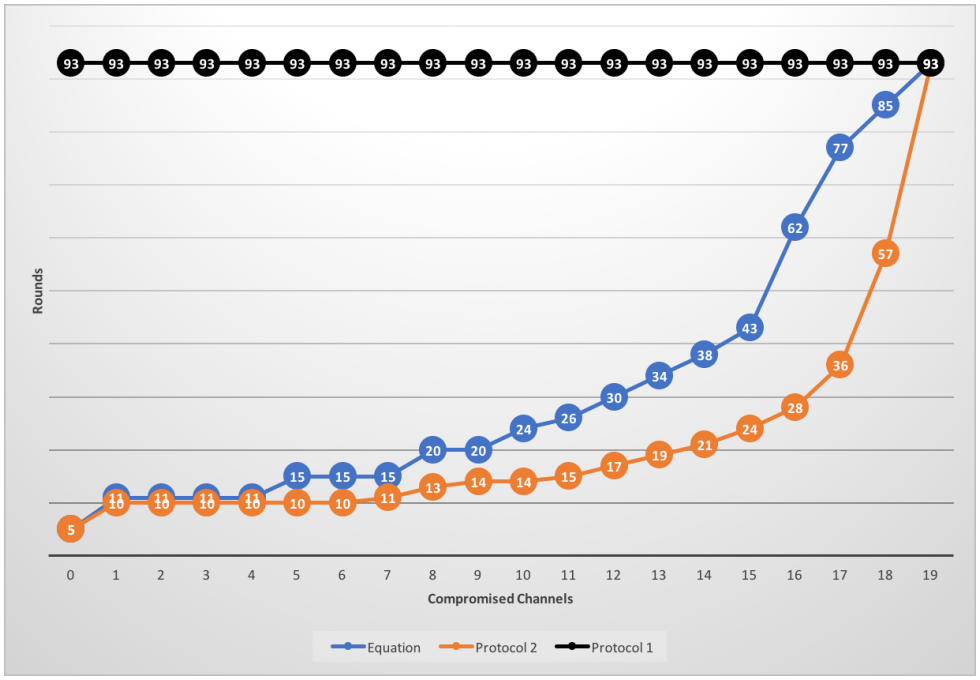

Figure 17: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #18). In addition in (Fig. #19) we compared them with protocol #1 by using 30 channels.

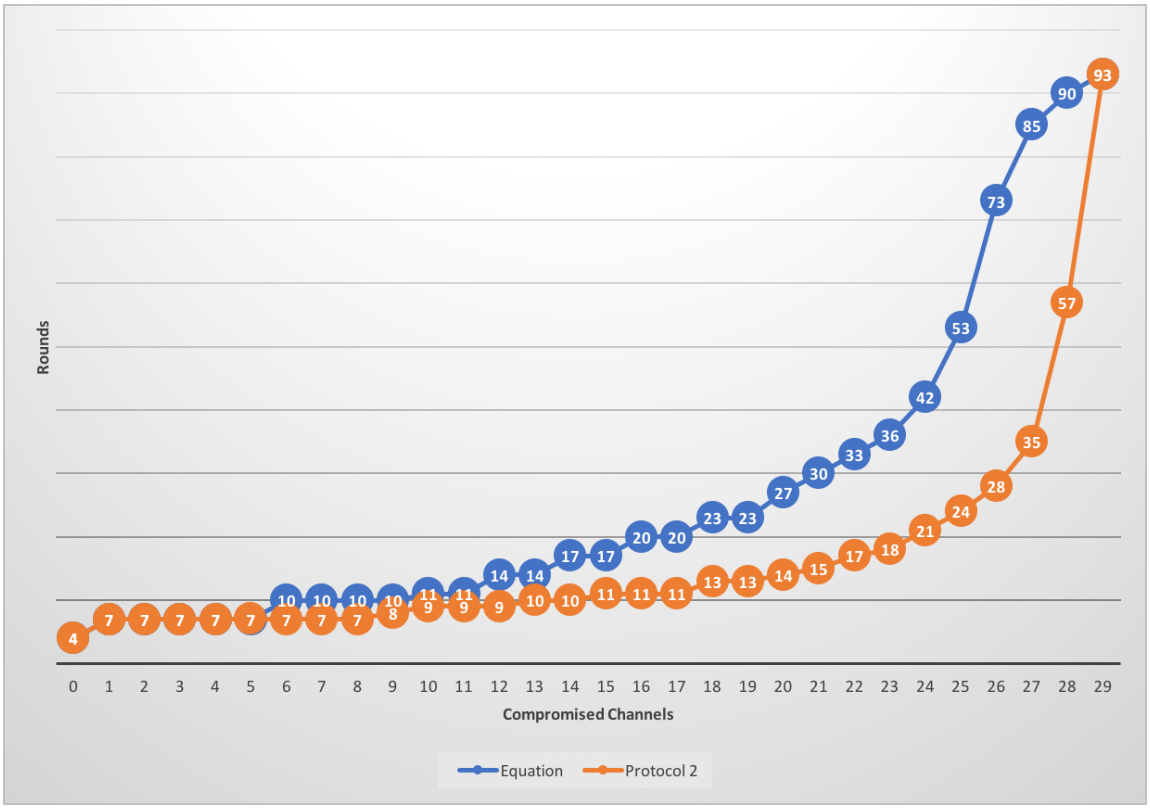

Figure 18: The difference in number of rounds taken protocol #2, #1

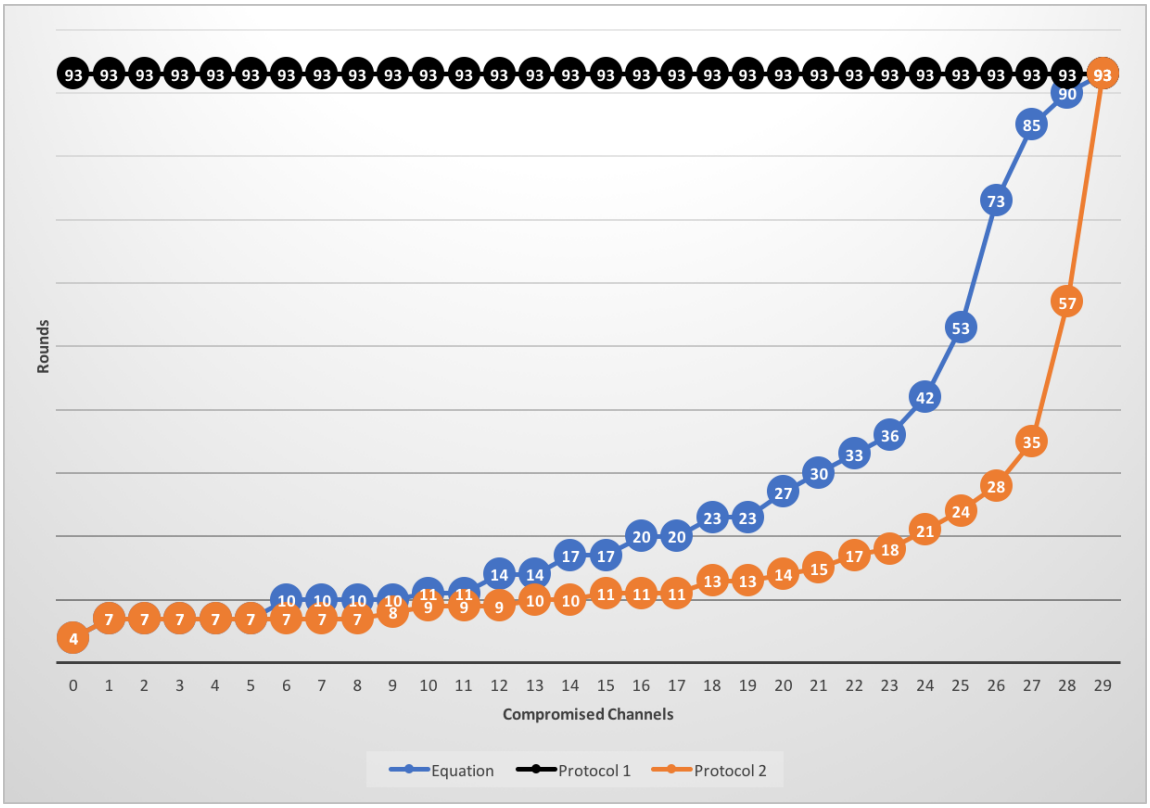

Figure 19: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the efficiency of the protocols is changing with respect to the number of corrupted channels.

- With a total number of 10 channels. (Fig. #20).
- With a total number of 20 channels. (Fig. #21).
- With a total number of 30 channels. (Fig. #22).

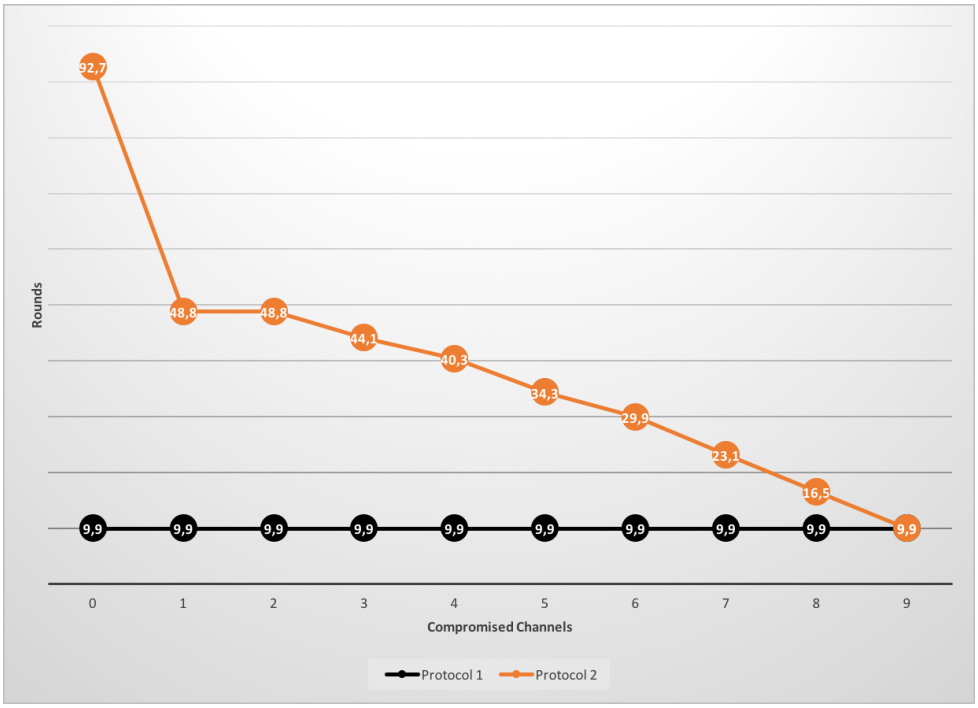

Figure 20: The efficiency of protocols #1 & #2 w.r.t 10 compromised channels.

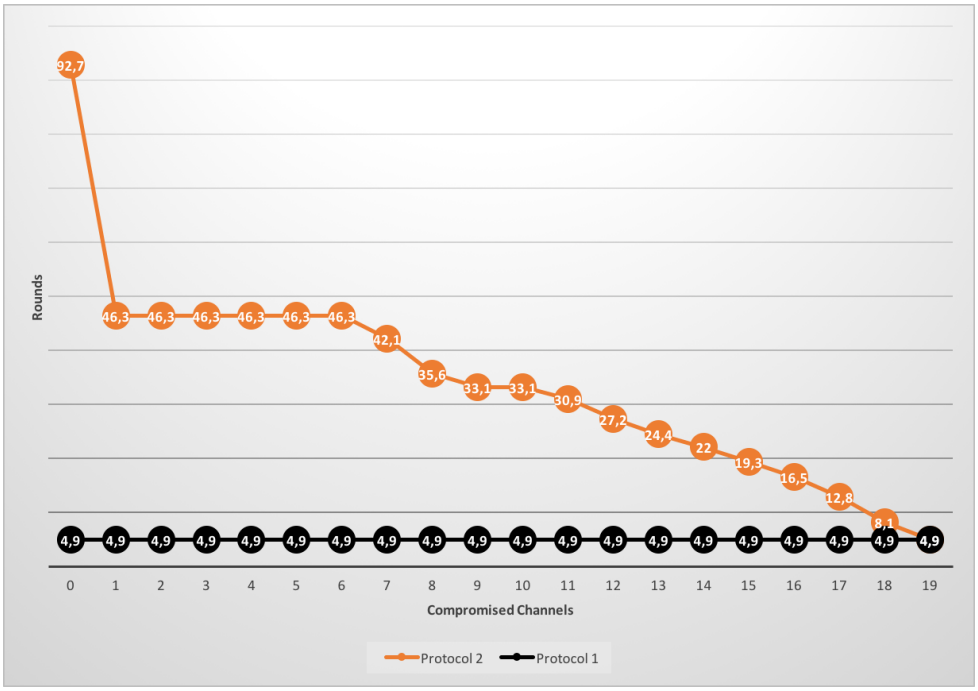

Figure 21: The efficiency of protocols #1 & #2 w.r.t 20 compromised channels.

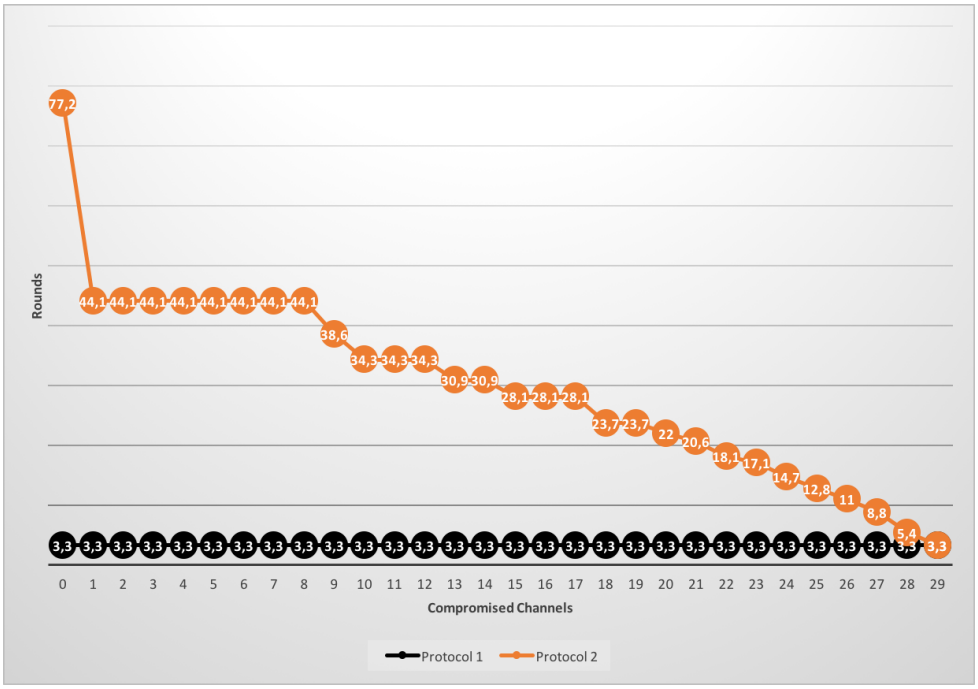

Figure 22: The efficiency of protocols #1 & #2 w.r.t 30 compromised channels.

6.1 Analysis

6.1.1 General Notes:

First thing we can notice is that Protocol #1 always needs a fixed number of rounds to send the file regardless of the number of available channels or the number of corrupted channels. And the number of rounds is the same as the number of packets to be sent which is expected from protocol #1 as it broadcasts every packet over all the channels at each round. So, by having for example, 10 packets the protocol will broadcast 1 packet in every round and therefore, to send 10 packets, it will take 10 rounds.

Now, by moving to protocol #2 we can notice that, with 0 compromised channels, the more total available channels we have the number of rounds needed to send the file decreases.

With only 1 channel that is not compromised, we can notice that protocol #2 acts in the same way as protocol #1, since at every round only 1 packet is correctly received.

6.1.2 Case Study: Figure #14

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

So this graph visualises how many rounds have taken the equation and the experiments to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 3 ***Compromised Channels***: A turning point for both the equation line and the experiments.
- 9 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 2$ ***Compromised Channels***: Both the protocol and the equation slightly increases but still taking the same number of rounds to send the file.
- $3 - 8$ ***Compromised Channels***: The number of rounds increases with both the equation and the experiments where with the equation it is always higher.
- $8 - 9$ ***Compromised Channels***: The line of the equation slightly increases till it reaches its maximum point while the line of the experiments increases rapidly to reach its maximum point.

6.1.3 Case Study: Figure #16

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

So this graph visualises how many rounds have taken the equation and the experiments to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 3 ***Compromised Channels***: A turning point for the equation line.
- 7 ***Compromised Channels***: A turning point for the experiments line.

- 15 ***Compromised Channels***: The turning of the equation line where it started to increase significantly.
- 16 ***Compromised Channels***: The turning of the experiments line where it started to increase rapidly.
- 19 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 4$ ***Compromised Channels***: Both the protocol and the equation lines slightly increase but still taking the same number of rounds to send the file.
- $5 - 15$ ***Compromised Channels***: The equation line is steadily increasing.
- $7 - 16$ ***Compromised Channels***: The experiment line is steadily increasing.
- $15 - 19$ ***Compromised Channels***: The equation line is rapidly increasing.
- $7 - 19$ ***Compromised Channels***: The experiment line is rapidly increasing.

6.1.4 Case Study: Figure #18

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

So this graph visualises how many rounds took the equation and the experiments to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 5 ***Compromised Channels***: A turning point for the equation line.
- 8 ***Compromised Channels***: A turning point for the experiments line.

- 24 ***Compromised Channels***: The turning of the equation line where it started to increase significantly.
- 27 ***Compromised Channels***: The turning of the experiments line where it started to increase rapidly.
- 29 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 5$ ***Compromised Channels***: Both the protocol and the equation lines slightly increase but still taking the same number of rounds to send the file.
- $5 - 24$ ***Compromised Channels***: The equation line is slowly increasing.
- $8 - 27$ ***Compromised Channels***: The experiment line is increasing little by little.
- $24 - 29$ ***Compromised Channels***: The equation line is rapidly increasing.
- $27 - 29$ ***Compromised Channels***: The experiment line is rapidly increasing.

We can deduce that, there is a proportional relation between the number of corrupted channels and the number of

rounds it takes the protocol to send a file. In other words, as the number of corrupted channels increases, the number of rounds will also increase.

6.1.5 Case Study: Figures (#20, #21&#22)

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the efficiency of each protocol.
- ***Black line***: The efficiency of protocol #1.
- ***Orange line***: The efficiency of protocol #2.

So these graphs visualise the efficiency of protocols #1 and #2 over the number of compromised channels.

By looking at the three figures, we can notice that the line of protocol #1 is always steady in each figure, and the efficiency is decreasing by increasing the number of total channels. Moreover, the efficiency and the number of compromised channels are independent. And that is because protocol #1 is broadcasting the same packet over all the channels at each round. Plus, by increasing the number of channels, we're sending more packets. And with that, we're making it less efficient. Therefore, we can say that the efficiency

protocol #1 is inversely proportional to the total number of available channels.

Analyzing trends of protocol #2

1. ***For Figure*** #20:
    - $0-1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
    - $1-2$ ***Compromised Channels***: The efficiency is stable.
    - $2-9$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.
2. ***For Figure*** #21:
    - $0-1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
    - $1-6$ ***Compromised Channels***: The efficiency is stable.
    - $6-19$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.
3. ***For Figure*** #22:
    - $0-1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.

- $1 - 8$ ***Compromised Channels***: The efficiency is stable.
- $8 - 29$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.

We can deduce that, the number of corrupted channels and the efficiency of protocol #2 are inversely proportional. In other words, as the number of corrupted channels increases, the efficiency decreases.

To confirm these results. We repeated the same experiments with a larger file $40MB$ that is divided into 230 packets in total.
The graphs below will show how the total number of rounds required to send the file change when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #23). In addition in (Fig. #24) we compared them with protocol #1. By using a total 10 channels.

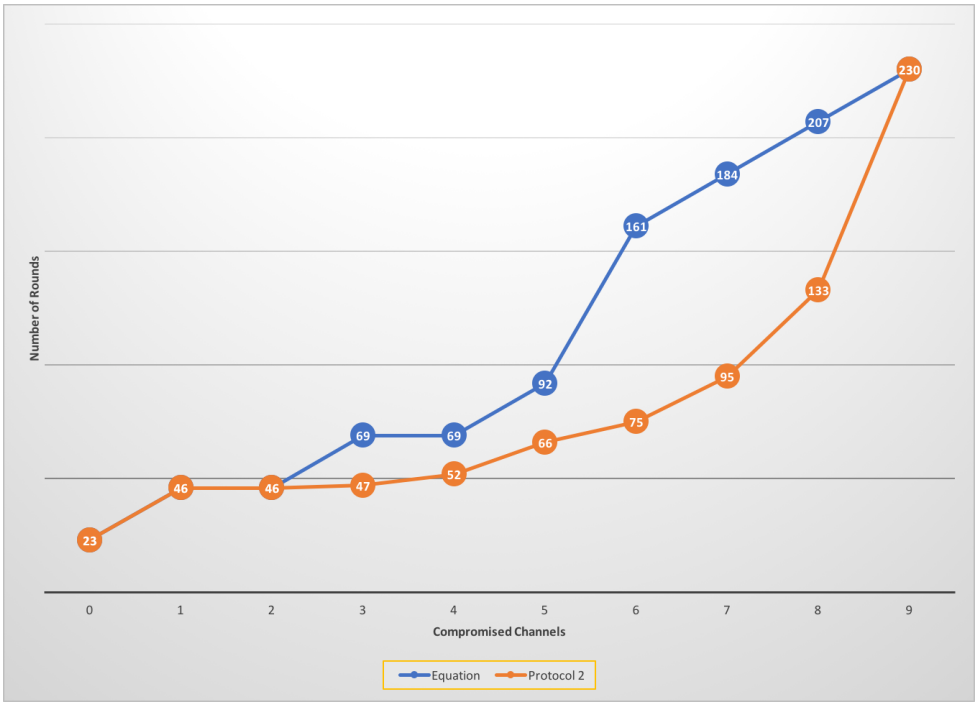

Figure 23: The difference in number of rounds taken protocol #2, and the equation

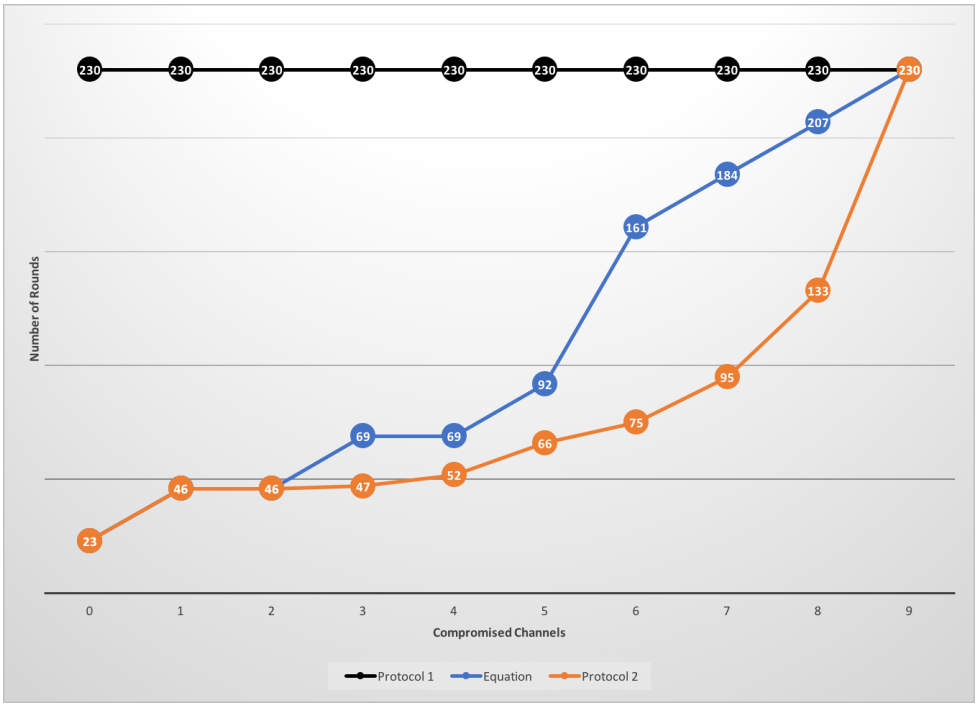

Figure 24: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #25). In addition in (Fig. #26) we compared them with protocol #1. By using a total of 20 channels.

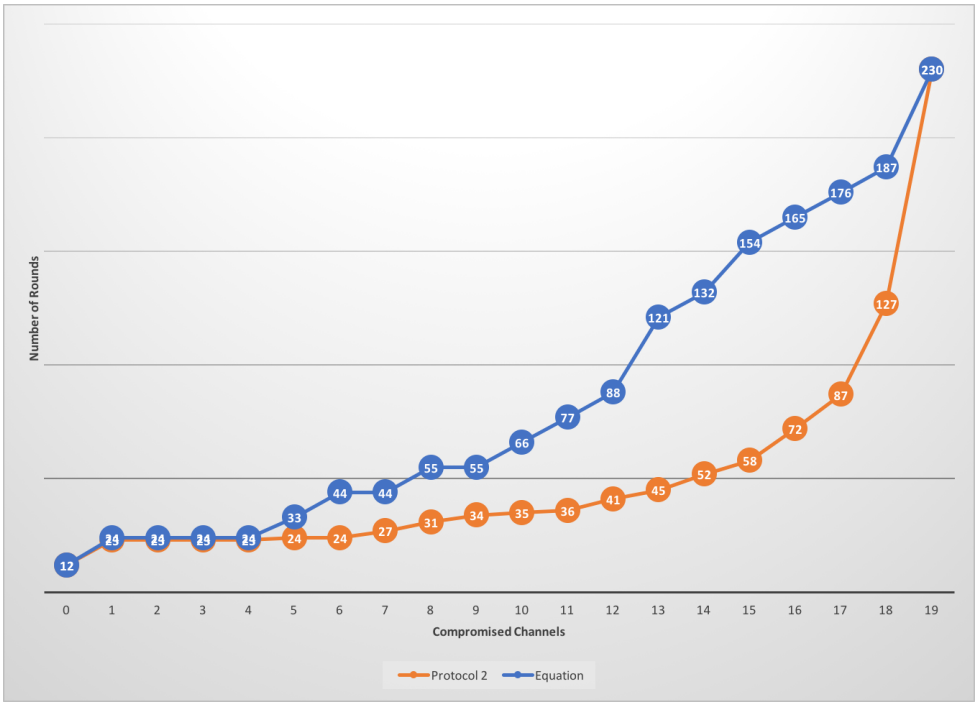

Figure 25: The difference in number of rounds taken protocol #2, and the equation

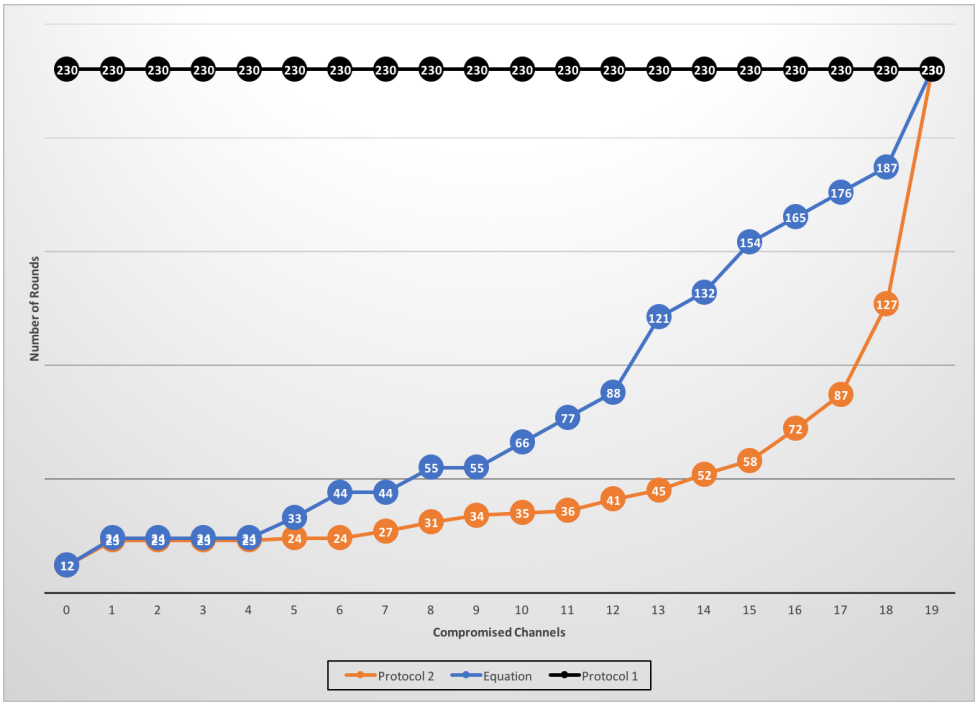

Figure 26: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the total number of rounds required to send the file changes when increasing the number of corrupted channels using protocol #2 and the equation (Fig. #27). In addition in (Fig. #28) we compared them with protocol #1. By using a total of 30 channels.

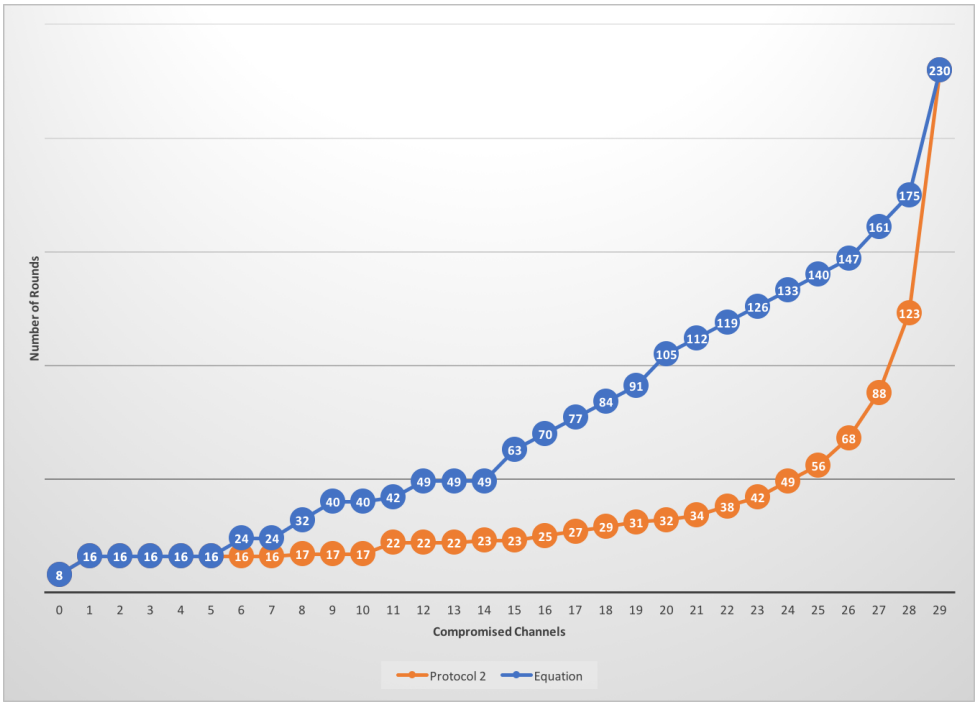

Figure 27: The difference in number of rounds taken protocol #2 and the equation

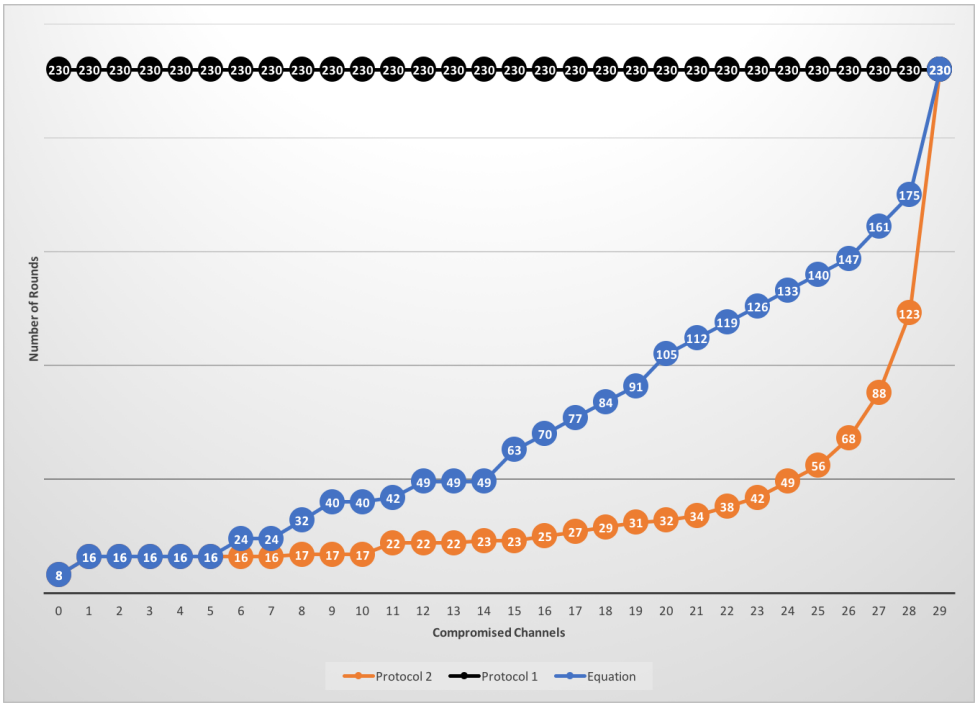

Figure 28: The difference in number of rounds taken protocol #2, #1 and the equation

The graphs below will show how the efficiency of the protocols is changing with respect to the number of corrupted channels.

- With a total number of 10 channels. (Fig. #29).
- With a total number of 20 channels. (Fig. #30).
- With a total number of 30 channels. (Fig. #31).

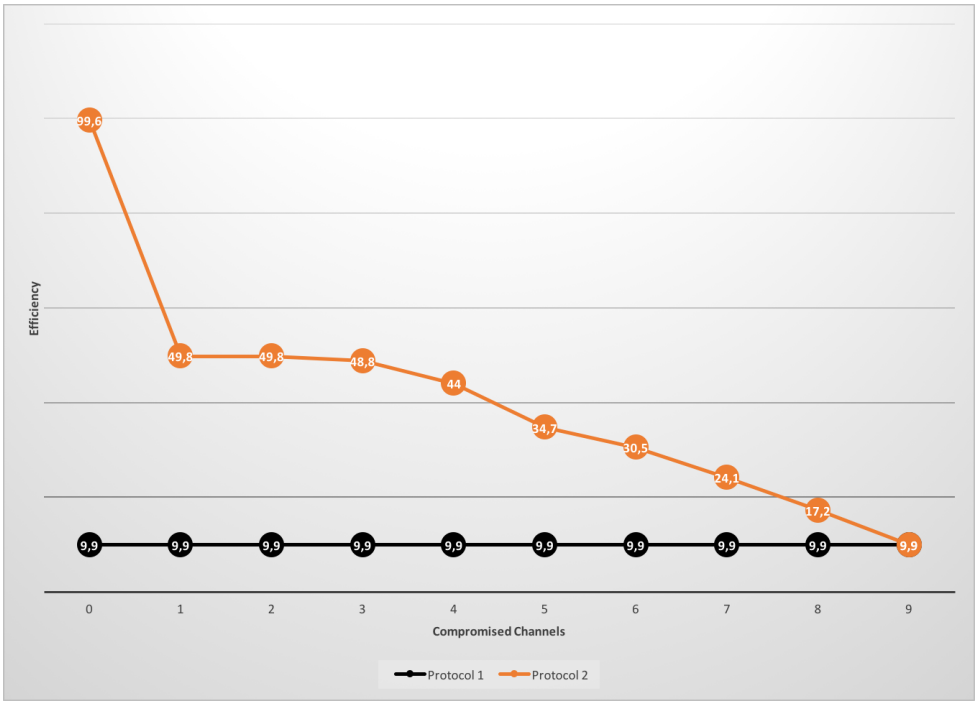

Figure 29: The efficiency of protocols #1 & #2 w.r.t the compromised channels.

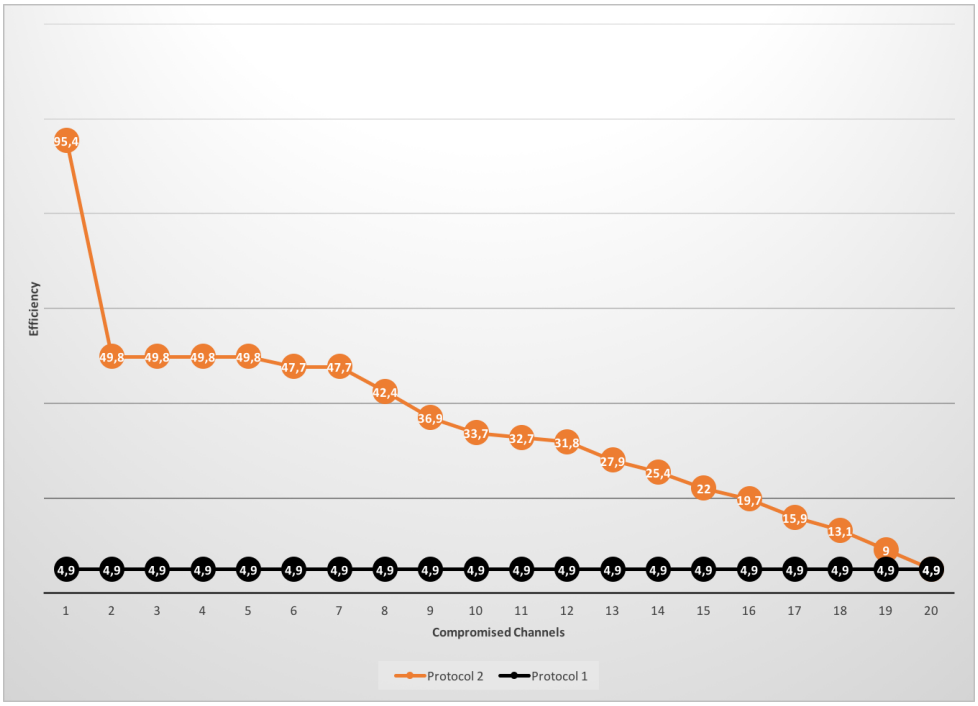

Figure 30: The efficiency of protocols #1 & #2 w.r.t the compromised channels.

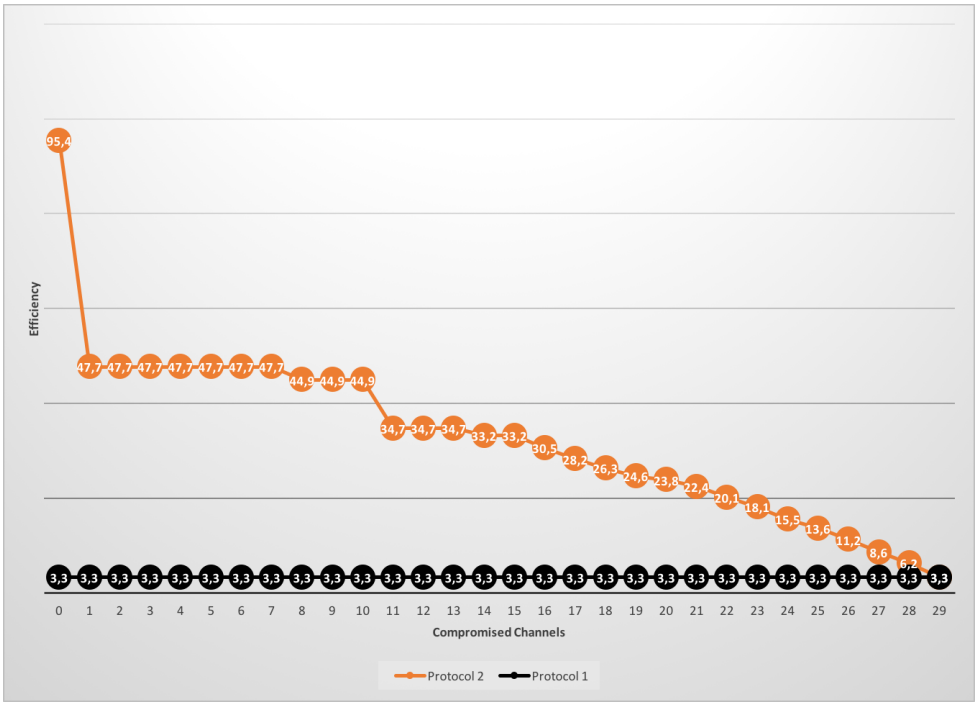

Figure 31: The efficiency of protocols #1 & #2 w.r.t the compromised channels.

6.2 Analysis

6.2.1 General Notes:

As before, we notice that Protocol #1 always needs a fixed number of rounds to send the file regardless of the number of available channels or the number of corrupted channels. And the number of rounds is the same as the number of packets to be sent which is expected from protocol #1 as it broadcasts every packet over all the channels at each round. So, by having for example, 10 packets the protocol will broadcast 1 packet in every round and therefore, to send 10 packets, it will take 10 rounds.

Now, by moving to protocol #2 we can notice that, with 0 compromised channels, when we increase the number of total available channels, the number of rounds needed to send the file decreases.

With only 1 channel that is not compromised, we can notice that protocol #2 acts in the same way as protocol #1, since at every round only 1 packet is correctly received.

6.2.2 Case Study: Figure #23

- ***X-axis***: Shows the number of compromised/bad channels.

- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

This graph visualises how many rounds have taken the equation and the experiments to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 3 ***Compromised Channels***: A turning point for both the equation line and the experiments.
- 9 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 2$ ***Compromised Channels***: The lines of both the protocol and the equation slightly increase but still taking the same number of rounds to send the file.

- $3 - 8$ ***Compromised Channels***: The lines of both the equation and the experiment increase where the line of the experiment is always higher.
- $8 - 9$ ***Compromised Channels***: The line of the equation slightly increases till it reaches its maximum point while the line of the experiments increases rapidly to reach its maximum point.

6.2.3 Case Study: Figure #25

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments of protocol #2.

This graph visualises how many rounds taken the equation, and the experiments of protocol #2 to send the file with respect to the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.
- 3 ***Compromised Channels***: A turning point for the equation line.
- 7 ***Compromised Channels***: A turning point for the experiments line.
- 12 ***Compromised Channels***: The turning of the equation line where it started to increase significantly.
- 17 ***Compromised Channels***: The turning of the experiments line where it started to increase rapidly.
- 19 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 4$ ***Compromised Channels***: Both the protocol and the equation slightly increases but still taking the same number of rounds to send the file.
- $5 - 12$ ***Compromised Channels***: The equation line is steadily increasing.
- $7 - 17$ ***Compromised Channels***: The experiment line is steadily increasing.

- 12 – 19 ***Compromised Channels***: The equation line is rapidly increasing till it reaches its maximum point.
- 17 – 19 ***Compromised Channels***: The experiment line is rapidly increasing till it reaches its maximum point.

6.2.4 Case Study: Figure #27

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the number of rounds taken to completely send the file.
- ***Blue line***: The number of rounds calculated by the equation.
- ***Orange line***: The number of rounds extracted from the experiments.

This graph visualises how many rounds taken the equation, and the experiments of protocol #2 to send the file over the number of compromised channels.

Analyzing points

- 0 ***Compromised Channels***: Both the protocol and the equation takes the lowest number of rounds possible to send the file.

- 5 ***Compromised Channels***: A turning point for the equation line.
- 27 ***Compromised Channels***: The turning of the experiments line where it started to increase rapidly.
- 29 ***Compromised Channels***: The peak of both the equation and the experiments lines.

comparing trends:

- $0 - 5$ ***Compromised Channels***: Both the lines of the protocol and the equation slightly increase but still taking the same number of rounds to send the file.
- $5 - 28$ ***Compromised Channels***: The equation line is steadily increasing.
- $11 - 28$ ***Compromised Channels***: The experiment line is increasing little by little.
- $28 - 29$ ***Compromised Channels***: The equation line is rapidly increasing till it reaches its maximum point.
- $28 - 29$ ***Compromised Channels***: The experiment line is rapidly increasing till it reaches its maximum point.

Now, we can confirm the proportional relation between the number of corrupted channels and the number of rounds

it takes the protocol to send a file. In other words, as the number of corrupted channels increases, the number of rounds will also increases.

6.2.5 Case Study: Figures (#29, #30&#31)

- ***X-axis***: Shows the number of compromised/bad channels.
- ***Y-axis***: Shows the efficiency of each protocol.
- ***Black line***: The efficiency of protocol #1.
- ***Orange line***: The efficiency of protocol #2.

These graphs visualise the efficiency of protocols #1 and #2 with respect to the number of compromised channels.

By looking at the three figures, as we noticed from the first experiment the line of protocol #1 is always steady in each figure, and the efficiency is decreasing by increasing the number of total channels. Moreover, the efficiency and the number of compromised channels are independent. And that is because protocol #1 is broadcasting the same packet over all the channels at each round. Plus, by increasing the number of channels, we're sending more packets. And with that we're making it less efficient. Therefore, we can say

that, the efficiency protocol #1 is inversely proportional to the total number of available channels.

Analyzing trends of protocol #2

1. ***For Figure*** #29:
    - $0 - 1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
    - $1 - 2$ ***Compromised Channels***: The efficiency is stable.
    - $2 - 9$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.
2. ***For Figure*** #30:
    - $0 - 1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.
    - $1 - 5$ ***Compromised Channels***: The efficiency is stable.
    - $5 - 19$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.
3. ***For Figure*** #31:
    - $0 - 1$ ***Compromised Channels***: The efficiency is at its maximum but it starts decreasing quickly.

- $1 - 7$ ***Compromised Channels***: The efficiency is stable.
- $7 - 10$ ***Compromised Channels***: The efficiency is slightly decreasing.
- $10 - 11$ ***Compromised Channels***: The efficiency is quickly decreasing.
- $11 - 29$ ***Compromised Channels***: The efficiency is decreasing steadily till it reaches its minimum point.

The same deduction with the first experiment, there is an inverse proportional relation between the number of corrupted channels and the efficiency of protocol #2. In other words, as the number of corrupted channels increases, the efficiency decreases.

7 Conclusions & Future Work

7.1 Conclusion

In this thesis, we introduced an innovative protocol for data exfiltration, which uses multiple covert channels over multiple ports to exfiltrate data. A discussion about various existing exfiltration techniques was provided along with experiments that prove the functionality of our protocol.

The main contributions of this protocol are:

1. Finding an effective way around a watcher who's trying to manipulate the data that is being exfiltrated by protecting the entire file by calculating its HMAC and then divide the file into small chunks that were protected by calculating the HMAC of each chunk.
2. One of the main advantages of our protocol is not having a single−point−of−failure. E.g, If one of the channels or ports is blocked, our protocol will continue to work normally as long as we have only one working channel.
3. Introducing the concept of rounds, which proved a very good and stable way to measure the effectiveness of the protocol.

7.2 Future Work

In this research we presented an innovative data exfiltration technique but since our motivation was to aid the cybersecurity community in fighting data exfiltration. Then as a future work we suggest a counter measure against this type of attack by presenting the concept of a "Fast Watcher" with the ability to add noise to multiple channels rapidly in a sense that in every round the watcher will try to disrupt as much channels as possible. We believe that this could help a lot in slowing down the process of exfiltration. Another way to fight against this technique would be by finding the malicious computer and then block the connection from it to Alice's server.

References

Brown, E., Yuan, B., Johnson, D., & Lutz, P. (2010). Covert channels in the http network protocol: Channel characterization and detecting man-in-the-middle attacks. In *International conference on cyber warfare and security* (p. 56).

Couture, E. (2010). Covert channels. *Commun. ACM*.

Dodis, Y., & Fiore, D. (2014). Interactive encryption and message authentication. In *International conference on security and cryptography for networks* (pp. 494–513).

Faloutsos, M., Faloutsos, P., & Faloutsos, C. (1999). On power-law relationships of the internet topology. In *Acm sigcomm computer communication review* (Vol. 29, pp. 251–262).

Febryan, A., Purboyo, T. W., & Saputra, R. E. (2017). Steganography methods on text, audio, image and video: A survey. *International Journal of Applied Engineering Research*, *12*(21), 10485–10490.

Fielding, R., & Reschke, J. (2014). *Hypertext transfer protocol (http/1.1): Semantics and content* (Tech. Rep.).

Gligor, V. (1993). *A guide to understanding covert channel analysis of trusted systems, version 1 (light pink book)* (Tech. Rep.). NCSC-TG-030, Library No. S-240,572.

Gligor, V. D. (1994). *A guide to understanding covert channel analysis of trusted systems* (Vol. 30). The Center.

Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM*, *16*(10), 613–615.

Liu, S., & Kuhn, R. (2010). Data loss prevention. *IT professional*, *12*(2).

Morandin, G. M. (2010, February 2). *Transmission control protocol (tcp)*. Google Patents. (US Patent 7,656,800)

Murdoch, S. J., & Lewis, S. (2005). Embedding covert channels into tcp/ip. In *International workshop on information hiding* (pp. 247–261).

Postel, J., & Reynolds, J. (1985). File transfer protocol.

Schear, N., Kintana, C., Zhang, Q., & Vahdat, A. (2006). Glavlit: Preventing exfiltration at wire speed. In *Proc. 5th workshop hot topics in networks* (pp. 133–138).

Simmons, G. J. (1984). The prisoners' problem and the subliminal channel. In *Advances in cryptology* (pp. 51–67).

Van Antwerp, R. (2011). *Exfiltration techniques: An examination and emulation* (Unpublished doctoral dissertation). University of Delaware.

von Solms, S., & van Heerden, R. (2015). The consequences of edward snowden nsa related information disclosures. In *Iccws 2015-the proceedings of the 10th international conference on cyber warfare and security: Iccws2015* (p. 358).

Wang, Z., & Lee, R. B. (2005). New constructive approach to covert channel modeling and channel capacity estimation. In *International conference on information security* (pp. 498–505).

Wendzel, S., & Mazurczyk, W. (2016). Poster: An educational network protocol for covert channel analysis using patterns. In *Proceedings of the 2016 acm sigsac conference on computer and communications security* (pp. 1739–1741).

Zander, S., & Armitage, G. (2008). Cchef–covert channels evaluation framework design and implementation.

Zander, S., Armitage, G., & Branch, P. (2007a). Covert channels and countermeasures in computer network protocols [reprinted from ieee communications surveys and tutorials]. *IEEE Communications Magazine*, *45*(12), 136–142.

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