KTH ROYAL INSTITUTE
OF TECHNOLOGY

Degree Project in Technology

Second cycle, 30 credits

A command-and-control malware design using cloud covert channels

Revealing elusive covert channels with Microsoft Teams

MASSIMO BERTOCCHI

Stockholm, Sweden, 2023

A command-and-control malware design using cloud covert channels

Revealing elusive covert channels with Microsoft Teams

MASSIMO BERTOCCHI

Master's Programme, Communication Systems, 120 credits
Date: June 30, 2023

Supervisors: Arve Gengelbach (KTH), Urs Müller (Compass Security Scheweiz AG)
Examiner: Cyrille Artho
School of Electrical Engineering and Computer Science
Host company: Compass Security Schweiz AG
Swedish title: En kommando och kontroll av skadlig programvara som använder en hemlig molnkanal
Swedish subtitle: Avslöjar svårfångade hemliga kanaler med Microsoft Team

Abstract

With the rise of remote working, business communication platforms such as Microsoft Teams have become indispensable tools deeply ingrained in the workflow of every employee. However, their increasing importance have made the identification and analysis of covert channels a critical concern for both individuals and organizations. In fact, covert channels can be utilized to facilitate unauthorized data transfers or enable malicious activities, thereby compromising confidentiality and system integrity. Unfortunately, traditional detection methods for covert channels may face challenges in detecting covert channels in such cloud-based platforms, as the complexities introduced may not be adequately addressed.

Despite the importance of the issue, a comprehensive analysis of covert channels in business communication platforms has been lacking. In fact, to the best of the our knowledge, this Master's thesis represents the first endeavor in identifying and analysing covert channels within Microsoft Teams.

To address this problem, an in-depth literature review was conducted to identify existing research and techniques related to covert channels, their detection and their countermeasures. A thorough analysis of Microsoft Teams was then carried out and a threat scenario was selected. Through extensive experimentation and analysis, three covert channels were then identified, exploited and compared based on bandwidth, robustness and efficiency.

This thesis sheds light on the diversity of covert channels in Microsoft Teams, providing valuable insights on their functioning and characteristics. The insights gained from this work pave the way for future research on effective detection systems for covert channels in cloud-based environments, fostering a proactive approach towards securing digital business communication.

Keywords

Covert channel, Command and Control, Microsoft Teams, Detection

Sammanfattning

Med ökningen av distansarbete har företagskommunikationsplattformar som Microsoft Teams blivit oumbärliga verktyg som är djupt rotade i arbetsflödet för varje anställd. Deras ökande betydelse har dock gjort identifiering och analys av dolda kanaler till ett kritiskt problem för både individer och organisationer. I själva verket kan dolda kanaler användas för att underlätta obehöriga dataöverföringar eller möjliggöra skadliga aktiviteter, vilket äventyrar sekretess och systemintegritet. Tyvärr kan traditionella detekteringsmetoder för dolda kanaler möta utmaningar när det gäller att upptäcka dolda kanaler i sådana molnbaserade plattformar, eftersom komplexiteten som introduceras kanske inte hanteras på ett adekvat sätt.

Trots frågans betydelse har det saknats en omfattande analys av dolda kanaler i plattformar för affärskommunikation. Såvitt vi vet är denna masteruppsats det första försöket att identifiera och analysera dolda kanaler inom Microsoft Teams.

För att ta itu med detta problem genomfördes en djupgående litteraturgenomgång för att identifiera befintlig forskning och tekniker relaterade till dolda kanaler, deras upptäckt och deras motåtgärder. Därefter gjordes en grundlig analys av Microsoft Teams och ett hotscenario valdes ut. Genom omfattande experiment och analys identifierades, utnyttjades och jämfördes sedan tre dolda kanaler baserat på bandbredd, robusthet och effektivitet.

Denna avhandling belyser mångfalden av dolda kanaler i Microsoft Teams och ger värdefulla insikter om deras funktion och egenskaper. Insikterna från detta arbete banar väg för framtida forskning om effektiva detekteringssystem för hemliga kanaler i molnbaserade miljöer, vilket främjar en proaktiv strategi för att säkra digital affärskommunikation.

Nyckelord

Hemlig kanal, ledning och kontroll, Microsoft Teams, upptäckt

Contents

List of Figures

Chapter 1

Introduction

The evolving threat landscape in cyberspace leads to the continuous improvement of corporate network monitoring. The aim is to detect abnormal network traffic, unusual accesses and associated infrequent actions; these are often the first indication of a security incident. Suspicious events are identified by the security operations centre, which may also take measures to counteract them, if necessary.

Today's network infrastructures are built in such a way that it is hard to gain direct access to a company's corporate network from the outside. Therefore, attackers rely on compromising workstations and initiating connections from the internal network to the internet, usually via a web proxy or a DNS tunnel. However, their strategy is getting more and more hindered as content filtering in proxies, as well as the monitoring of DNS traffic, is getting more advanced. Nowadays, with the increasing acceptance of remote working, the need for remote collaboration introduces new threats for the company's security through essential tools such as virtual phones, chat software or videoconferencing like offered by MS Teams, representing a new tunnel out of the company's network.

1.1 Background

This work shows how it is possible to defy the network's monitoring system and security measures to exfiltrate information via Microsoft Teams. This is done by establishing a communication with the victim's machine using one of the MS (Microsoft) Teams covert channels, and carrying out data extraction with a command-and-control approach (see Figure 1.1).
A covert channel refers to a type of cybersecurity attack in which a channel not intended for information transfer is actually exploited to exfiltrate data, bypassing the company's security policy [1].
A command-and-control approach refers to a computer-controlled by a third party, used to execute programs and transfer data to a malicious server.

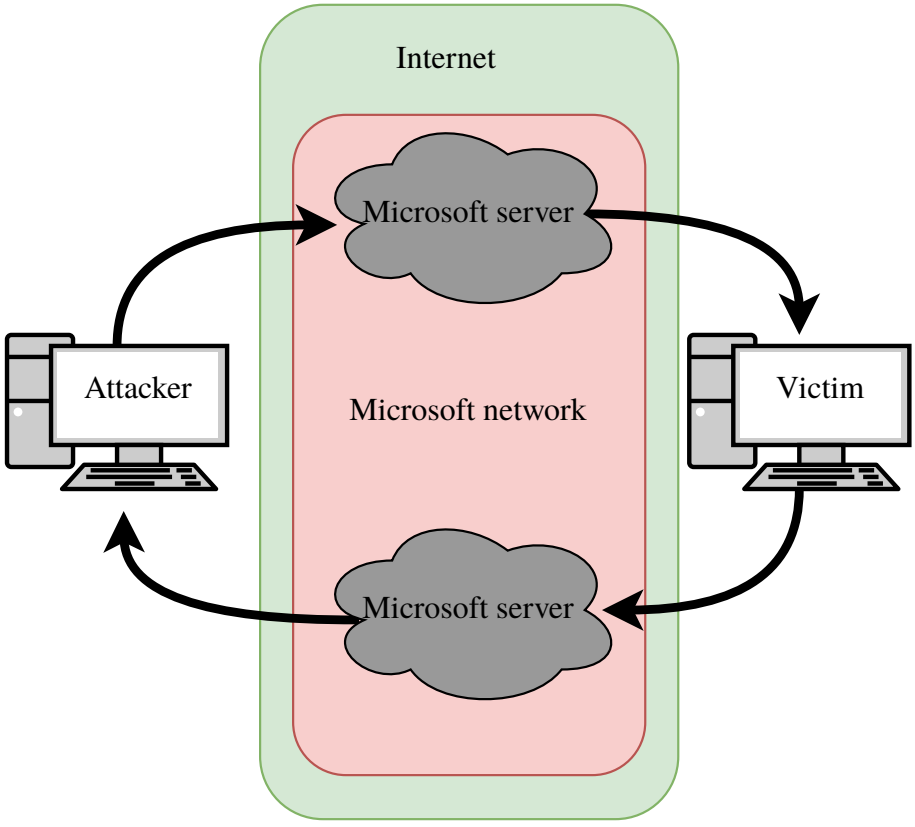


Figure 1.1: Covert channel scheme with MS Teams (Microsoft servers)

1.2 Problem

The problem question tackled in this work may be summarised as it follows:

Are the traditional methods sufficient to detect modern covert channel attacks using a cloud command-and-control architecture? Can we prove the existence of covert channels in MS Teams?

1.3 Scientific and engineering issues

The security perimeters defined by the security engineers in companies take for granted the security of the cloud applications and the ability of the provider to analyse, detect and prevent malicious activities. This situation might lead to malicious behaviour being undetected and network traffic not properly parse and sanitise.

1.4 Purpose

As a proof of concept, this work aims at designing a malware capable of exfiltrating information out of a victim's machine in a network, despite the security and monitoring measures, through Microsoft Teams. The logic behind the malware design proposed may then be applied to other cloud-based applications with equivalent functionality, pushing further the vulnerability discovery in other scenarios. The cloud provider has an important role when it comes to sensitive data transmitted through the cloud, and this work wants to draw attention to a shared security protection duty between the cloud provider and its customers.

New technologies have multiple weaknesses and vulnerabilities that might be known by the service owner, but a lack of funds prevents those from being fixed, stacking them up in a disregarded security backlog. However, as this security backlog grows, a new serious vulnerability which builds on top the previously noted ones might be discovered — like a covert channel, remote code execution, or information leaks. This paper raises awareness on the matter to security researchers and engineers, highlighting the importance of communication channels hidden from the end user.

1.5 Goals

The goal of this project is to propose a covert channel attack using a cloud infrastructure and then propose a mitigation technique to prevent this malware class. This has been divided into the following four sub-goals:

1. Analyse the functionality of Microsoft Teams.
2. Prove the existence of covert channels in Teams.
3. Create a command-and-control Malware using the Teams security weaknesses.
4. Evaluate the performance of the covert channel identified.
5. Analyse the covert channels against the state-of-the-art detection system.

The optimal outcome would be a custom malware to avoid state-of-the-art detection systems and robust cybersecurity policies: this implies standard procedures implemented by companies conforming to the best practices, such as blocking all incoming connection from the firewall, using robust cipher suite, or keeping the system updated to patch known security vulnerability.

1.6 Research Methodology

The project will start off with a security analysis of the MS (Microsoft) Teams application and its architecture. By analysing the message exchange of the Teams API, and comparing it with the documentation provided by Microsoft, one can note that the application exhibits undocumented behaviour.
We study whether the undocumented API endpoints have fewer security features implemented compared to the documented one, and a reverse engineering process needs to be then performed to explore all the API's functionality. We can infer the above by relying on the concept of security by obscurity, in order to avoid and maintain the system secure, the system needs to be hidden and secret. Finding a vulnerability within a hidden system can be more challenging than finding a vulnerability in a well-defined program. After discovering its weaknesses, the malware design might take advantage of one or more of these vulnerabilities.

1.7 Delimitations

The malware will be designed for Windows systems, and therefore has no guarantee to work also on Linux or iOS systems. The proposed mitigation is tested against the data collected from the Windows subsystem and it is not reliable for other systems.

The security analysis of MS Teams is not exhaustive, and new covert channels might still be discovered in the application or in the architecture of the system.
Additionally, Microsoft can roll out new patches to prevent and secure the weaknesses found during this thesis.

The code for this project will not be included in the Master's thesis, due to an NDA (Non-disclosure agreement) between the student and the company. Nevertheless, the concept behind the attack and the detection methodology are explained in detail.

1.8 Ethics and sustainability

When examining the design of malware, it is essential to consider the ethical and sustainability implications of such actions. The creation and use of malware can cause significant harm to individuals and organisations, leading to financial losses, compromised security, and even personal harm. As such, it is important to approach the thesis with a strong sense of ethical responsibility and to consider the potential consequences of creating such tools.

Furthermore, it is essential to consider the sustainability implications of malware design. Malware can have a long-lasting impact on systems, leading to prolonged security vulnerabilities and potential data loss. As such, creating malware that is designed to be sustainable, meaning it can be quickly and effectively removed, is crucial to reducing the overall impact on systems and the environment.

During the last phase of the project, all the found vulnerabilities and weaknesses are reported to the Microsoft security team. They would eventually take care of the security concerns and appropriately patch the

software to block threat actors to take advantage of this research.

1.9 Structure of the thesis

Chapter 2 presents relevant background information about network security, network communication and detection of covert channels. Chapter 3 presents the methodology and method used to solve the problem. Chapter 4 shows the security analysis made on MS Teams and the possibility to create covert channel over the Microsoft architecture. Chapter 5 highlights the properties and evaluate the covert channel described in the previous Chapter. Lastly, Chapter 6 indicates the future work on this topic and address the problem statement.

Chapter 2

Background

This chapter provides basic background information about network security, the technologies used in business networks to safeguard the environment from malicious third parties, as well as the network communication, protocol and architecture that will be employed in this project. This chapter also discusses prior research in the field of covert channels, along with their potential implementation, cloud-based command and control, a collection of tools and strategies used by an attacker to maintain influence over the victim's users, and covert channel detection.

2.1 Network security

Network security is the combination of multiple technologies together in order to keep applications, devices and systems that are connected to a network infrastructure secure. An abstract definition might be a set of rules and actions to maintain the protection of the Confidentiality, Integrity, and Availability (CIA) [2] throughout a network environment, thus a set of mechanisms to avoid unauthorized traffic to move around the system.

There are multiple aspects of network security, but this project will mainly discuss secure connections and intrusion detection systems.

2.1.1 Firewalls

Firewalls are systems that monitor and analyse incoming and outgoing connections based on defined policies and rules [3]. This prevents

unauthorised access to a network since they are placed inline on the border between an external and internal network. There are multiple types of firewalls:

- Packet filtering firewall [4]: This is the simplest firewall solution, filtering the packet based on the IP and port of its source and destination. As it does NOT have any connection context, it is a stateless firewall. Figure 2.1 shows how you can block a specific inbound IP connection: the client might reach the server, but the reply will be blocked by the Packet filtering firewall.

- Stateful inspection firewall [4]: In contrast with the previous one, this firewall has the connection context, and it then keeps track of the connection and allows traffic if it's related to another packet or to another flow. A firewall with a stateful "deny-all" policy for incoming traffic will reject any incoming packet unless it is a response to an earlier outgoing packet, as shown in Figure 2.2.
  The three pictures show the "deny-all" policy in three different cases: the packet filtering Firewall would block all the connections from outside, the stateful firewall would allow incoming connection if it's related to a previous request, and finally, the third case where the stateful firewall would block the incoming connection due to missing request from the client. It is important to monitor the context of the network communication since it allows identifying threats based on where they are coming from and where they are going to.

- Proxy Firewall [4]: Adding onto the previous firewall, the proxy firewall is able to analyse the content of the packets. Thus, it can filter and distinguish valid packets from malicious ones.
  This extra feature allows engineers to protect and monitor incoming data and deny specific requests made for the external network. For instance, a proxy firewall can analyse the HTTP request and deny a POST request for a specific endpoint; the supplementary feature allows companies to set a more granular policy compared to the previous cases.

- NGFW (Next Generation Firewall) [4]: This final firewall adds the features of application awareness and IPS (Intrusion Prevention Systems). With application awareness, it comes into play the idea of QoS (Quality of Service) [5] and of IPS (Intrusion prevention system). QoS has been developed to prioritise the different types of network traffic. Using QoS permits prioritising time-sensitive and crucial time

over web browsing data for instance, ensuring that the traffic is delivered quickly and in a more reliable way. On the other hand, the IPSs help to filter out malicious activities before they are executed on the target host. They also efficiently work at detecting vulnerabilities and exploits. During the rollout of a new free-bug version of specific software the system might be exploitable, therefore to prevent these malicious actions, an IPS can be configured to block the exploitation in the traffic patterns.
This IPS can usually block traffic patterns, block the source address or alarm the administrator of an anomaly.

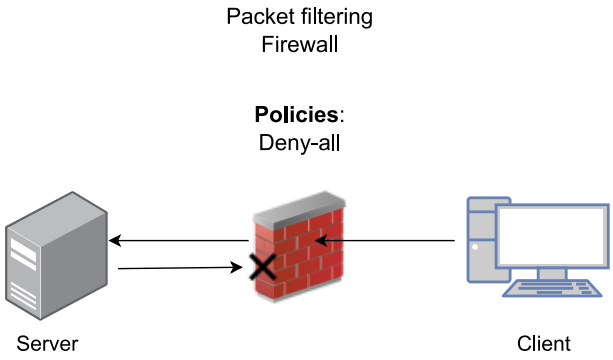


Figure 2.1: Packet filtering with a deny-all policy

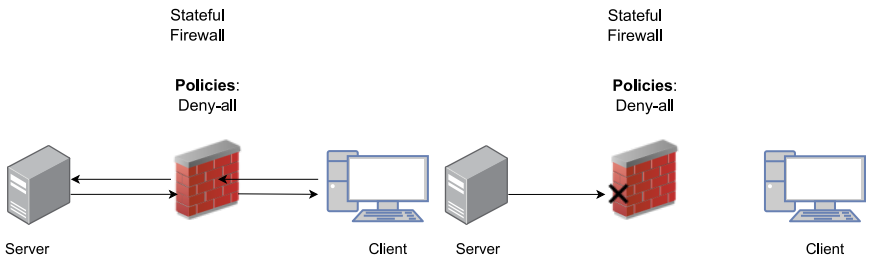


Figure 2.2: Stateful firewall with deny-all policy: left side scenario where the client starts the connection - right side scenario where server starts the connection

2.1.2 TLS and connection splitting

TLS is a cryptographic protocol used by applications to communicate securely across the network, ensuring integrity and confidentiality across the devices [6]. Although it has been extensively used in VPNs, IoT, and other

communication patterns over an insecure network like the internet, it is mostly used in web browser communication. The first TLS comes after the evolution of the SSL (Secure Socket Layer), which provides the same functionality with a weak implementation, standardized by RFC 2246 in 1999 [7], while the most recent version is 1.3 regulated by the RFC 8846 in 2018 [6].
Historically, the data were sent over the network unencrypted, and therefore, a person in the network path could easily check the network traffic and every credential, sensitive data and company secret bound to the packets. The monitoring of network traffic as well as the tampering, erasing, and rearranging of packets on the network posed additional risks for end users.

TLS uses a combination of symmetric and asymmetric techniques to deliver excellent performance and high levels of data security. The data is encrypted and decrypted using symmetric cryptography, when both the sender and the recipient are aware of the security key. If compared to asymmetric cryptography, this cryptography is more effective, faster, and only needs a smaller key for both the encryption and decryption phases. From a client perspective, symmetric cryptography is required to encrypt the data of the request and decrypt the response web page; from a server perspective, it is necessary to receive the request, which may contain personal data, and respond with a general or customized webpage for the users.

Connection splitting (TLS proxy) is a feature provided by firewalls to analyse the inbound and outbound connections of the network. Since practically all connections now use TLS encryption, businesses are unable to effectively monitor the traffic, therefore it was required to find a method to exploit this situation. In this architecture, the proxy intercepts the connection and functions as a man-in-the-middle or TLS server as opposed to end-to-end encryption from client to server. Therefore, there will not be a TLS tunnel between the client and the server; instead, all communication will be encrypted by the client up until the proxy, which will then decrypt and parse it before making a new connection to the target server as shown in Figure 2.3.

The response will follow the same path, it will be sent from the server over a TLS tunnel to the TLS proxy and over a new TLS connection to the client.
This architecture allows companies to deploy and configure multiple security systems and help to unfold security branches or prevent new vulnerabilities. Besides, it introduces downsides to the system, the confidentiality and integrity will be no longer guaranteed for the connection since the packets

might be changed and read by the man-in-the-middle, the TLS proxy.

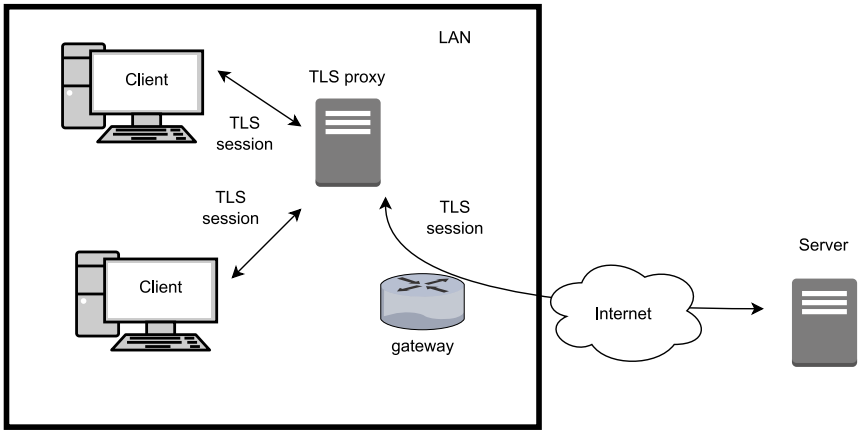


Figure 2.3: TLS splitting architecture

2.1.3 Security protection tools

IPSs are security protection tools that identify malicious traffic in the network. They are employed and embedded in NGFW and other modern firewall systems to monitor traffic from inside to outside the network [8]. In order to work efficiently, these security tools are most of the time combined with the TLS splitting, allowing a full plain-text inspection of traffic. The traffic cannot be analysed if the encryption is made in the application layer; in that case, the IPS cannot decrypt the data.
There are two different types of IPS:

1. Signature based: the system tries to match the signature of a known threat against the request. The drawback of this method is that it is able to identify only a known threat, or a variant of it, but it cannot identify new ones;

2. Anomaly-based: the system tries to find anomalies in the network by comparing a traffic span against a baseline. This method is widely used since it can detect unknown threats, and in recent years researchers exploited machine learning techniques to achieve better performance;

A further classification can be applied:

1. Network-Based IPS: It is positioned at the network perimeter and continually monitors all incoming and outgoing traffic;

2. Host-Based IPS: It is deployed in hosts and keeps track of the ingress and egress traffic from the host perspective.

IPSs are not infallible security measures, and there is typically a trade-off between false positives and real negatives: while skimming through all the warnings is very time-consuming, overlooking one may lead to an assaulter leaking data without the company being aware of it. Therefore, to improve the level of security, multiple methods can be combined to achieve better accuracy.

On the other side of the attack, a bad actor must manage to overcome the security mechanisms implemented, including frequently IPS and firewall, to achieve his goal. A strategy for the attacker's point of view would be to properly craft, obfuscate, or hide traffic parts in the traffic pattern; this requires a thorough knowledge of the system and how it operates for a given packet.
One further layer of security may be injected by the host, which for instance would include an endpoint security or an antivirus, so that the IPSs do not represent alone the protection shields built by companies.

2.2 Network communications

Network communications can be defined as a set of protocols, or rules, that allow computers to exchange information with each other, independently of the operative system they are running on [9].
There are four main entities in each network communication: a client, a server, the data and a protocol. The client is the entity that requests a service on the network, while the server is the entity that tries to fulfil the request made by the client. The data is the object of the request, contained in the response from the server, while the protocol is the communication method used for transmitting the response data.

2.2.1 Covert channels

Covert channels were firstly defined by Butler W. Lampson [1] in 1973 as channels "not intended for information transfer at all", yet exploited for unauthorised data leakage. They represent an attack in computer security where an attacker communicates with another party in a manner that bypasses normal security controls. This type of communication can be used to transfer sensitive information, or to exfiltrate data from a target system. They can take many forms, such as exploiting unused or hard-to-detect network bandwidth,

hiding data within seemingly innocent packets or files, or even using physical storage devices to physically transport information. Covert channels are often difficult to detect, and defending against them requires a multi-layered approach that includes regular security audits, monitoring, and continuous system updates.

2.2.2 REST API

Representational State Transfer, or REST, is a protocol for inter-application communication [10] and it allows exchanging of data among applications over an HTTP(s) connection as shown in Figure 2.4. It was designed based on the following constraints: (1) requests should identify resources, (2) the client can receive, modify or delete a resource, (3) the server does not store information about previous connections/requests, (4) cacheable resources and (5) the possibility to create layered application [11]. An application programming interface created on top of the REST architecture is known as a REST API.
A REST request is composed of four main parts:

1. URL Endpoint: a service implements a REST API endpoint for exchanging the data and this is represented by a URL (domain, port, path) - blue square in Figure 2.4;

2. HTTP method: different HTTP methods can be used by the endpoint to map them to a different action - GET (read), POST (create), PUT (update) and DELETE (delete) - light blue square in Figure 2.4;

3. HTTP headers: information sent in the HTTP request, like cookies, authentication tokens and other standard HTTP headers - purple square in Figure 2.4;

4. Body data: data sent with the HTTP request. It allows defining objects transmitted with a POST request for instance - green square in Figure 2.4.

In most cases, the body data is used from the POST request to query or create objects on the server side. It can be defined using HTML, XML, raw data or JSON. The JSON format is the most common one, since it allows the definition of a dynamic data structure and it can be easily parsed by the server due to its key-value structure. It is then the duty of the server to properly extract the needed data from the JSON body.

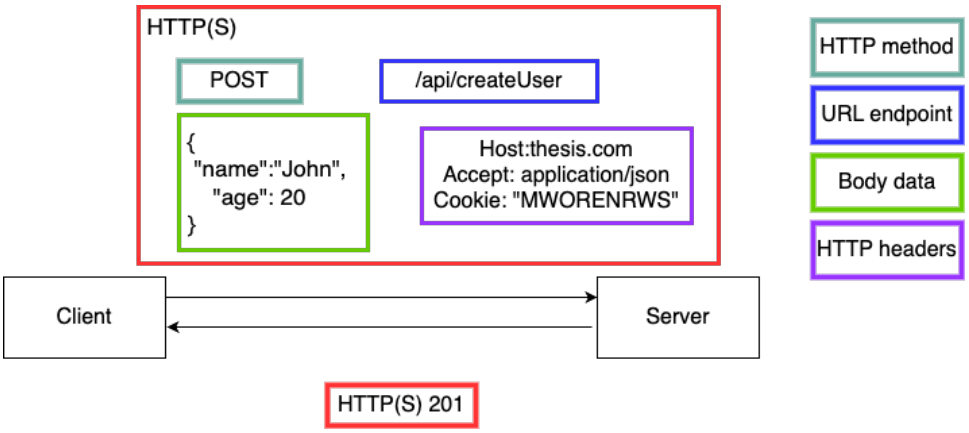


Figure 2.4: REST architecture

2.2.3 Cloud Command and Control (C2)

Command and control, also known as C2 or C&C, is a technique to communicate with compromised devices over a network using a hard-to-detect channel [12].
Malicious actors exploit the channel to send instructions to the compromised devices, which will then be executed by these devices upon receival and their results will be sent back. In this way, the attacker gains complete control of the compromised device and can execute any code. The network architecture of the C2 consists of the malware in the compromised device acting as a client, and the attacker's device acting as a server; relatively known as Zombie and MasterBot by the security community. The malware can be horizontally spread across the compromised device's network creating a botnet, see Figure 2.5. In such scenario, all the compromised devices are still acting as clients, and the attacker is a server that instructs and controls them.

Once the malware is spread to a new device, there is a beaconing process to let the server be aware of the new device. These continued requests allow the attacker to know of the existence of the new machine and periodically poll the server for new commands that the compromised device needs to execute. The C2 channels are used for fetching internal information as well as exfiltrating data storage from the environment. This allows stealing data from companies, for example, over covert channels (Section 2.4.1) or some hard-to-identify communication methods like out-of-band channels.

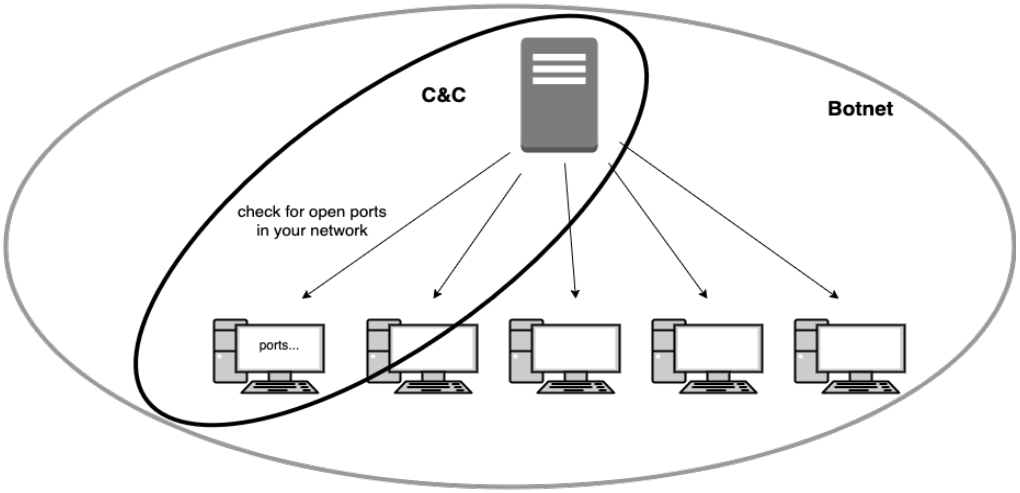


Figure 2.5: C&C and botnet schema

2.2.4 Cloud and applications

With the advent of the cloud, many applications have been moved from an enterprise solution to a cloud-based solution. Cloud computing is an on-demand computer resources available over the internet, offering a fast, flexible and scalable service [13]. Therefore, instead of using in-house servers to provide services to clients, the application runs on the cloud infrastructure offered by the provider. As the cloud allows for a full horizontal scalability, each response to a client's request might come from different physical servers, yet looking like they are coming from a single one.

2.3 MS Teams

The malware in this project is designed on top of the cloud application Microsoft Teams (Section 2.2.4) and the covert channel (Section 2.2.1) will be established between the attacker and the victim machine on the Teams architecture. It follows a commands and control architecture (Section 2.2.3), where both channels (the attacker-to-victim channel and victim-to-attacker channel) are hidden inside the client-server messaging exchange.

Microsoft Teams is a messaging app that allows to share the workplace, communicate, and exchange information among users. It continuously

provides additional plugins and features, with new updates, to enhance the user's communication experience. The application recently became popular due to the shift from on-site work to remote work, becoming fundamental in the new hybrid workflow of many companies; this new key role played by the application made it gain unsolicited attention also from hackers and malicious actors, trying to find ways to penetrate it.

The application is usable across many devices and operative systems, allowing it to be quite common among worldwide companies, especially for businesses where it is not possible to create their own messaging application. We can then highlight a few features provided by this application:

- Chat: message with someone, a few people, or a group;
- Call: call/video-call someone, a group, or join external calls by links;
- Teams: group across the company to share resources;
- Sync: synchronisation with other Microsoft Apps.

Looking in detail into the Microsoft Teams system, we could see the possibility to send messages to users using a REST API, with the proper request and authentication we are able to send text messages to another user. This message will not be directly forwarded to the end user, instead, the text message is sent to a Microsoft server and later pushed down to the client using a WebSocket.

WebSocket is a protocol used for bi-directional and real-time communication between a client (typically a web browser), this allows to receive messages from the server without any prior request from the client.

The diagram presented in Figure 2.6 illustrates an established channel utilising WebSocket for client and server notifications, specifically for notifying the arrival of a new message. The REST request to send the messages is also visible in the figure, where the client will send the information without using the WebSocket.

The calling system works a little bit differently: There are mainly two options either a peer-to-peer connection or a client-server connection. The first option establishes a peer to peer connection between the two clients using the STUN protocol. STUN (Session Traversal Utilities for NAT) is a protocol

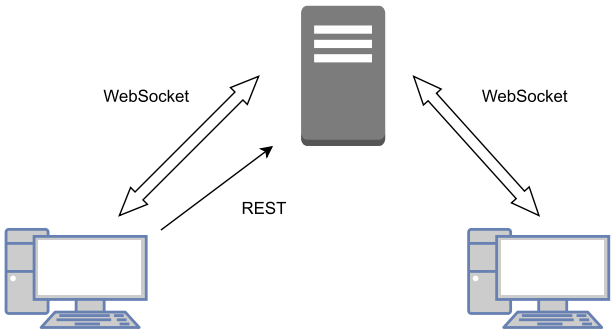


Figure 2.6: Teams messages schema with the involved protocols

used to assist in the discovery of network address translation (NAT) devices and to facilitate communication between clients behind a NAT. The STUN protocol allows clients to determine their public IP address and port, as seen by the outside world, as well as the type of NAT they are behind. STUN also has a variant called TURN (Traversal Using Relay NAT) that can be used when direct communication is not possible due to firewalls (Section 2.1.1) or other network restrictions (Section 2.1.2). In this case, a TURN server acts as a relay for the client's communication, which is the protocol used in the second option. It establishes a client-to-server communication between each client and the Microsoft servers, the last one will act as a relay server, transmitting information between the two users.

Regardless of the options, the protocol used to carry the call data is WebRTC (Web real-time communication), which enables real-time communication directly in web browser without the need for plugins or additional software. Figures 2.7 and 2.8 describe the major difference in terms of architecture between the two options. In the scenario above 2.7, the WebRTC channel is established between the two peers allowing a connection without any third entity. In the scenario below 2.8, the firewall (see section 2.1.1 for more details) prevents the peer-to-peer connection due to security concerns and the system will fallback into a TURN scenario, where the webRTC channel is established with a third entity relaying the traffic to the other client.

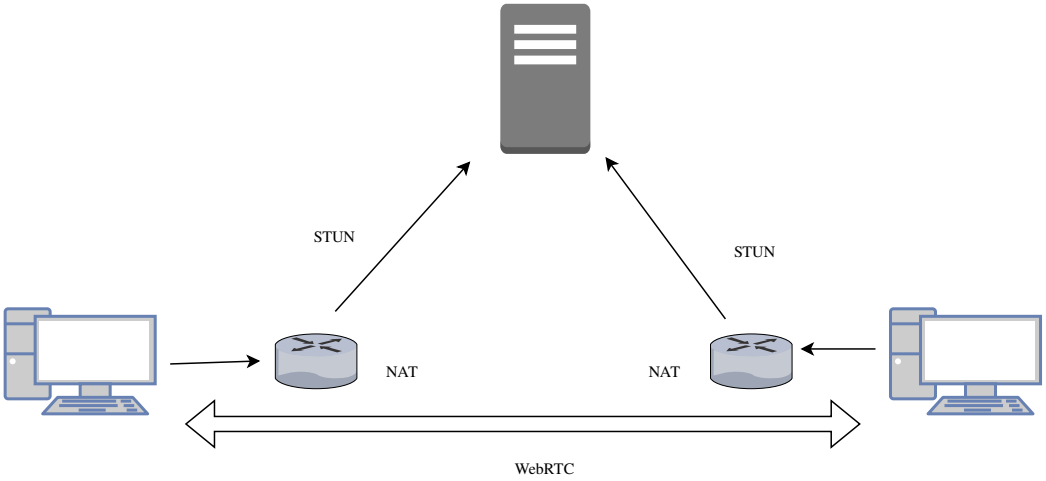


Figure 2.7: Teams Calls using STUN

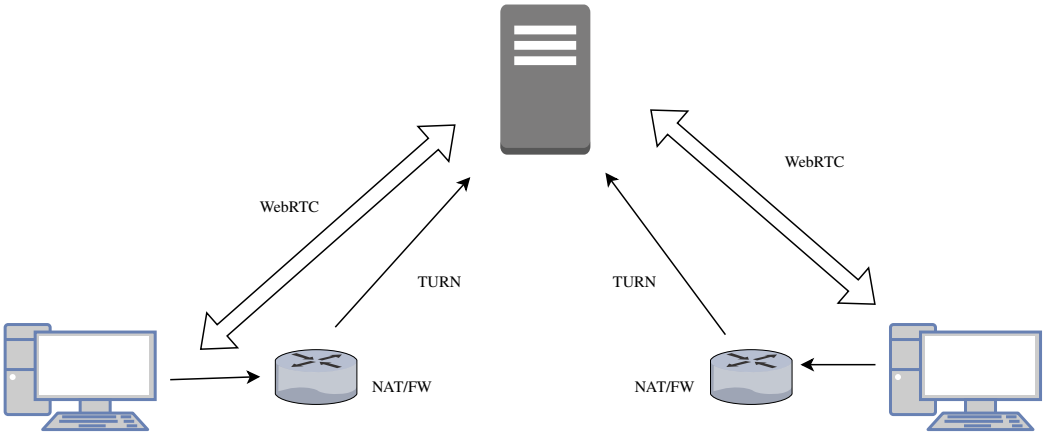


Figure 2.8: Teams Calls using TURN

2.4 Related work

This section refers to the literature on the definition and categorisation of covert channels, as well as the latest detection techniques employed to identify them. The primary research emphasis lies in the employment of machine learning, statistical methods, and traffic anomalies for detection, although only a limited number of studies explore the detection of covert channels within a cloud-based architecture. Typically, detection is established when a sufficient amount of data is available for a particular scenario, such as DNS or TCP covert channels, resulting in a gap in security detection systems when the cloud is involved.

2.4.1 Types of covert channels

Traditionally, covert channels may be discerned into two main categories: (1) storage covert channels and (2) timing covert channels. These categories are defined based on the method to encapsulate and transmit information to the receiving side.

Storage covert channels

Storage covert channels [14] allow communication with one process writing to a storage location and another reading from it.

It has been shown that the cellular voice channel in Android can serve as storage covert channels to leak information [15]. The designed rootkit for Android permits outbound connection to transmit data in "speechless" calls, avoiding the notification to the phone owner. But storage covert channels attack may also occur in cloud storage applications, such as Dropbox. The first work to allow two endpoints covertly exchanging data through Dropbox in fact suggested two methods involving the tampering of files: in the "Renaming of File", the filename is exploited as the information carrier, while the "Alteration of File" method exploits the content of a file as carrier [16]. Both methods resulted to be robust and performing, leading the way for next-generation malware for cloud storage applications [16].

Timing covert channels

Timing covert channels [14] allow communication through measurements of the timing of packets/processes to represent the communication being sent over the channel.

An example of attack carried out through a timing channel is present in the past work of M. Torkashvan and H. Haghighi [17], in which the Wi-Fi signal is jammed based on the message that one wants to send as a bit sequence, and the data is transmitted outside the network through this time interval.

2.4.2 Cloud Command and Control

Covert channel attacks using the Cloud Command and Control can occur when an attacker uses the communication channels within a cloud computing environment to transfer sensitive information or to exfiltrate data.
A cloud environment could be exploited for instance through one of the following:

- Abusing cloud storage: an adversary can leverage the storage mechanisms within a cloud environment to transfer sensitive information, such as by hiding data within seemingly innocent files or packets;
- Abusing network bandwidth: an adversary can take advantage of unused network bandwidth of the cloud environment to transfer data;
- Misusing APIs: an adversary can misapply the API to transfer information outside the target system.

Compared to old-fashion attacks, a BotCloud attack needs a lower budget and fewer skills to be set up [18, 19].

2.4.3 Detection of covert channels

Nowadays, systems' security policies are frequently violated through covert channels being leveraged to retrieve sensitive information and packet data.

Traffic normalization [20] is one of the techniques used to counteract this phenomenon, and it consists in enforcing a constant stream of fixed-length data packets, using padding when needed. Exploiting this technique, a traffic normalization protection tool could annihilate storage covert channels that based on changing the lengths of transmitted packets [21].

Code augmentation techniques in the Linux kernel, like the extended Berkeley Packet Filter (eBPF), may also be exploited to gather useful data revealing the presence of covert communication. This information may include system calls, page faults, memory occupation and network packets [22]. eBPF is a mechanism for Linux applications to execute code in the Linux kernel, and running your code in eBPF allows full access to all hardware, like CPU, memory, network cards, etc. Analysis metrics such as the Kolmogorov-Smirnov Test and entropy may also be a valid option to detect covert channels [23]. These last techniques detect the anomalies by comparing the traffic distribution in a given time frame against the baseline traffic distribution of the system.

Most covert channel detection methods rely on discerning normal traffic from the abnormal one that characterises a data leak. However, most traditional methods are failing to detect covert channels with only slight deviations of traffic, which are constantly introduced by new efforts in research.

In this regard, recent studies suggest that the usage of machine learning techniques may enhance traffic detection capabilities in these cases [24]. However, there is no state-of-the-art machine learning or deep learning model that is effective in every case, as each approach has its own strengths and limits [25]. In fact, such methods were observed to struggle to perform accurately and reliably [26]. However, this under-performance may be due to the absence of representative data, without which such methods cannot be properly trained and fine-tuned, since the available datasets are characterised by uneven and outdated content [25]. As the main idea behind most of the relevant detection methods is to discern a statistical variance from the median of the normal traffic pattern, no machine learning technique is then able to generalise for such a task, and it would anyway require extensive and continuous training to constantly adapt and learn the network traffic pattern even for a specific case [26].

Among the machine learning techniques investigated over the years, SVM (Support Vector Machine) was found to be very effective in the IP/TCP protocols, while storage covert channels in DNS protocols were detected best by decision trees [27]. Nevertheless, SVM is unable to detect non-linear chaotic data in a TCP packet if the Sequence Number, the TCP Control Flag or the TCP checksum are not provided [28]. In such cases, a deviation score computed over TCP headers in a two-stage analysis process outperformed SVM while requiring less features [28].

When it comes to BotClouds, cloud providers should try to actively detect the anomalies caused by the MasterBot, identifying and responding to its suspicious activity [19]. For instance, a cloud provider could identify the traffic way between the MasterBot and its bots, which follows the equation $T = N \times S$ with N referring to the number of bots and S being the average size of messages [18]. As in non-cloud environments, entropy may also be leveraged by looking at the randomness of the communications between the MasterBot and its bots [29].
If the bots in the network are assumed to reply within the same timeframe, botnet communication can be detected through “behaviour similarity”, as leveraged by BotSniffer [30] without prior knowledge of the communication pattern.
If the Command-and-Control input commands are frequent enough, the computers in the network could be grouped together in small clusters, and the variance could be checked for each server-group pairing, regardless of whether

the botnet communication is encrypted or not [31].

2.4.4 Countermeasures

Once the covert channels has been detected, a countermeasure is necessary to prevent data leakage. However, such countermeasures are often subject to trade-offs, based on their ultimate functional goal [32]:

- security vs. quality, as enforcing security could reduce the quality perceived by the end-user;
- accuracy vs. performance, as the accuracy of the limitation (or elimination) may impact the performance due to overheads and an excessive amount of resources requested;
- complexity vs. cost, as complex mitigation methods may require expensive software, hardware and maintenance costs;
- blockage vs. functionalities, as a required blocking of a protocol for a network or host could reduce functionalities;
- security vs. risk, as in some cases it may not be possible to eliminate the covert channel, but one could limit the bandwidth to prevent data leaks.

There is no single countermeasure that can be effective enough against BotClouds, and cloud providers should come together to block them [18].

Chapter 3

Methods

This chapter addresses the methodology and the environment in which the project will be carried out. Additionally, this chapter defines the threat scenario and all the involved actors throughout the project. Lastly, the data collection is defined and how the data are evaluated against the below defined properties.

3.1 Research Process

Previous work has been done on malware exploiting covert channels with high bandwidth in cloud applications. An example of it is the Dropbox case study, where the channel was built on the "Renaming of the File" or on the "Alteration of the File" methods [16]. However, it is possible to carry out such attacks also with a lower bandwidth, as in the case study on Skype [33], where the covert channel is established on top of the Skype call's inter-packet delays.

This research wants to follow the same path as above. In both aforementioned papers, the starting point consists of a security assessment of the target application. Thus, it is important to define the platform's behaviour and every interaction it has with the servers. After evaluating and identifying the weaknesses of the architecture, a concept for the malware can be developed to exploit these findings; a good amount of time needs to be spent on this analysis in order to achieve a good outcome at the end of the project.

The malware in this project leverages covert channels, or channels used in an originally unintended way, to communicate with the malicious server. In this case, these channels are meant to be valid Teams communication channels,

but they are used by the malware to exchange information and elude the system in place to exfiltrate data.

3.2 Research Paradigm

The research follows mainly an empirical approach, trying to experiment with possible ways to extract data from a cloud platform such as MS Teams. The research for weaknesses is performed in a black-box manner. This initial phase consists in proving the existence of a vulnerability. According to Dijkstra's ACM Turing Lecture in 1972 [34], program testing can effectively show the presence of vulnerabilities, but it cannot prove their absence, unless a proof of correctness is provided. It then follows that it could be possible to find a security vulnerability in the system or infrastructure, as no convincing proof of correctness is provided.

Most companies do not publish the source code of their applications, and Microsoft is inclined to this philosophy, although there are open-source projects from Microsoft like VS Code (the code is open source, although the Microsoft build includes additional proprietary code). Their reason can span from intellectual property to competitive advantage or security concerns. Avoiding publishing the source code makes finding security vulnerabilities more challenging, as the latter can be used by hackers to manipulate the behaviour of the software and steal data.

3.3 Research Strategy

This research primarily focuses on the communication between client and server throughout each application phase. The core application requests should be identified, and a trial-and-error approach can be carried out on the request parameters by changing them and observing how the application reacts. It might be interesting to modify the order of the requests, as well as block logically connected requests and forward only a subset of them. This approach can then be used to research unexpected behaviour of the application or induce unpredictable states in the application.

Given the complexity of the system and application, it is crucial to narrow down and define the research scope. In modern web applications, there are

a hundred REST requests passing through the client and the server, often for minor resources like images, motions, and graphical effects. Filtering these requests via a proxy, such as Burp suite, or other similar tools is necessary to limit the traffic analysis.
Alternatively, a more efficient strategy involves rebuilding a minimal Microsoft Teams application with the company-provided SDK to efficiently analyse the client's traffic. This ensures that graphical content and JavaScript requests are excluded from the traffic, thus separating the functional requests from the not functional requests.

Both of these strategies are pursued in this work to obtain as many insights as possible and to achieve a better coverage of the application.

3.4 Threat modelling

The selected threat scenario assumes an adversary, who aims to exfiltrate information from a company network, and an already compromised machine in the network, that can be infected with the malware in different methods (i.e., phishing or social engineering). The victim is part of a company and has access to internal network resources with his compromised machine (Red area in Figure 3.1), while the adversary might be outside the network and not be related to the company in any way (Green Area in Figure 3.1). The victim has a Windows computer and with a standard configuration setup, no additional software is needed in order to achieve this goal. There are no requirements on the client (victim's computer) side apart from the Teams client installed.

In the first part of the threat scenario, when the adversary-to-victim communication takes place using a covert channel, the adversary instructs and executes commands on the victim's machine. In this situation, the attacker wants to establish a connection from outside to inside the network. We assume that the firewall blocks all inbound connections and incoming packets can go through the firewall only if they belong to an existing connection. In the second part of the scenario instead, the victim-to-adversary communication takes place with a covert channel allowing the forwarding of the query results. This threat scenario then follows a command and control architecture (Section 2.2.3).

Both of these channels must be stealthy to avoid and escape the security system of the victim's company. The company's security system in this scenario is configured with an NGFW (Section 2.1.1) with an inline IPS

(Section 2.1.3) represented as “Company firewall” in Figure 3.1. These last two components control and block signatures against malware or evasion methods. Furthermore, in this case, the IPS is set to anomaly detection in order to prevent data leakage of the network, according to the state-of-the-art methods as described in section 2.4.3. Thus, any kind of data anomalies will be detected by security systems.
To avoid data exfiltration on the system, there is a TLS proxy that sits in the middle between each connection (Section 2.1.2). It will apply a TLS splitting method to decrypt all the incoming connections from the client and encrypt again the connection until the destination server. The TLS proxy would help the IPS due to the fact that the IPS will work on plain text data and not on the encrypted ones.

Concerning the OS, this threat scenario consists of only Windows client machines in the compromised network, with the red area in Figure 3.1 and Windows Defender active in these machines. The tenant, configured by a system administrator in the network, provides authentication and authorisation, and the client machines are also equipped with intercommunication and management tools. Among these, Microsoft Teams is installed for internal communication. For Microsoft Teams, all the users are enforced to use the electron JavaScript application version rather than the browser version, due to security concerns with the browser.
Electron JavaScript is an open-source framework for building desktop applications using web technologies such as HTML, CSS, and JavaScript.

Finally, a few IPs need to be whitelisted in order to keep a video call smooth and avoid packet loss during audio and video sharing for Microsoft Teams communication [35]. This communication is established between the red and the blue area in Figure 3.1. The purple area in Figure 3.1 consists of a cloud provider offering a server to the attacker, the attacker needs to set up this server to be reachable from the outside (The attacker’s computer is not reachable from the outside due to the NAT and the Firewall).

3.5 Data collection

After proving the existence of covert channels, the network traffic will be collected to compare and evaluate them. As mentioned in section 1.5, the goal is to design a command-and-control malware using the infrastructure of Microsoft Teams.

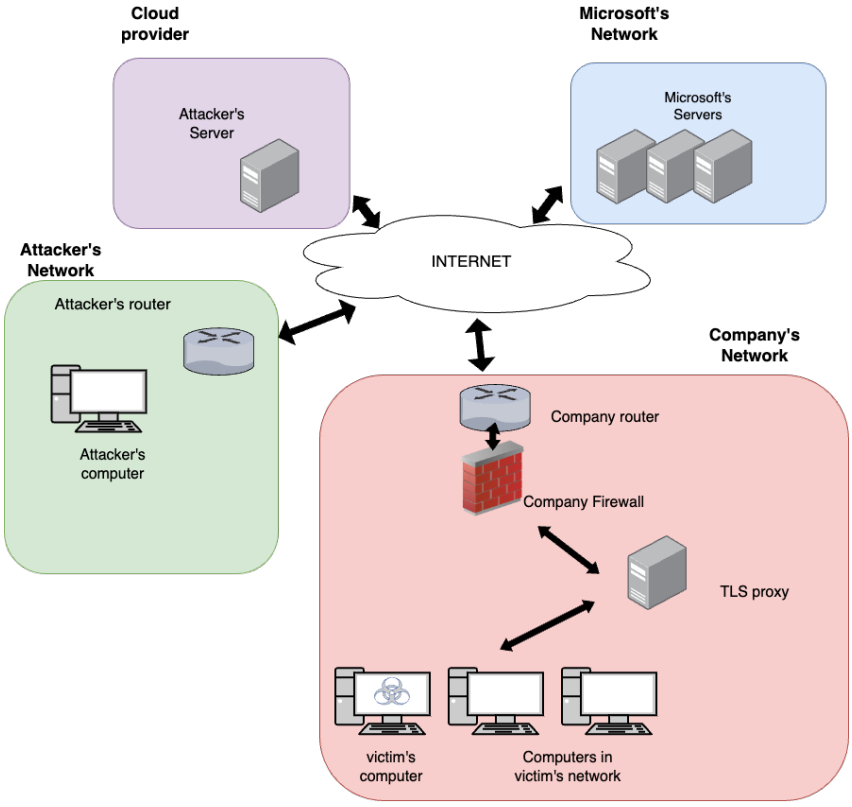


Figure 3.1: Threat scenario

There are two channels in the command-and-control architecture: the instruction and the extraction channel as mentioned in the section 2.2.3. The first channel consists in measuring the time, capacity, and bandwidth to send over the instruction from the attacker's machine to the victim's machine. Later, an evaluation of the robustness of the channel can be performed, taking into consideration the entities and the external factors that can change the behaviour of the channel. The second channel, from the victim's machine to the attacker's, requires the same assessment. Further details about the evaluation can be found in section 3.6.
The channels rely on the underlying infrastructure as mentioned in section 3.5.1 and all the measurement depends on the status of it. Re-evaluating the channels after weeks/months could lead to different results and different outcomes. The time to execute the instruction is excluded from the measurement; otherwise, the latency time would be highly coupled with the execution time of the command.

3.5.1 Targets and actors

It's important to consider that in the evaluation not only the attacker's and victim's machine are involved, but also the latency of Microsoft servers, the carrier network and the network devices like Firewalls and proxies.

3.5.2 Assessing reliability and validity of the data collected

The collected data is gathered in Zürich, Switzerland during the winter/spring semester of 2023. Since most of the data taken relies upon the network bandwidth and connections between the machines and the central Microsoft server, the evaluation might or might not be reproofed in the future. Therefore, the evaluation depends on three factors: machine specifics, network, and Microsoft server. The last element in particular will be keener to improve over time and it will speed up the whole system. In order to achieve better accuracy, the evaluation and the data will be collected on multiple non-adjacent days to avoid unsuspected failure, latency, or packet drop on the Microsoft side.

3.5.3 Data analysis technique

Quantitative data analysis will be performed for the evaluation. The collected data will result from malware performance analysis at the end of the experiment, and statistical techniques will be applied to profile the malware according to the evaluation framework 3.6.

3.6 Evaluation framework

There exist multiple evaluation systems to identify and track the characteristics of covert channels.
The one proposed by Torkashvan and Haghighi [18] evaluates stealthiness, effectiveness, efficiency, and robustness: stealthiness refers to the impossibility to detect the ongoing communication between the bot and the botmaster; effectiveness refers to the destructive power of a botnet (highly coupled with the dimension of the botnet); efficiency represents the latency between the client and the server; robustness indicates the reliability and stability of the botnet architecture. Another evaluation framework refers to the bandwidth as one of the major aspects of covert channel evaluation, and it was proposed by Gligor in his book titled "A guide to understanding covert channel analysis of

trusted system" [36]. Gligor suggests that "the maximum attainable bandwidth of any channel must discount both noise and delays, and must assume that only the senders and receivers are present in the system", and so this evaluation framework takes into account also the external factors affecting the bandwidth.

The evaluation framework defined in this thesis follows the guidelines of Torkashvan and Haghighi as they assess some general criteria of the covert channel. An analysis of the bandwidth is then added to it, based on Gligor's study. Additionally, robustness is redefined to evaluate the stability of the channel rather than of the botnet architecture. Therefore, the work in this paper is evaluated according to the following criteria:

- Bandwidth: The amount of data that can be sent per second using a covert channel. To eliminate the elapsed time of process switching in the same system, the bandwidth needs to be evaluated with the least number of processes running on the machine; the measurements are repeated several times, and the minimum value is then selected [36]. A higher bandwidth means a higher transfer rate and a more effective covert channel.
- Efficiency: The amount of time it takes to transfer one data unit using a covert channel.
- Robustness: The ability of a covert channel to maintain its functionality throughout its lifespan. A more robust covert channel is less likely to lose or not deliver packets to the receiver.

It is important to note that evaluating the performance of a covert channel attack requires a thorough understanding of the target systems and networks, as well as the methods and techniques used by attackers. The results of the evaluation can be used to identify areas for improvement and to inform the implementation of security measures to defend against covert channel attacks.

3.7 Software Tools

The following are the used software to perform the security assessment of Microsoft Teams and to design the malware:

- VMWare: to manage and use virtual machines. In the testbed, two virtual machines are used to perform the security analysis and the evaluation.

- Burp suite: to explore, brute-force, and analyse the API. This tool can be useful in case a full analysis of the outgoing and incoming requests needs to be performed. The intruder functionality allows multiple brute force techniques to test user-given values and variables passed from the client to the server.
- Python3 and lib: to develop a proof of concept of the malware, to analyse the traffic, and to perform the evaluation. Additionally, python is needed to create customized scripts to run against the Microsoft infrastructure. This programming language is chosen for its versatility, even though other program languages can be used as well.
- Wireshark: to analyse the traffic between client and server. As the tool Burp, Wireshark can be used to fully filter all the content leaving the client to the server. Contrary to Burp, wireshark analyses the packets in the network later and not the communication in the application layer.

Chapter 4

Malware design

In this chapter, we analyse the client-server communications, before moving onto the client application's behaviour. Analysing the packet flows and the requests can simplify the research and lead to a better understanding. Among the various requests, we can see what information the MS Teams client application needs to know once it is started up and what information need to be pulled from the Microsoft server. If the application client stores data, it is necessary to verify what is stored and what would happen if the file system changes. If the application client has to request some information from the server, it is essential to catalogue the API and its services, how to communicate with them, and if something is hidden inside the traffic.

The client side of the Microsoft Teams application is an electron JavaScript app or its browser version. After analysing and comparing the server interactions carried out by both of these versions, their similarities in the exchange with the server would help in reverse engineering the application.
This chapter gathers and summarises all the findings throughout this work. Apart from testbed at Section 4.1, where it is described the setting used to reproduce the threat scenario, the rest of this chapter provides detailed descriptions of the MS Teams assessment, weaknesses, and covert channels discovered during the thesis.

4.1 Testbed

As a testbed, we utilize two virtual machines, one running Ubuntu OS for the attacker and the other running Windows OS for the victim. Additionally, we use a virtual machine in Azure to create a web server available from the

internet. It is important to note that the malware is designed specifically for Windows OS and may not be compatible with a Linux OS, as discussed in the preceding Chapter 1.7. It involves indirectly other services in the testbed, including the Microsoft Azure services and the Teams server, which are needed relatively for Authentication and information exchange.
To access the Microsoft services, the two clients have access to the Internet, but with a firewall sitting in the middle to block all the connections between the two. The testbed uses two separate Azure tenants, which allow businesses to configure a single sign-on system for their employees and manage users with different roles and scopes within their business domain. By using the admin webpage of the admin tenant, it is possible to configure Microsoft services (i.e., MS Teams, Outlook, SharePoint) for the domain. The next step involved creating test users in two different tenants and applying the same default configuration for both of them. However, it is necessary to grant access to MS Teams for the users in the domain, as it is not included by default.

The firewall is set up on the Windows side such that it blocks all inbound communication between the two virtual machines, and no incoming connections are possible to be established if a request is not made by the network client beforehand. This security measure is essential to maintain integrity and safeguard against potential security threats. An extra server is needed to accomplish one of the covert channels attacks, and so we will utilise an Azure Virtual Machine in order to set up a listener later on in the experiment.

4.2 MS Teams analysis

This section will discuss the analysis and observations made throughout the thesis on the Microsoft Teams systems. We outline the unique behaviours of the system, and it provides insights for the final design. The first pattern involves analysing network patterns by profiling the traffic generated by MS Teams. In the second part, we will discuss the undocumented behaviours of the MS Teams application and specific message exchanges between the client and the server.

4.2.1 Traffic analysis

In the initial analysis of the MS Teams client, uploaded bytes, from the client to the Microsoft server, were captured over a specific time frame in three usage

scenarios: the standby phase (Figures 4.1 and 4.2), where the user does not interact with the client; the messaging phase (Figure 4.4), where the user interacts with another user through chat; and the call phase (Figure 4.3), which involves establishing a call between two users. The data sent from the client to the Microsoft server is captured during real situations, thus none of the traffic is crafted or artificially created.

The data are taken using a profiling program for measuring the amount of data in upload and download per application in the system. The application is called GlassWire endpoint security, a project driven by Jon Hundley.

In Figure 4.1 the upload bytes are displayed over a time interval of 400 seconds. It is evident that only a few bytes are transmitted, following a repetitive pattern during the timeframe. This pattern is due to the WebSocket activities, which involve sending ping/pong messages to ensure that the other side is active. In a WebSocket connection, ping/pong messages are used to keep the connection alive and ensure that both the client and server are still available and responsive.

In the middle of the plots, we can see a huge spike of 2 KB sent, which is a REST request for pulling the configuration updates.

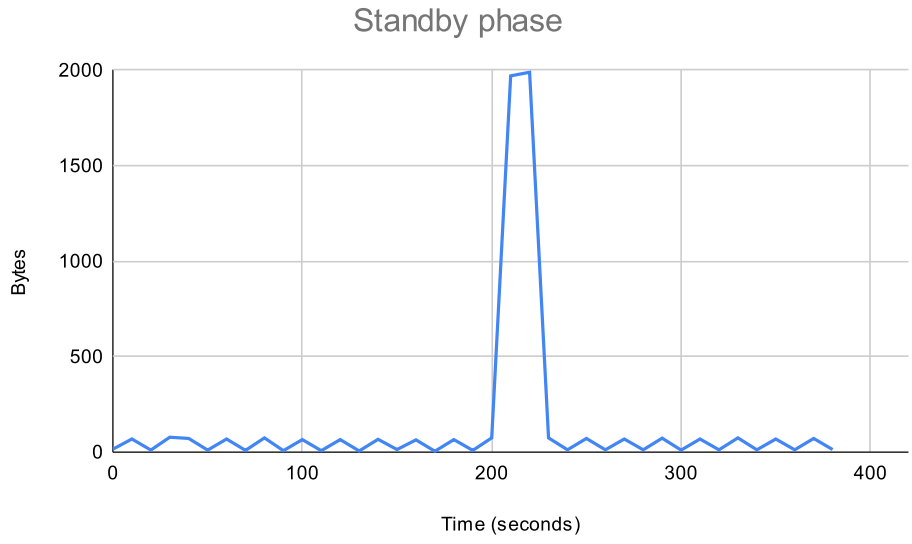


Figure 4.1: Standby phase without background activities. Bytes sent over 400 seconds with one second sample size

Figure 4.2 emphasises on some of the background activities executed by the Teams client. These activities include REST requests, that are unlikely to occur, as they may come from a complete or partial refresh of data fetched from the server. The activities observed in this scenario are considerably higher compared to the previous one, with peaks reaching up to 40 KB, 60 KB, and 124 KB per second.

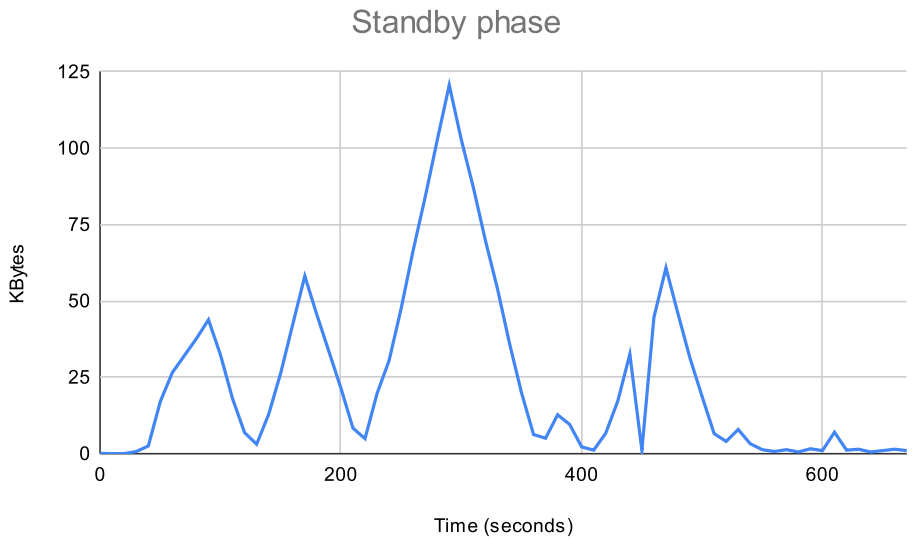


Figure 4.2: Standby phase during background activities. Bytes sent over 600 seconds with one second sample size

Figure 4.3 illustrates the upload data of a call within a minute. The peak upload speed reaches approximately 900KB, and stabilises at 800KB per second. By leveraging this pattern, we can optimize the traffic pattern for data extraction. If we know the characteristics of a Teams call traffic, we can shape the data extraction to match the same pattern.

The final scenario involves uploading a file into the system, and the file transfer time can vary depending on the data size. Figure 4.4 displays the amount of traffic sent per second during a brief period of time. The file took approximately 6 minutes to upload into the system, with an upload speed reaching up to 9 MB. This can significantly impact bandwidth for data extraction, but it is not sustainable for more than a few minutes to conform with the traffic pattern.

Figure 4.3: Call traffic between two users. Bytes sent over 750 seconds with one second sample size

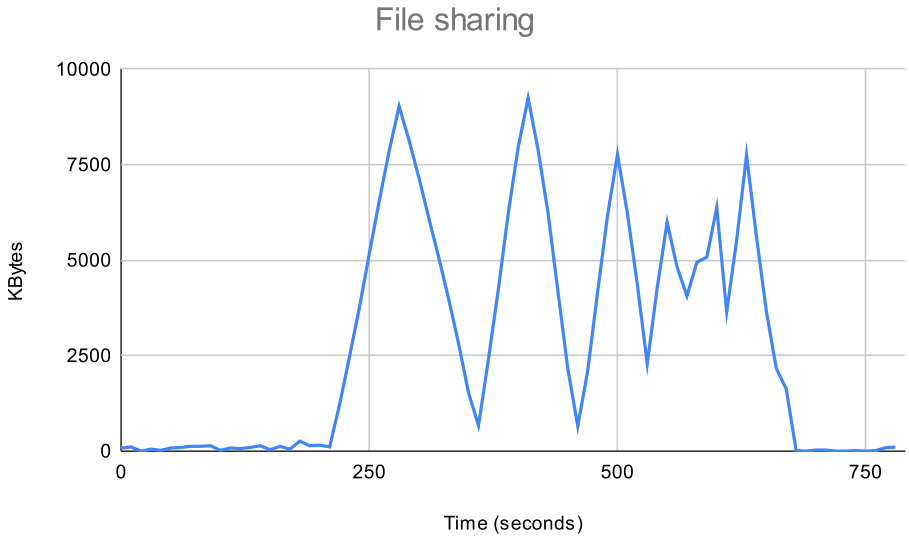


Figure 4.4: File sharing using the MS Teams chat. Bytes sent over 750 seconds with one second sample size

4.2.2 Peculiar functionality

MS Teams tries to keep up with the trend, so its functionality quickly expands with every update. As a modern chat software, there is the possibility to attach a webhook to a group-chats. A webhook triggers actions based on a specific event that occurs in another system or application, allowing for real-time communication and automation between the two. For instance, a webhook can be used to send event invitations or broadcast messages in general.
A peculiarity of this software concerns the upload process of a file into the chat. In fact, the system does not allow a direct upload onto the MS Teams system, but instead, it uploads the file first onto OneDrive, and its download link is later shared in the chat. OneDrive is a cloud-based file hosting and service developed by Microsoft, it allows users to store and share files with others globally.

When it comes to user calls, Microsoft Teams offers a functional advantage over other chat softwares. If a peer-to-peer communication cannot be established between two entities, MS Teams can fall back to a client-server call system using TURN as mentioned in Section 2.3. In addition to the architectural layer, MS Teams attempts to establish a UDP channel first, and defaults to TCP channel.
Suppose Entity A wants to communicate with Entity B without a peer-to-peer connection. In that case, the call can go through the TURN server, which will attempt to establish a UDP connection with Entity A. If this fails, Entity A will request a TCP connection. Entity B can then establish a communication channel with the TURN server. This channel can be a UDP one, even though Entity A initiated a TCP channel with the TURN. MS Teams thus allows for mismatched protocol call communication channels between two entities.

4.2.3 MS Teams user IDs

In MS Teams, each user has a UUID version 4 assigned to them. A UUID version 4 is a type of Universally Unique Identifier, that is randomly generated. It is a 128-bit number represented by 32 hexadecimal digits, typically displayed in the format of eight hyphen-separated groups of hexadecimal digits, such as “xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx”. The “4” in the third group of digits indicates that it is a version 4 UUID. It is commonly used in computer systems and applications to identify unique entities, such as objects, users, or transactions. Whenever a user wants to communicate with another one, the system combines the two UUIDs. Using this technique, it inherits all the

properties from the UUIDs: each chat will have a unique ID, no chat will have a double and it can easily decompose to retrieve the single users in the chat.

4.2.4 WebSockets connection

WebSockets play a crucial role in MS Teams, serving three main functions as revealed by the analysis: 1) verifying the client's status with ping pong messages, 2) receiving messages from other users, and 3) receiving notifications of incoming calls. With regard to the ping pong function, messages are continuously exchanged between the client and server throughout the day to ensure that the connection is not lost. The messages are plain text sent through the channel, as well as some call information. In MS Teams, there are two different WebSocket, one serving the notification for the incoming calls, while the second one serving the notification for messages.

4.2.5 Message flows

This section illustrates the major flows of chat messages between users. A chat message consists of an exchange of information performed using REST requests as shown in weakness 2.2.2 and consequentially fetched through Websocket. There are three main types of chat message flows: create, replace, and delete. In order to perform any of these actions through the API `https://amer.ng.msg.teams.microsoft.com/`, the *skypetoken_asm* token is needed. The method used to retrieve such token from the computer will be described later in Section 8.

Create a message

Sending a POST request to the endpoint `https://endpoint/v1/users/ME/conversations/<chat_id>/messages` with the body template in Listing 4.1 allows a user to send a message to the MS server. The message will then be downloaded by the receiver's WebSocket, as discussed in Section 2.3.

Listing 4.1: JSON for message creation

```
{
  "content": "<text␣message>",
  "contenttype": "text",
  "messagetype": "text",
  "clientmessageid":<clientmessageID>,
```

```
    "imdisplayname":"",
    "properties":{
      "importance":"",
      "subject":""
    }
}
```

A combination of the sender's and receiver's IDs (Section 4.2.3) is placed instead of "chat_id" in the API to define the message's receiving side. Before sending out the POST request, define a "clientmessageID" and store it to allow modification later.
The Server needs to reply with a 201 HTTP code [37] and the response body would contain an ID used in the message replacing. A result code beginning with 5xx likely indicates poor formatting of the body, whereas a code starting with 4xx suggests an incorrect URL.

Replace a message

To replace a message, use the body template in Listing 4.1 and append at the end of the API the server ID, acquired from the response of the creation message flow. The final URL then follows the template `https://endpoint/v1/users/ME/conversations/<chat_id>/messages/<id_server>`. The id given by the server is intended to act as a timestamp [38], which is necessary for identifying a unique object within a group, such as a message within a conversation. Maintaining the same "clientMessageID" is essential to achieve the desired behaviour. Failure to do so could result in the system either creating a new message or returning an error. Finally, the REST request is a PUT [11], and not a POST as in the previous case, and the API requires the same $skypetoken\_asm$ token to authenticate the client.

Delete a message

Deleting a message is done by sending a DELETE request [11] to the endpoint of the replace message. In the Microsoft Teams client, by default, there is an additional parameter in the URL called $behavior = \text{“}softDelete\text{”}$. This can be omitted since the server will not check the presence of this variable

Fetching old messages

While not a primary component of the message flows, the ability to retrieve previous messages can be beneficial in the event that the WebSocket

connection is lost or if there is no connection between the client and server. The previous messages can be fetched through the same endpoint as before, but through a GET request with two parameters: the number of elements called "pageSize" and from which point in time called "startTime".
The full GET request should look like `https://endpoint/v1/users/ME/conversations/<chat_id>/messages?pageSize=200&startTime=1682512340`

4.2.6 Preflight requests

Preflight requests are a part of the Cross-Origin Resource Sharing (CORS) [39] mechanism used by web browsers to guarantee that requests sent from one domain to another are secure and accepted by the server. These requests are sent before each message and call request. The system does not check if the preflight request is made before the proper request, and thus it can be excluded and not forwarded to the server [40].

4.2.7 Primary and secondary APIs

As per Microsoft's documentation [41], in order to use the MS Teams infrastructure, it is necessary to use the "Microsoft Graph API". This API can be found at "https://graph.microsoft.com/" endpoint and it can be used to retrieve any information in the Azure domain, which may be tokens, resources, manage authentication and messages. Analysing carefully the REST requests passing through the client and the server, we can notice that the request made by the client, once the user wants to exchange a message, use a different endpoint such as `https://amer.ng.msg.teams.microsoft.com/`. Notice that the 5th ("amer" is the 5th level subdomain in the previous URL) level domain might change according to the geographical area. Any of them would work regardless of the user's geographical location, but with different latencies according to the distance.
The two APIs have a significant difference: the graph API needs several tokens to carry out a REST request, whereas the client application's API only requires the $skypetoken\_asm$ token. The additional assumption consists of use the second API because it might be less secure since Microsoft advertises the use of the first one, which is the only documented API. However, reverse engineering the API is necessary to comprehend and contextualise each endpoint tree's conceivable parameters.

4.3 Weaknesses in MS Teams

These are the weaknesses found and used throughout the study to create the covert channels. Analysing each of them individually can result in low-rating findings [42], but as a whole they can be used to exploit a system (as is explained in Section 4.4.6). Additionally, the findings are mapped to the CWE framework (CWE) [42], which stands for Common Weakness Enumeration. It is a community-driven catalogue of software and hardware weaknesses that allows for categorising malicious software and hardware behaviours into a standardised catalogue, helping to rate and compare weaknesses.

1. By default, external communication with users from other tenants (Microsoft Domains) is enabled in Microsoft Teams, allowing messages and calls to be exchanged between domains. This default configuration can be changed by the admin through the dedicated portal.

   This weakness is categorised under the CWE-453: Insecure Default Variable Initialisation. This weakness can be addressed by limiting the MS Teams communication with ONLY users in your tenant by default.

2. In Microsoft Teams, all client activities, such as call status and messages, are logged in a file. This file does not have specific permissions and can be read by any user. The file is located at `HOME/AppData/Roaming/Microsoft/Teams/IndexedDB/https_teams.microsoft.com_0.indexeddb.leveldb`, where `HOME` is the user directory. The file at this path is periodically replaced by MS Teams: it is moved within the same folder with an "OLD" keyword prepended to its filename, and a new file is created from scratch. The names of such log files are incremented by a decimal value.

   This weakness is categorised under the CWE-552: Files or Directories Accessible to External Parties. The weakness can be addressed by correctly applying an ACL (Access Control List) to the file. For instance, only admins or the application can access the file and not any users in the system.

3. The system does not validate part of the input during the POST request for sending chat messages to the endpoint: `https://amer.ng.msg.teams.microsoft.com/v1/users/ME/conversations/<chat_id>/messages`.

The request must define the body as the JSON shown in Listing 4.2, where the relevant parts are the “content” and the “message type” keys.

Listing 4.2: JSON for message structure

```
{
   "content": "<p␣
      ↪ value='secretMessage'>message</p>",
   "contenttype": "text",
   "messagetype": "RichText/Html",
   "clientmessageid":clientmessageID,
   "imdisplayname":"",
   "properties":{
     "importance":"",
     "subject":""
   }
}
```

Sending HTML instead of plain text is possible in MS Teams, and the attribute value can be set within the <p> tag without any response from the client GUI. However, the entire message, along with the clientmessageID, will be logged in the WebSocket and the log file. It is important to note that the clientmessageID must be unique for all messages, and it is necessary in order to modify or replace a message. Most of the tags are validated and there is a whitelisting for the input tag and especially for some key attributes to inject JavaScript. The script tag, the onError tag and other keywords cannot be injected. There is the possibility to send images as well with the same JSON structure, but the image URL must be within the Microsoft domain.

This weakness is categorised under the CWE-790: Improper Filtering of Special Elements. The weakness can be addressed by parsing correctly the sent JSON and whitelisting the essentials tags and attributes.

4. A chat message, or part of it, can be changed in the delivery side without further notification on the receiving side. In any case, the new message will be added in the log file and in the WebSocket .

5. Anyone can send a message to the endpoint of a webhook without any sort of authentication . The webhook is structured as `https://xxxxx.webhook.office.com/xxxxxxxxx`, and it can be used via POST request by anyone without the need of cookies/tokens,

as opposed to other scenarios. The xs are letters generated by Microsoft during the creation of the webhook.

This weakness is categorised under the CWE-306: Missing Authentication for Critical Function. This weakness can be addressed by enforcing an authentication on the webhook requests and applying a limit for requests per user.

6. By default, client settings and updates are regularly pulled from the server with an interval of roughly one second. The client takes a perceivable time to update itself and to display the information on the screen if an action is performed by another device.

7. Everyone can subscribe to the WebSockets using the tokens, as there is no authentication of the client application . Unofficial applications can use the WebSocket service without a preregistration. There are two WebSockets: one for the messages and another one for the incoming calls. Both of them require a Ping/Pong request to keep the channel alive.

   This weakness is categorised under the CWE-287: Improper Authentication. This weakness can be addressed by authenticating the sender, thus only registered and verified applications can request content.

8. The authentication cookies are securely stored in an encrypted format on the machine's file system. However, any user can access these cookies without requiring permission. To decrypt the cookies, it is necessary to retrieve the symmetric key used for encryption from `C:/Users/username/AppData/Roaming/Microsoft/Teams/LocalState`. This symmetric key is encrypted using the TPM, which stands for Trusted Platform Module, and employs the DPAPI encryption scheme. Before decrypting the cookies, the TPM-encrypted symmetric key must be decrypted. The TPM is a specialised hardware component that enhances the security of a computer system by providing a secure storage area for cryptographic keys and a secure environment for executing cryptographic operations. It helps protect against unauthorised access to sensitive data and prevents tampering with the system's configuration.

   The DPAPI, which stands for Data Protection API, is a component of the Windows operating system that offers data protection services.

It is used to encrypt and decrypt sensitive data, such as passwords or cryptographic keys, using either the user's login credentials or the computer's credentials as the basis for the encryption key. This ensures that sensitive data remains protected while at rest on the system's disk.

Since the master key is exclusively stored within the TPM, decrypting the symmetric key involves submitting it to the TPM using a system call. The TPM, in turn, utilizes the DPAPI to decrypt the symmetric key. Once decrypted, this key is used in combination with AES in GCM mode to decrypt the cookies file stored at `C:/Users/username/AppData/Roaming/Microsoft/Teams/cookies`. The cookies file contains all the cookies used by MS Teams and is encrypted using the same method. Within these cookies, there are the $skypeotoken\_asm$ and the $bearer$ tokens The cookie file is a database file, which needs to be opened through an appropriate library, such as SQLite, or similar tools. The cookies' values can be then fetched using SQL.
This entire cookie decryption process is summarised in Figure 4.5.

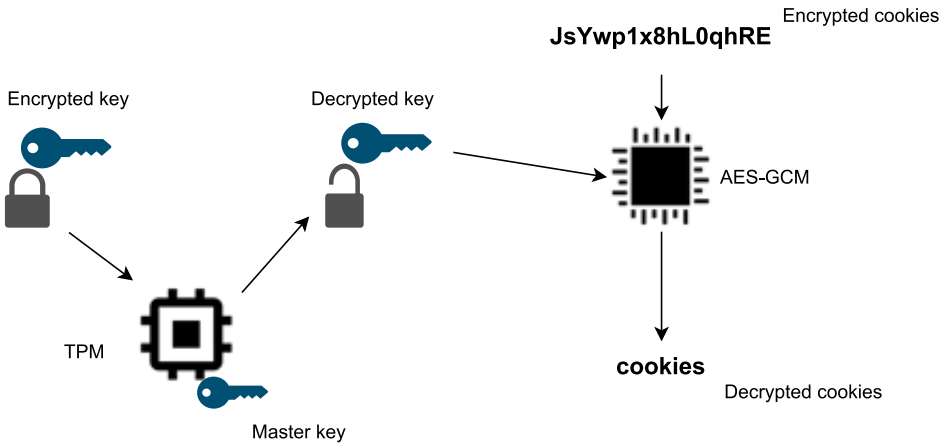


Figure 4.5: Cookie decryption schema to retrieve the skypetoken_asm and the bearer tokens

This weakness is categorised under the CWE-693: Protection Mechanism Failure. The weakness can be addressed by having a shared secret between the client and the server or bounding the cookie with the browser fingerprinting.

9. Incoming packets from the client are not validated by the TURN server, and this allows crafting and sending of additional packets to the server. If there is a UDP channel established between the client and the server,

as shown in Figure 4.6, it is also possible to inject packets into the channel, and the TURN server will relay those packets to their intended destination. The packets require to be extracted from the flow by the receiving side to separate the proper call traffic from the injected ones. The additional packets do not influence the Teams client.

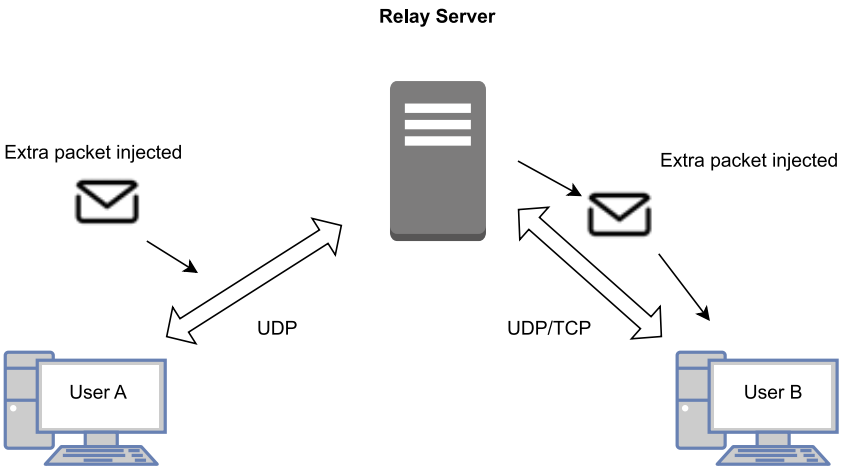


Figure 4.6: Packet injection into a MS Teams call

This weakness is categorised under the CWE-345: Insufficient Verification of Data Authenticity. The weakness can be addressed by validate the incoming data with content filtering or applying a sequence number inside the stream.

10. There is no CSRF token in the credential requests for the WebRTC token/relay server token. It can be requested many times with a successful response.

    This weakness is categorised under the CWE-352: Cross-Site Request Forgery. The weakness can be addressed by adding a CSRF token for each request.

4.4 Covert channels and malware design

The weaknesses in Section 4.3 lay the foundations for four covert channels, namely the message, the call, and the webhook, which will be explored in the Section. As mentioned in Section 1.1, these covert channels enable communication between the attacker and the victim without establishing direct

connections. Figure 4.7 then illustrates the schema of the malware design, leveraging the aforementioned covert channels to extract information from the victim's computer, and exploiting a message covert channel to issue instructions to the victim's computer. In the following subsections, this thesis will detail the process of establishing the covert channels, referring to the extraction channels as outgoing ones and the instruction channel as incoming ones.

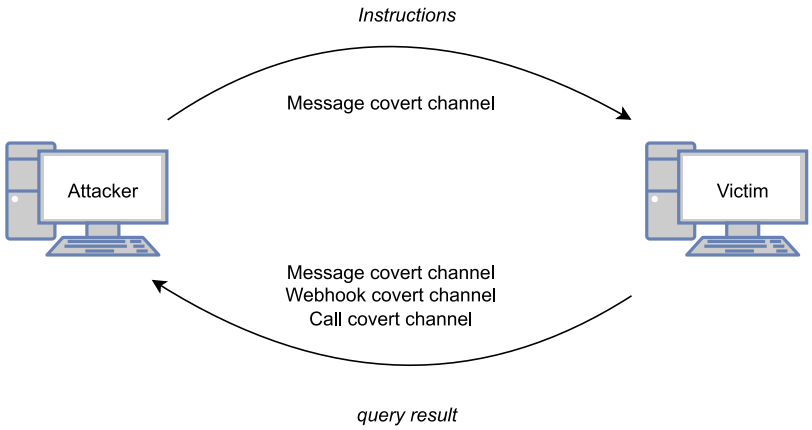


Figure 4.7: Schema of malware design

4.4.1 Incoming channel

The incoming channel needs to instruct the victim machine to execute a specific command. This channel needs to be robust since it will be the control channel, and losing it would break the attack chain and leave the attacker without any attacker-victim connection.

After the victim has initialised the connection with the attacker (see Section 4.4.6 for more details), the attacker has an opened MS Teams chat with the victim. Once the receiving side accepts the first incoming message from the sender, the chat becomes open. It means that no further notification will appear for the new sender. In order to send out the commands, the attacker exploits weaknesses 1 and 3: the input validation weakness allows the forwarding of the attacker's chat message to the victim machine, given that all communication with external users is enabled by default.
Weakness 3 may also be useful in case our attack is discovered by the end-user or by a system admin: the sent message is displayed in the application as a simple "Hello" chat message, but it carries the extra layer of information

in the “value” attribute of the <p> tag in the HTML. This value is not shown by any Microsoft application, but it can be read in the WebSocket or in the logfile, as described in weaknesses 7 and 2. The notification of the incoming message is not even displayed to the end user, as described in more detail in Section 4.4.6.

Once the first attacker’s chat message is sent, we want to keep instructing the victim’s computer to perform actions for us. Using weakness 4, we can then modify that first message to contain additional instructions. This step requires parsing the response ID given by the API in the previous step and recalling the clientMessageID from the message before.

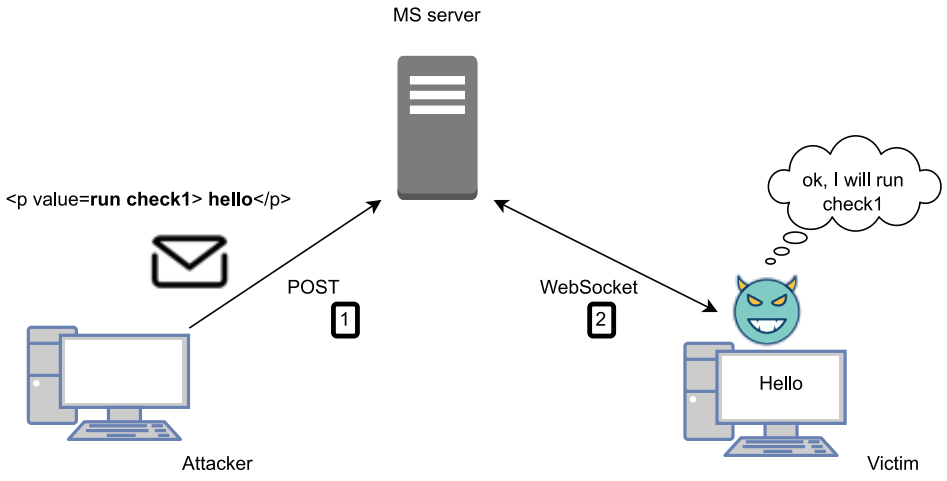


Figure 4.8: Schema of incoming channel. Instructions are sent using a chat message affected by input validation

This incoming channel works for the following reason: The message is not sent directly to the other user, but instead it is sent to the server and then, the WebSocket will fetch the message from it. The TLS proxy does not control for WebSocket traffic, and the firewall cannot block an already established connection. In order to prevent such a communication, the TLS proxy has to parse the WebSocket communication and distinguish or detect keywords in the messages.

It is important to note that in the WebSocket flows different system commands or properties, since the server may need to communicate some actions to the client. Trying to detect the malicious message and leave the valid ones can be challenging with the risk of limiting the functionality of

the application itself. Additionally, we can assume that the data written in the WebSocket is legit and already sanitised since it comes from a known source. There is still the chance that a Firewall can block the establishment of the duplex channel of the WebSocket, but this would limit and block the functionality of many applications.

4.4.2 Outgoing channel - webhook

This first outgoing channel explored uses the webhook as a carrier of information. One of the options to move the data from the victim to the attacker is through the webhook.

This channel is created on the attacker's side; as a user, you can create as many webhooks as you wish and then attach them to any MS Teams channel. Any user can then forward a message using the webhook URL, even though it is not his own webhook or it belongs to the same domain. Only the knowledge of the webhook URL is sufficient to forward messages, as mentioned in weakness 5. Sending the message to the webhook is only the first part of the covert channel; the second part consists in a properly crafted message that allows the attacker to render a Microsoft Teams card [43]. Using this Teams card, a user can specify a title, a paragraph, an author and an image. The image does not need to be an internal Microsoft resource, as it can be fetched by the Microsoft server from anywhere on the Internet.

This covert channel is then created by sending to the webhook a Teams card with an image URL that points to the attacker's webserver. The image is actually the response to the attacker's query in an encoded or/and encrypted form. The result of the attacker's query could be then seen by the requested endpoint, as shown in Figure 4.9.

4.4.3 Outgoing channel - message

The second type of malware design uses messages as a means of transferring data between the victim and the attacker. This covert channel mainly takes advantage of weakness 8 to impersonate the user, and weakness 6 to ensure a user does not perceive the ongoing attack. Once the attacker has the session cookies, he is able to impersonate, access the user's service, or read sensitive data. With cookies obtained through weakness 8, the malware can then send MS Teams messages acting as the victim.

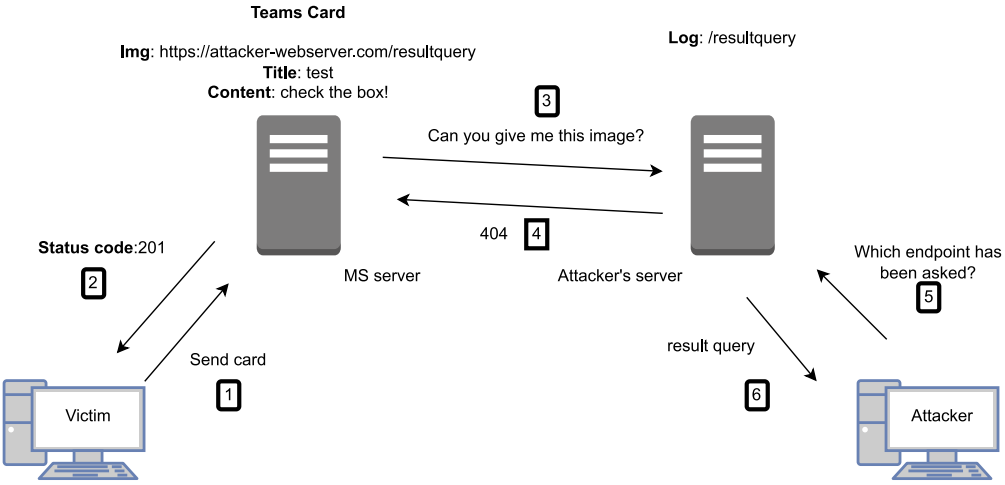


Figure 4.9: Webhook outgoing channel with MS Teams card rendering process

In MS Teams there is also the possibility to mute and hide a chat. This results in avoiding pop-up notifications if a message is received in a specific chat, and not displaying it in the list of Teams chats. The only way to undo this setting is to actively search for the desired sender, right-click on it and select the unmute option. This is unlikely to happen since the victim user needs to know the username or user id of the attacker, and in this scenario, the victim is not aware of the ongoing attack.

4.4.4 Outgoing channel - call

The third malware design method uses calls as a carrier of information. In this particular scenario, we assume that the victim's company blocked all peer-to-peer connections. Peer-to-peer communication played historically a critical role in modern network [44], being a blind spot for system administrators. The major issue comes from the impossibility to correlate traffic through a "stable" entity, such as a server, thus leaving room for data extraction from bad actors. Due to the fact that peer-to-peer communication is not available for MS Teams calls, Microsoft's fallback moves all the calls in a client-server architecture, using a TURN server (see Section 2.3). Additionally, a company can either choose to block all the UDP connections, and use TCP ones instead, or take advantage of the UDP properties for a faster connection [45].

Assuming that a company would use the UDP protocol to establish a

connection with the TURN server and it is possible to read the WebSocket (see weakness 7), the attack starts with the malware listening on the WebSocket for incoming call notifications and later creates a ghost call to transfer the data. The Listing 4.3 shows an example of the call notification, the most relevant fields are displayed, while the less meaningful were deleted for readability.

Listing 4.3: JSON received by the Teams server

```
{
   "id":106757833,
   "method":"POST",
   "url":"/v4/f/RKyQnklY9ku8DB_NyzSxOg/",
   "headers":
   {
      "Trouter-Timeout":"19298",
      ...
      "Host":"xxxxx-.trouter.teams.microsoft.com:22004",
      ...
      "trouter-request":
      {
         "id":"182c457b-xxxxxxxx",
         "src":"trouter2-azsc-euno-0-b",
         "port":31012

      }
   },
   "body":{
   "evt":107,
      ↪ "cp":"H4sIAAAAAAAACs1VTY/bNhD9Lzr00t00t00t...",
   ...
   }
}
```

It is important to identify this specific call among all the WebSocket messages, with particular emphasis on the "cp" field in the body section. Although it may look like random bytes, it contains the endpoints to accept and start the call process. To retrieve the plain text of the "cp" field, it is necessary to decode from base64 to bytes and then decompress the result. The plain text would result in something like the following strings (the xs represent random characters generated by Microsoft on run-time):

- attach: api.flightproxy.teams.microsoft.com/api/v2/ep/x.cc.skype.com /cc/v1/forked/5fa8xx-xx-803b-xxxx/27/i1/941/**attach**?i=10-60-1-21

- progress: api.flightproxy.teams.microsoft.com/api/v2/ep/x.cc.skype.com /cc/v1/forked/5fa8xx-xx-803b-xxxx/27/i1/941/**progress**?i=10-60-1-21
- reject: api.flightproxy.teams.microsoft.com/api/v2/ep/x.cc.skype.com /cc/v1/forked/5fa8xx-xx-803b-xxxx/27/i1/941/**reject**?i=10-60-1-21
- accept: api.flightproxy.teams.microsoft.com/api/v2/ep/x.cc.skype.com /cc/v1/forked/5fa8xx-xx-803b-xxxx/27/i1/941/**accept**?i=10-60-1-21
- answer: api.flightproxy.teams.microsoft.com/api/v2/ep/x.cc.skype.com /cc/v1/forked/5fa8xx-xx-803b-xxxx/27/i1/941/**answer**?i=10-60-1-21

At this point, the real application client would need an estimated one second to create the pop-up for the incoming call for the user, but this can be avoided if the malware accepts the call right away. Based on the decompressed "cp", the following steps should be carried out to accept a call:

1. a POST request to attach;
2. a POST request to progress (optional);
3. a POST request to accept (to avoid the display of the notification to other clients);
4. a POST request to answer.

The response to the attach POST request (step 1) can be interesting since it may contain the API endpoints to send chat messages, to mute/unmute, and the possibility to redirect the channel. After the response to answer POST request (step 4) has been received, the Microsoft infrastructure would wait for an incoming connection in order to establish a bridge between the two users. This is done in two steps: first the negotiation with the webserver of the ICE and then the request for relay server credentials.

The ICE protocol is needed to negotiate with the server which protocols should be used in the channel, as the client is not aware of the policy enforced by the firewall, and thus it needs to verify if the UDP or TCP connection can be established. The ICE protocol also negotiates where to connect (to which relay server, which would then carry the webRTC datagram) and lets the client be aware of the NAT setting of its own network. To establish a connection with the relay server, a GET request to `https://teams.microsoft.com/trap/tokens` needs to be performed with the skypetoken_asm cookie. The server would respond with a pair of credentials that can be used to authenticate

the client to the relay server. After the authentication with the client succeeds, the client should be able to forward traffic to the server using UDP.

4.4.5 Malware horizontal spreading

Horizontal spreading, or so-called lateral movement, allows spreading the malware across the network in an easier way, without passing through the firewall or other network security devices [46].

The proposed malware does not have any features that allow for a direct spreading of itself, but it can be nevertheless used in phishing attacks across the company. Phishing is a type of attack where an attacker pretends to be a trustworthy entity in order to trick the victim.
Usually phishing attacks come from emails/messages where the sender looks like a real one, but it is not. An example could be a misspelled word in the email that seems like the original one or an email with similarities to the real one.

Using this design, the attacker can impersonate the victims into another (not infected) user to trick other users into opening a file or performing some action. Through the outbound covert channel, we could extract the tokens from the systems and send them over the channel. The implant installation can be done through various malware propagation methods such as through macro, pdf or links.

4.4.6 Malware flow

The malware design for the threat scenario described above in Section 3.4, using a cloud infrastructure, has the following requirements:

- The TLS proxy needs to be avoided, otherwise it would analyse all requests passing through and trigger the alarms. If it is not possible, the application layer encryption needs to be performed.
- The HTTP REST requests by the malware should be done using reliable IPs such as Microsoft, Google, or Amazon. This would prevent triggering conditions in the firewall since it expects many requests with these IPs for internal services such as Microsoft Word or Google Single Sign-On.

- The user of the infected computer should not have interactions with the system in place, or the flow would be compromised otherwise.
- The incoming channel should not be blocked by the firewall.

We will now go through the whole command-and-control malware flow from the beginning.
Once the malware is installed in a company's computer, the first step is to inform the attacker about the status of the implant. The implant has the knowledge of a webhook URL, a session key for symmetric encryption, and the Teams ID of the attacker. Before starting the initial contact, the implant has to fetch the cookies from the machine, as described in the Weakness 8. The cookies, private and public IPs, Teams User ID, and tenant ID are then encrypted and sent over through the webhook using an MS Teams card format. The Teams user ID and the tenant ID can be requested at `https://teams.microsoft.com/api/mt/emea/beta/users/aggregatedNotification` using the cookies. The attacker needs to automatically fetch the information from the public webserver, where the request was made, and decrypt the result. Section 5.3 explains in detail the advantages and drawbacks of this solution, and a further investigation is described in Chapter 5.1. One of the main drawbacks of this outgoing channel is the unreliability of multiple requests, and therefore cannot be used for the command and control (see Section 2.2.3) communication.

Once the first interaction with the attacker is performed, a chat between the victim and the user is to be created. This second outgoing channel is meant to be a reliable and secure way to exchange information between the attacker and the client, so the webhook will not be further used. The cookies sent in the first webhook allow restoring the connection, have a fallback plan, and spread horizontally the malware as described in Section 4.4.5. The peer-to-peer chat can be initialised by combing the IDs of both users together as follows: the request has to be delivered to the endpoint `https://emea.ng.msg.teams.microsoft.com/v1/threads` with the JSON in the Listing 4.4. The listing contains the MS Teams chat configuration in JSON format.

Listing 4.4: JSON body to create Teams chats

```
{
   "members": [
      {
         "id": "8:orgid:cac502b0-xxx-xxx....",
```

```
        "role": "Admin"
      },
      {
        "id": "8:orgid:e49a1c77-xxx-xxx...",
        "role": "Admin"
      }
    ],
    "properties": {
      "threadType": "chat",
      "chatFilesIndexId": "2",
      "uniquerosterthread": "true",
      "fixedRoster": "true"
    }
}
```

The IDs need to be changed according to the sender and receiver. The other fields should remain as above, although we have not fully tested this functionally, unpredictable behaviour might occur by changing the other field, such as the role admin. The resulting chat would have the first ID concatenated with the second one, and both would be appended as follows: `https://teams.microsoft.com/conversations/19:<id1>-<id2>@unq.gbl.spaces` After the creation of the chat, the malware needs to mute, unpin, and hide the chat, hiding any incoming message from the user. This can be accomplished with three GET-request to the following URLs: `https://emea.ng.msg.teams.microsoft.com/v1/users/ME/conversations/<id>@unq.gbl.spaces/properties?name=alerts`, to avoid notification for incoming message, `https://emea.ng.msg.teams.microsoft.com/v1/users/ME/conversations/<id>@unq.gbl.spaces/properties?name=unpinnedTime` and `https://emea.ng.msg.teams.microsoft.com/v1/users/ME/conversations/<id>@unq.gbl.spaces/properties?name=historyHiddenTime`, to set in background the notification. The unpin and the hide GET-requests have the JSON body as $unpinnedTime : 168442xxxx$ and $historyHiddenTime : 168442xxxx$, respectively. The key values of both bodies are epoch timestamps of 10 digits long. The timestamp needs to be always updated for request, or setting it to a future epoch would work as well. All these three requests are PUT REST requests, contrary to the other REST requests.

At this point, the attacker can instruct the victim's computer by sending a message through the incoming channel, as described in Section 4.4.1.

The receiving side can use the WebSocket or the log file as highlighted in weaknesses 7 and 2. Both options work, but using a log file to transfer information is stealthier since there will be no additional connection from the client to the MS Teams web server.

The malware can read the instruction and send over the result through the message channel. In this case, the hide, mute, and unpin sequence needs to be performed again. These multiple requests would not be perceived by the user due to the amount of time that the client takes to fetch the new information about the chat status from the server.

The attacker can then instruct the malware to open up a call between them if a bigger chunk of information needs to be exchanged, as defined in Section 4.4.4. For instance, if a database file needs to be transferred, then a call covert channel would perfectly fit this use case. Even in this case though, we should consider the duration and traffic limitation described in Section 5.3: a call can be detected if it lasts for an unrealistic amount of time (a call hardly lasts more than 1-2 hours) or if too many bytes are consistently uploaded.

The flow described in this section is summarised in Figure 4.10 with the three actors attacker, victim and Microsoft server. Although there is an arrow between the victim and the attacker, the connection always goes through the Microsoft service as described in Sections 4.4.1, 4.4.3, 4.4.2 and 4.4.4.

4.5 Summary and key points

Initially, the analysis provided an overview of certain traffic patterns that arise from the usage of Microsoft Teams. Given the extensive capabilities and numerous features of MS Teams, it becomes possible to transmit data through hidden channels using multiple different patterns. The main focus of the chapter revolves around investigating the behaviour of the undocumented API and its distinct characteristics. Within the Teams application, a total of 10 weaknesses have been uncovered, each accompanied by their corresponding CWE (Common Weakness Enumeration) references. Lastly, building upon the identified weaknesses, we defined four covert channels within the application and propose a design for a malware that exploits them.

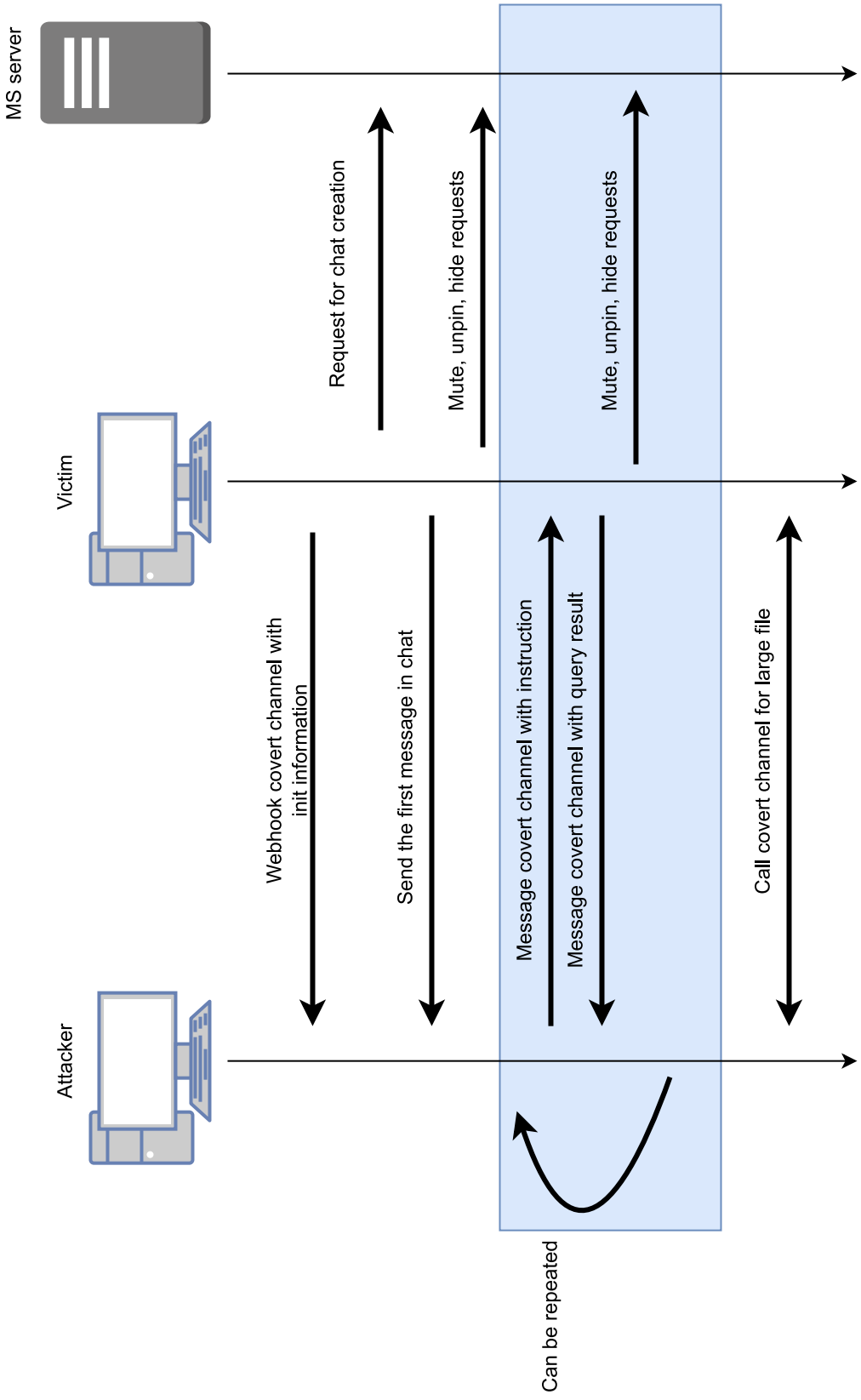


Figure 4.10: Malware design flow

Chapter 5

Results and Analysis

In this chapter, we present the evaluation of the discovered covert channels. For each channel the bandwidth, the efficiency, and the robustness are tracked. The focus of the evaluation concerns the reasoning behind the chosen outgoing channels in the malware design, discussed in Section 4.4.6. Therefore, this evaluation does not prove which channel is better, but instead, highlights its characteristics and its properties.

The incoming channel, the channel from the attacker to the victim, is not explicitly evaluated since it follows the same properties as the message outgoing covert channel, the channel from the victim to the attacker. Lastly, we discuss the reliability of the analysis and its reproducibility as well as the detection methods to identify the discovered covert channels.
The outlined behaviours are not reported and documented by Microsoft and the information about the limits of the request per second/minute did not reflect the analysis.

5.1 Evaluation

The following plots depict the characteristics of the identified covert channels against the evaluation framework described in Section 3.6.
To conduct the measurements, two virtual machines (one emulating the attacker and another one emulating the victim) and also an external server in the webhook case were employed, as described in Section 4.1. Python served as a tool to estimate the bandwidth, proof robustness and measure the time for the efficiency. To maintain consistency in efficiency measurements, the system clocks of the machines were synchronised. It is worth acknowledging

that the choice of Python as a programming language may result in increased latency due to its interpreted nature, which generally makes it slower compared to other programming languages [47]. Nevertheless, all measurements were taken under identical settings and using the same internet connection.

5.1.1 Bandwidth

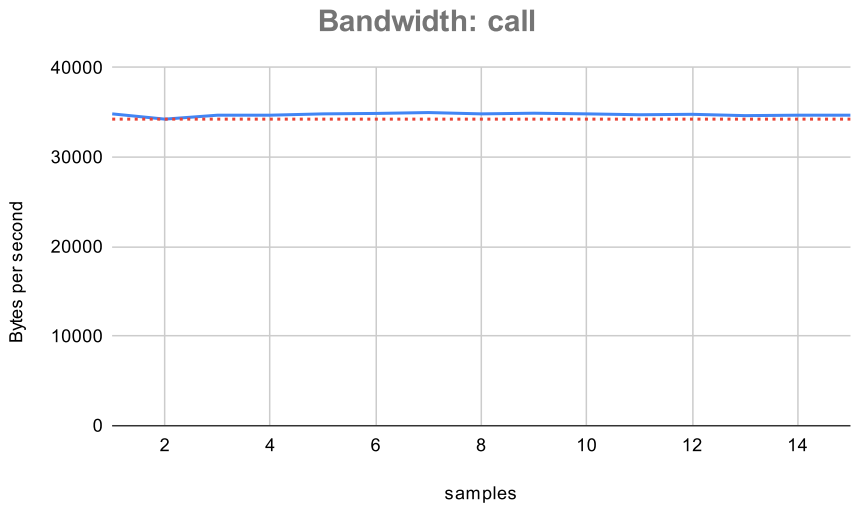
The bandwidth, as defined in Section 3.6, is "The amount of data that can be sent per second using a covert channel. To eliminate the elapsed time of process switching in the same system, the bandwidth needs to be evaluated with the least number of processes running on the machine; the measurements are repeated several times, and the minimum value is then selected [36]. A higher bandwidth means a higher transfer rate and a more effective covert channel. The values are calculated by measuring the number of bytes sent to the server within a one-second interval. In the bandwidth plots (Figures 5.1a, 5.1b and 5.1c), the blue line represents the trend observed during the evaluation, while the red dotted line is determined by the minimum observed bandwidth value and defines the available bandwidth of the channel. The vertical axis corresponds to the measurement of bytes per second, whereas the horizontal axis represents the sample number. Each sample is an experiment of either 10, 20 or 30 seconds. During each experiment, the malware attempted to send as many requests as it could within the given time frame. The total number of requests transmitted was then divided by 10, 20, and 30 seconds, respectively. Finally, the resulting values were multiplied by the maximum amount of data that a covert channel data unit (i.e., chat message, image path and UDP payload) can transfer.

Figure 5.1a shows the bandwidth of the call covert channel. It presents a nearly linear graph, with the minimum one closely reflecting the overall trend, indicating a bandwidth of 34,300 Bytes per second. In this specific scenario, the channel's throughput aligns with its bandwidth due to its lossless behaviour, as highlighted in the robustness Section 5.1.2. Despite the relatively small size of the UDP packets used to encapsulate the covert channel data, the observed bandwidth of this channel is almost five times larger than the webhook case. This high throughput is achieved because UDP, being a stateless underlay protocol, does not necessitate ACK (Acknowledgement) from the receiving side, ensuring a fast and responsive channel.

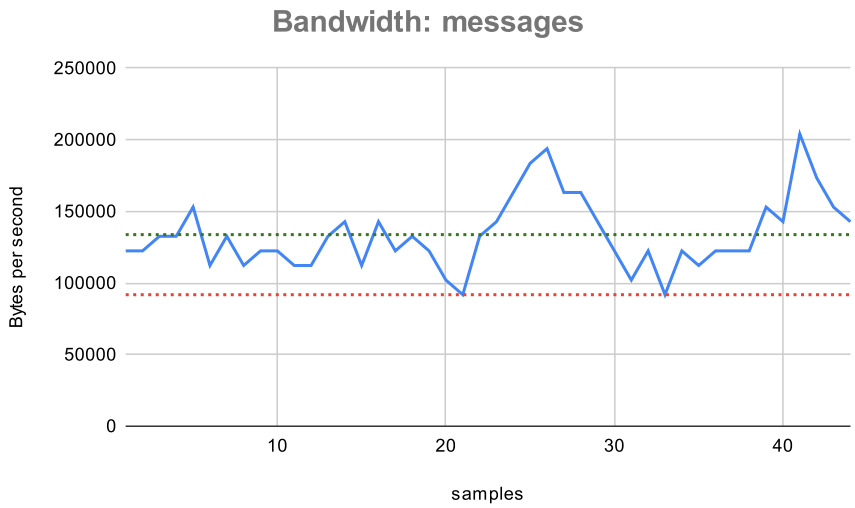
Figure 5.1b shows the bandwidth of the message covert channel, this plot

refers to the outgoing message channel as well as the incoming message channel. Unlike the previous case, the values depicted in the plot demonstrate a fluctuating pattern in the channel. While two experiments achieved a rate of 200 KB per second, the majority of the experiments exhibited bandwidths ranging from 100 to 150 KB per second. Additionally, the average value of the series, denoted by the green dotted line, stands at 133,579 bytes per second, contrasting with the minimum value and bandwidth of 91,800 bytes per second.

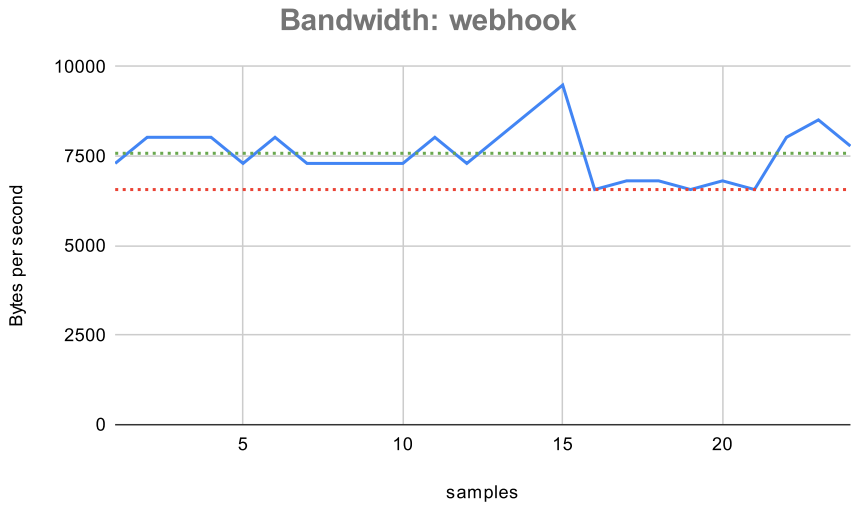
The last analysed bandwidth corresponds to the webhook, as depicted in Figure 5.1c. The plot illustrates a less spiky graph compared to the previous one, characterised by the relatively small difference between the red dotted line (representing the minimum) and the green dotted line (representing the average). The channel's minimum value and bandwidth amount to 6,567 bytes per second, while the average hovers around 7,500 bytes per second. The bandwidth of this covert channel falls in the last position, being 15 times less effective than the message channel and 5 times less effective than the call channel.



(a) Bandwidth measured during call: 34,300 bytes per second



(b) Bandwidth measured using the chat message: 91,800 bytes per second



(c) Bandwidth measured using the webhook: 6,567 bytes per second

Figure 5.1: Evaluation channel bandwidth over multiple samples. The blue line marks the samples, and the red dotted line outlines the minimum value.

5.1.2 Robustness

It is crucial for the whole command-and-control system to be robust, in order to reduce the likelihood of packet loss in the communication process. In some cases, a highly robust channel may outweigh the bandwidth. For instance, a channel with a 30% delivery rate and 10 bytes per second may be less efficient compared to a channel with a 100% delivery rate with 5 bytes per second. In the case of the 30 % delivered channel, the additional overhead comes from re-transmitting the lost packets on the way or employing error correction, which can increase the overall amount of time to transfer a file through the channel.

The robustness plots in Figure 5.2 present the percentage delivered ratio on the y-axis, while the number of consecutive requests is shown on the x-axis. If not indicated, the time interval between two adjacent requests is assumed to be 0.

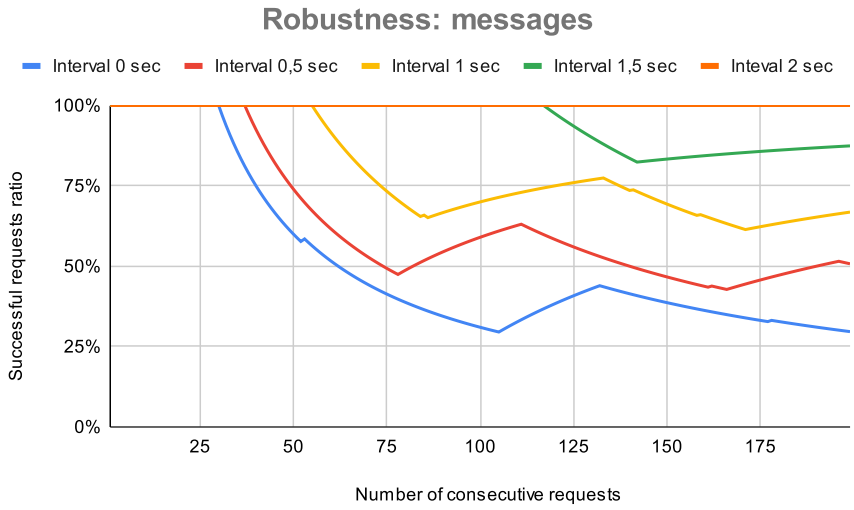
Figure 5.2a displays the robustness of the Teams messages with various interval times between requests. The experiment includes over 175 requests per scenario and indicates the accumulated percentage of delivered messages at nth made requests. For instance, the yellow line, which represents a 1-second interval, has a 75% successful request ratio after 75 requests made. This implies that 56 messages were received and 19 were lost in transit. This plot reveals the difference and importance of the interval time between consecutive requests for this channel. Running the system with no time between requests leads to a low delivery rate of around 30%. Already with an interval of 0.5 seconds, the delivery ratio raises to 50%, which can be already considered a reliable covert channel. The 2 seconds interval is the lower bound interval for achieving a 100% successful delivery rate. It is important to notice that in any system the initial requests always succeed and the decay starts after n requests according to its interval time. As shown in Figure 5.2a, the first drop happens after the $30th$ request with a 0-second interval and after the $55th$ request with a 1-second interval. This happens due to the DoS (Denied of service) protection [48]. Once a server receives many requests (or in this case many messages) from a client, the DoS protection system takes over the load and blocks some requests to allow the server to have enough time to process the already received requests.

Figure 5.2b represents the robustness of the webhook with multiple samples, where the experiment is conducted multiple times to obtain a more

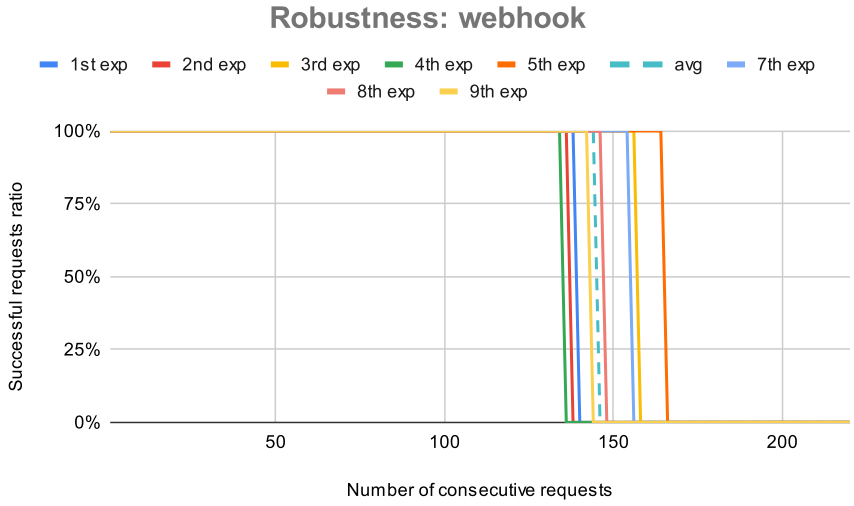
precise result. The dotted line is the average value of the collected samples, while the individual lines correspond to different experiments.

Based on the plots, the webhook exhibits a normal behaviour that allows for multiple consecutive requests, typically up to approximately 165 requests. Beyond this threshold, the webhook ceases to function. Although this information is not explicitly shown in the plot, it is worth noting that the webhook resumes operation after approximately an estimated day. Based on the experiments, it looks like a counter being reset after a certain amount of time. While this channel demonstrates high reliability and enables the transmission of multiple requests without any loss, it is not suitable for request-expensive operations.

In the process of assessing the robustness of the call covert channel, the same methodology was employed. Across the channel, all packets were effectively transmitted to the receiving end without any loss, amounting to zero packet loss out of five thousand packets. The experiment was repeated several times within the same day and on different days throughout the week, consistently yielding the same outcome. Despite the UDP protocol underpinning the channel, the success rate of packet delivery remained at 100%.



(a) First packet lost after 30 - 37 - 55 - 117 - 175+ requests in their interval



(b) Webhook dropping on average after 145 consecutive requests, maximum after 165 consecutive requests.

Figure 5.2: Evaluation robustness over different samples.

5.1.3 Efficiency

Efficiency, as defined in this study in Section 3.6, refers to the latency time between the sender and the receiving side of the covert channel. To assess the efficiency criterion, two timestamps are recorded for each measurement: one before sending out the data and one after having received it. The latency is calculated by subtracting the two timestamps, providing a measure of the time it takes for data to traverse the covert channel. To ensure accuracy, the system clock time was synchronised before conducting the evaluation, minimising potential errors.

The call channel stands out as the fastest solution proposed, as shown in Figure 5.3a. The blue line marks the consistent trend of the data sent through the channel. When data is sent, it is directed to the STUN server, which then forwards it to the intended receiver. The transmission of data occurs within the channel using the UDP protocol. These characteristics contribute to the channel's snappy nature, resulting in lower latency compared to other covert channels in Figures 5.3b and 5.3c.

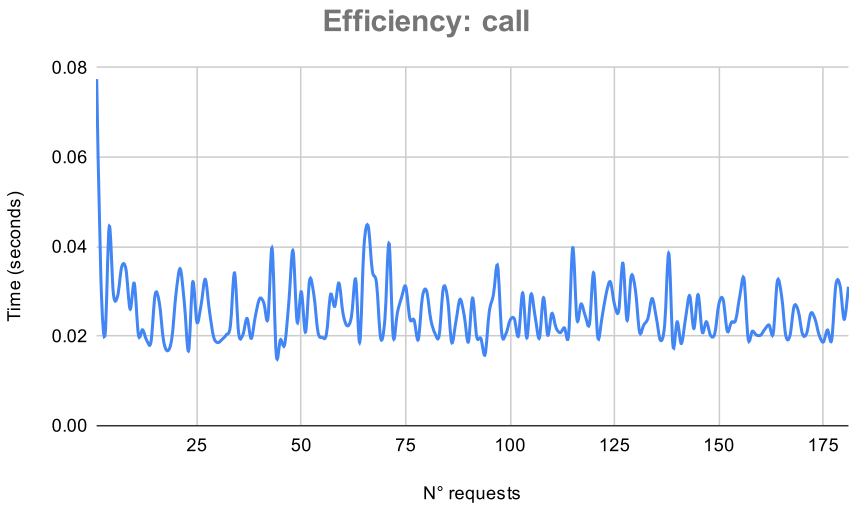
The plot illustrates the latency values within a range of 0.02 to 0.04 seconds. It is noteworthy that across the 175 measurements, no outliers were observed, indicating a consistently stable channel in terms of efficiency.

Figure 5.3b illustrates the efficiency of the message channel. The blue line represents the latency time, ranging from 1.4 seconds to 2.2 seconds. However, there are a few outliers in the evaluation, with some reaching 4.5 and 6 seconds. Despite these outliers, the overall trend of the line remains consistent over time, with an estimated efficiency of 1.7 seconds to reach the receiver. In this data transmission method, the data is sent using a POST request to the server and then pushed to the client through an open WebSocket connection.

The efficiency of the webhook covert channel is examined in Figure 5.3c. In this channel, the data is transmitted through the webhook, triggering the Microsoft server to render the Microsoft Teams card with its associated image. The rendering process initiates the delivery of the data to our malicious server. This process introduces a notable latency.

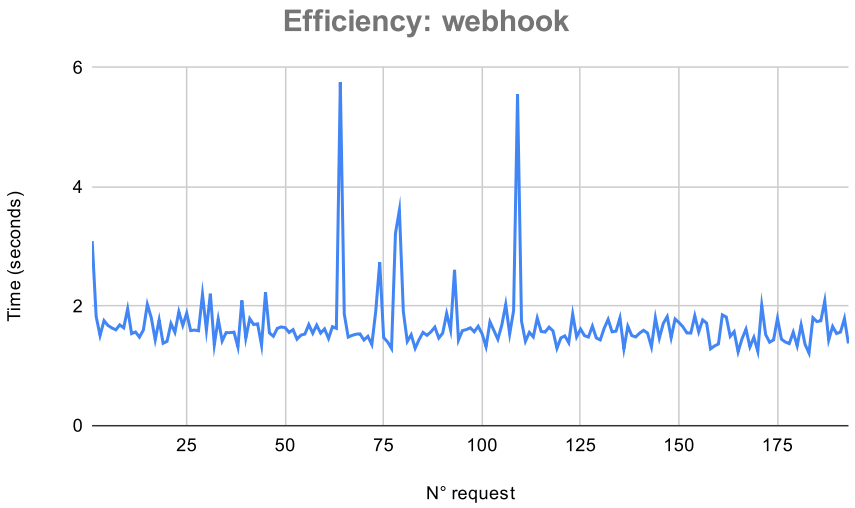
Based on the analysis of the experiment, the webhook covert channel takes an average latency of approximately 1.8 seconds for delivering the

message to the receiving side. This latency is relatively high compared to the other proposed channels, making it one of the slower covert communication methods. The latency remains consistent over time, as shown by the trend observed in the chart. However, a few data points occasionally surpass the threshold of two seconds.



(a) Efficiency call channel on average 0.025 seconds.

(b) Efficiency chat message channel on average 1.75 seconds.



(c) Efficiency webhook channel on average 1.6 seconds.

Figure 5.3: Evaluation efficiency channels. Seconds taken from sender to receiver.

5.2 Threats to validity

The measurements were collected in the Zurich (Switzerland) region during March and April 2023. If the same data was to be collected in a different geographical area or at a different time, the result would possibly vary. If the channels were measured in one, two or more years, the characteristics of the communication channels might change significantly, or they could even be blocked by Microsoft.

This results can be hardly applied into other setup such as Google Meets or Zoom, although the same functionally can be presented in this collaborative application. The message covert channel can result in completely different result by applying a more strict filter for how many messages per seconds can be sent. The webhook covert channel might be peculiar to the MS Teams application due to the Teams Card and the possibility to request images outside the Microsoft domain. Lastly, the call covert channel can be applied in different scenario, and the expecting properties might reflect the once identified in this work.

In April, a major update was released for Microsoft Teams, which reportedly affected the client. However, it may have also impacted the back-end servers. To obtain a more accurate measurement, alternative tools should be used instead of Python. Particularly, when considering efficiency, the values are relatively small, and a more precise analysis could be achieved by directly retrieving the data from the network interface card, instead of fetching the data from the OS through Python.

5.3 Detection

To bypass state-of-the-art detection systems, it is essential to avoid synchronous communication among the bots in the network [30, 29]. Network correlation and "behaviour similarity" among peers in the network represent a non-trivial problem for botnets not targeted.
For covert channel detection, machine learning is not an out-of-the-box solution. Currently, the training datasets for this problem do not exist, making it impossible to apply supervised machine learning techniques such as SVM and decision trees [25]. On the other hand, unsupervised algorithms might be able to detect anomalies among REST requests or traffic distributions.

Small and medium (or even large) companies cannot afford to train machine learning algorithms based on their traffic distribution, especially when it comes to keeping track of every application-based baseline and updating it periodically [26]. Even if they could, the required data for such a solution can be an overhead for many realities and it may create a high amount of false positives with low accuracy [26].

One of the most advanced techniques to detect covert channel attacks is traffic normalisation [20], enforcing a fixed packet size across all the applications. This should be theoretically applied to the Teams calls as well. In fact, having such a strict policy would increase the jitters of the call, making unusable such applications.

5.3.1 Webhook channel

A Microsoft webhook URL follows the template `https://xxxxx.webhook.office.com/yyyyyyy`: the xs stands for the tenant name, where the webhook has been created, and ys stands for the randomly generated id. This domain is mapped to an IP in the Microsoft domain, which can be difficult to detect. Additionally, the secret data is stored in the URL of the image in a base64/encrypted format, which cannot be decrypted by the TLS proxy, since the data is encrypted in the application layer and not in the transport layer. The peculiar behaviour of this channel consists of multiple requests to the same endpoint, which might attract some attention from the security system [49]. The IPS in combination with the TLS proxy might in fact detect it by looking at the request's entropy [23]. On the other hand, a firewall would not be helpful in this case due to the reliability of the IP (The IP belongs to a well-known service of Microsoft [50] and Microsoft advise whitelisting them).

The webhook technique consists of a POST request discussed in Section 4.4.2, thus the connection would be split by the TLS proxy. The body of the request is then analysed using the IPS policies, such as entropy requests. Although the data is encapsulated using encryption and base64, the entropy of the body is slightly different compared to the entropy of an image path [51], and thus liable to detection. Assuming that the IPS would track only the image field in the request body, the detection system might trigger an alert based on the entropy difference between encrypted data (the case of the covert channel) and plain text (the case of a normal Teams card) [52].

5.3.2 Message channel

Any message sent between peers in a company environment reaches the MS Teams server and then goes back to the company's network. It implies that any message for any domain needs to go through the server and use the same IP as the destination, i.e., the "closest" Microsoft server.
The utopian defence scenario would be to decrypt on the fly all the communication to analyse them, but this rarely happens in the real world. Instead, the most common solution involves logging all the messages of the users and using them in case of an investigation [53].

The message covert channel might be useful to transfer chunks of data at once, being extremely reliable and versatile. Nevertheless, it is essential to not overcome the threshold tracked in Figure 4.4, as this would help avoid being detected through the Kolmogorov-Smirnov Test [23] or other traffic anomaly tests.

5.3.3 Call channel

The call consists of one of the fastest and most advanced covert channels among the ones identified, but with many unreliable aspects. The assumption based on the threat scenario (see Section 3.4) is that the call uses the STUN protocol over UDP to exchange data between users as described in Section 2.3. In the STUN protocol, the first two bytes of the UDP packets are the channel ID, while the next two are the payload length, followed by the payload [54]. The firewall, in this case, does not prevent data extraction due to the unorderedness of UDP flow. The TLS proxy is instructed to not intervene in this process, leaving the IPS the last line of defence. The IPS could detect it by using the Kolmogorov-Smirnov Test [23], comparing the traffic distribution between two different calls. Nevertheless, the traffic anomaly detection can be easily avoided by limiting the traffic generated by the implant to 800 KB per second, trying to mimic the traffic sample in Figure 4.3, which shows the uploads byte during a MS Teams call sample. This channel comes with quite a stealthy footprint with a unique way to establish a covert channel, carrying one drawback: limited transmission time, as a call would last between 30-60 minutes, with rare cases above 120 minutes.

5.4 Summary

We analysed the properties of the covert channel based on the defined evaluation framework — bandwidth, robustness and efficiency. The result highlighted the characteristics of the discovered covert channel such as the stealthiest channel being the call covert channel and the webhook covert channel being unreliable. The message covert channel is then described as robust and reliable, lacking stealthiness in some cases; it has been selected for the control messages in the command-and-control system for these reasons. According to the measurement bandwidth, the message covert channel takes the first position, followed by the call and the webhook channel. Despite the message channel being the most performant, it can be detected if carrying too much information for a long time. The call covert channel takes over this scenario, where the data transfer is large enough and a continuous byte transfer, as a call would behave, is needed.

Chapter 6

Conclusions and Future work

In the conclusions section, we analyse the stated objectives and propose a potential solution to address cloud command-and-control using covert channels. The limitations section outlines the constraints of this work and emphasises the need for modifications in the design before its implementation. The future work section identifies areas where the thesis can be further enhanced. Lastly, the reflections section discusses ethical concerns related to covert channels and the design of this malware.

6.1 Conclusions

The rise in remote and hybrid working [55] has led to a higher demand for a different and greater variety of software to be installed on office workstations. As a result, the businesses' attack surface has expanded, creating new opportunities for potential security breaches. Although there have been initial efforts made to address the topic, a comprehensive analysis of covert channels in business communication platforms has been lacking, making state-of-the-art detection systems obsolete and inadequate for their complexities. For this reason, this research paper inspected MS Teams, a business communication platform at the core of many companies' workflows, to uncover its covert channels. A command-and-control malware that leveraged them was also developed to further analyse and compare their peculiarities.

The traffic analysis of the undocumented MS Teams API was performed, with traffic patterns for chat messages, calls, and standby phases being recorded, and their upload behaviour being tracked. A security assessment on the same API was carried out, uncovering the 10 weakness explored

in Section 4.3. These weaknesses implied the existence of storage covert channels to leak and transmit information between the internal and external networks using MS Teams as a carrier. To prove that messages, calls and webhooks can be leveraged as three different covert channels, a command-and-control malware was then designed to defy the detection security measures of the threat scenario in Section 3.4. Thanks to their traffic generation being indiscernible from a normal usage of the platform, and by interacting with Microsoft services rather than to the MasterBot directly, all three covert channels are able to pass through firewalls, IPSs and TLS proxy completely undetected.

Relying solely on traditional methods is insufficient for detecting modern covert channel attacks that utilise cloud-based command-and-control infrastructures, as we saw throughout this work. According to the paper [19], we want to advise cloud providers and application owners to step forward and to react proactively on this issue. As exemplified by the Dropbox case [16] and discussed in this thesis, application providers possess a comprehensive overview of the ongoing situation and should intervene when anomalous requests are detected among the incoming traffic.

Throughout this thesis, we have demonstrated the existence of covert channels in three different models. Based on the conducted security analysis, a command-and-control system was proposed utilising incoming and outgoing covert channels for the communication between the master (the attacker's side responsible for sending instructions and receiving query results) and the slave (the compromised computer executing the instructions given by the master).

The last goal of this work was to evaluate all the covert channels and establish their distinct properties through a comparative analysis. Different channels possess unique characteristics and they can be used for different use cases. The evaluation section (referring to Section 5.1) emphasised the suitability of specific covert channels for routine information exchange or their effectiveness in transferring large files between parties.

6.2 Limitations

The security assessment of Teams was conducted between February and March 2023. During this period, the findings were utilised and reported to establish a final malware design and define the covert channels. Subsequently,

during the evaluation and the final adjustment, the covert channels were again tested and proved to ensure the smooth functioning of the entire system. However, a few updates were made between March and April 2023 that introduced minor changes to the user interface as well as some modifications to the back-end.

Regarding this project, two changes in MS Teams software affect the reliability of the malware. Firstly, the call notification no longer arrived in .gzip compressed format encoded in base64, but rather solely encoded in base4. The content of this incoming notification was needed to accept, answer, or reject a call. Secondly, there was a change in the location of the cookie file, that is required to authenticate to the Microsoft API. Teams created a folder called "Network" and the file is now placed inside. While these modifications may seem minor, they could disrupt the flow of the channel and impact its functionality.

It is worth noting that Microsoft implements changes within a span of less than six months, and it is highly likely that new changes will be introduced in the near future. Therefore, the malware system requires constant updates to maintain its functionality, and some of the covert channels may not work effectively in a few years' time.

6.3 Future work

Despite the extensive work and security analysis conducted throughout the project, there remains one aspect that has not been fully addressed. Specifically, in the call covert channel discussed in Section 4.4.4. The malware successfully accepts the call, authenticates, and receives credentials from the signalling server. It then proceeds to authenticate with the relay server. However, no traffic is transmitted from the receiver user. The Microsoft architecture allows for multiple requests for relay credentials, but the association of these credentials with the incoming call was not fully comprehended.

However, once a connection is established between two users, it opens up the opportunity to create the call covert channel and transmit data through it. As a result, there is a slight remaining gap in this project. Due to the project's time constraints, it was not possible to fully comprehend the underlying architecture.

6.4 Reflections

The weaknesses found and described in Chapter 4.3 are reported to Microsoft on the 16th of June 2023. The product owner is informed about the possibility to create covert channels within the MS Teams application.

It is essential to emphasise that the purpose of this research and the exploration of covert channels is not to enable the creation of malicious malware or facilitate the extraction of data from systems. Instead, the main objective is to raise awareness about the potential security implications associated with such applications. This study aims to shed light on the vulnerabilities and risks that may arise within these communication platforms.

While the findings of this thesis demonstrate the feasibility of covert channels in the context of Microsoft Teams, it is important to note that the actual code and implementation details of these channels are not disclosed within this thesis. This decision is made based on ethical considerations and to prevent any potential misuse or harm that could arise from public availability of such information.

Looking ahead, this research opens up possibilities for further exploration and analysis. The techniques and concepts employed in this study can serve as a foundation for identifying covert channels in other applications or systems. By applying similar methodologies, it may be possible to discover covert channels in various contexts, highlighting the need for stronger security measures and improved threat mitigation strategies. However, it is crucial to approach such endeavours responsibly and with full consideration of the ethical implications involved. Any future work in this area should adhere to ethical guidelines and adhere to legal boundaries to ensure that the findings are utilised for legitimate and constructive purposes.

References

[1] B. W. Lampson, "A note on the confinement problem," *Commun. ACM*, vol. 16, no. 10, p. 613–615, oct 1973. doi: 10.1145/362375.362389. [Online]. Available: https://doi.org/10.1145/362375.362389 [Pages 2 and 12.]

[2] S. Samonas and D. Coss, "The CIA strikes back: Redefining confidentiality, integrity and availability in security." *Journal of Information System Security*, vol. 10, no. 3, 2014. [Page 7.]

[3] R. 2979. (2000) RFC 2979. Accessed: 10/2/2023. [Online]. Available: https://www.ietf.org/rfc/rfc2979.txt [Page 7.]

[4] NIST, "Guidelines on firewalls and firewall policy," https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-41r1.pdf. [Page 8.]

[5] R. 3644. (2003) RFC 3644. Accessed: 3/2/2023. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc3644 [Page 8.]

[6] R. 8446. (2018) RFC 8446. Accessed: 4/3/2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8446 [Pages 9 and 10.]

[7] R. 2246. (1999) RFC 3644. Accessed: 3/2/2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc2246 [Page 10.]

[8] NIST. glossary term. Accessed: 10/4/2023. [Online]. Available: https://csrc.nist.gov/glossary/term/intrusion_prevention_system [Page 11.]

[9] D. K. Michael G. Solomon, *Fundamentals of Communications and Networking*. Jones & Bartlett Learning, 2021. [Page 12.]

[10] M. Masse, *REST API*. O'Reilly Media, Inc., 2001. [Page 13.]

[11] L. Gupta. REST architectural constraints. Access on 2/3/2023. [Online]. Available: https://restfulapi.net/rest-architectural-constraints [Pages 13 and 38.]

[12] NIST. C2 command and control. Accessed: 1/4/2023. [Online]. Available: https://csrc.nist.gov/glossary/term/command_and_control [Page 14.]

[13] N. I. of Standards and Technology. Cloud definition. Accessed: 1/4/2023. [Online]. Available: https://csrc.nist.gov/glossary/term/cloud_computing [Page 15.]

[14] J. McFarland, "Covert channels: An overview," 12 2017. doi: 10.13140/RG.2.2.34969.47202 [Page 19.]

[15] B. Aloraini, D. Johnson, B. Stackpole, and S. Mishra, "A new covert channel over cellular voice channel in smartphones," *CoRR*, vol. abs/1504.05647, 2015. [Online]. Available: http://arxiv.org/abs/1504.05647 [Page 19.]

[16] L. Caviglione, M. Podolski, W. Mazurczyk, and M. Ianigro, "Covert channels in personal cloud storage services: The case of Dropbox," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 4, pp. 1921–1931, 2017. doi: 10.1109/TII.2016.2627503 [Pages 19, 23, and 72.]

[17] G. Shah and M. Blaze, "Covert channels through external interference," 2009. [Page 19.]

[18] M. Torkashvan and H. Haghighi, "CBC2: A cloud-based botnet command and control," *Indian Journal of Science and Technology*, vol. 8, 09 2015. doi: 10.17485/ijst/2015/v8i22/59773 [Pages 20, 21, 22, and 28.]

[19] K. Clark, M. Warnier, and F. Brazier, "Botclouds - the future of cloud-based botnets?" 01 2011, pp. 597–603. [Pages 20, 21, and 72.]

[20] M. Byrenheid, M. Rossberg, G. Schaefer, and R. Dorn, "Covert-channel-resistant congestion control for traffic normalization in uncontrolled networks," in *2017 IEEE International Conference on Communications (ICC)*, 2017. doi: 10.1109/ICC.2017.7996936 pp. 1–7. [Pages 20 and 68.]

[21] D. Frolova, K. G. Kogos, and A. V. Epishkina, "Traffic normalization for covert channel protecting," *2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus)*, pp. 2330–2333, 2021. [Page 20.]

[22] M. Zuppelli, L. Caviglione, and M. Repetto, "Detecting covert channels through code augmentation," in *Italian Conference on CyberSecurity (ITASEC'21)*, 08 2021, p. 12. [Page 20.]

[23] S. Zander, "Bro covert channel detection (BroCCaDe) framework: Scope and background," 2017. [Pages 20, 68, and 69.]

[24] M. A. Elsadig and Y. A. Fadlalla, "Network protocol covert channels: Countermeasures techniques," in *2017 9th IEEE-GCC Conference and Exhibition (GCCCE)*, 2017. doi: 10.1109/IEEEGCC.2017.8447997 pp. 1–9. [Page 21.]

[25] Y. Xin, L. Kong, Z. Liu, Y. Chen, Y. Li, H. Zhu, M. Gao, H. Hou, and C. Wang, "Machine learning and deep learning methods for cybersecurity," *IEEE Access*, vol. 6, pp. 35 365–35 381, 2018. doi: 10.1109/ACCESS.2018.2836950 [Pages 21 and 67.]

[26] M. A. Elsadig and A. Gafar, "Covert channel detection: Machine learning approaches," *IEEE Access*, vol. 10, pp. 38 391–38 405, 2022. doi: 10.1109/ACCESS.2022.3164392 [Pages 21 and 68.]

[27] M. A. Ayub, S. Smith, and A. Siraj, "A protocol independent approach in network covert channel detection," in *2019 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, 2019. doi: 10.1109/CSE/EUC.2019.00040 pp. 165–170. [Page 21.]

[28] H. Nafea, K. Kifayat, Q. Shi, K. N. Qureshi, and B. Askwith, "Efficient non-linear covert channel detection in TCP data streams," *IEEE Access*, vol. 8, pp. 1680–1690, 2020. doi: 10.1109/ACCESS.2019.2961609 [Page 21.]

[29] W. Lu, M. Miller, and L. Xue, "Detecting command and control channel of botnets in cloud," 10 2017, pp. 55–62. [Pages 21 and 67.]

[30] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic." in *BotSniffer: Detecting Botnet*

Command and Control Channels in Network Traffic., 01 2008. [Pages 21 and 67.]

[31] F.-H. Hsu, C.-W. Ou, Y.-L. Hwang, Y.-C. Chang, and P.-C. Lin, "Detecting web-based botnets using bot communication traffic features," *Security and Communication Networks*, vol. 2017, pp. 1–11, 12 2017. doi: 10.1155/2017/5960307 [Page 22.]

[32] L. Caviglione, "Trends and challenges in network covert channels countermeasures." *Applied Sciences*, vol. 11, no. 4, p. 1641, 2021. doi: https://doi.org/10.3390/app11041641 [Page 22.]

[33] R. Archibald and D. Ghosal, "Design and analysis of a model-based covert timing channel for Skype traffic," in *2015 IEEE Conference on Communications and Network Security (CNS)*, 2015. doi: 10.1109/CNS.2015.7346833 pp. 236–244. [Page 23.]

[34] E. W. Dijkstra, *The Humble Programmer*. New York, NY, USA: Association for Computing Machinery, 2007, p. 1972. ISBN 9781450310499. [Online]. Available: https://doi.org/10.1145/1283920.1283927 [Page 24.]

[35] Microsoft. Microsoft 365 and Office 365 URLs and IP address ranges. Access on 25/1/2023. [Online]. Available: https://learn.microsoft.com/en-us/microsoftteams/office-365-urls-ip-address-ranges [Page 26.]

[36] V. Gligor, *A guide to understanding covert channel analysis of trusted system*. National computer security center, 1993. [Pages 29 and 58.]

[37] I. E. T. Force. HTTP status code. Access on 23/04/2023. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc7231 [Page 38.]

[38] IETF. (2002) RFC 8446. Accessed: 23/4/2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3339 [Page 38.]

[39] Mozilla. CORS cross-origin resource sharing. Access on 26/4/2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS [Page 39.]

[40] MDN. CORS cross-origin resource sharing. Accessed: 25/4/2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Glossary/Preflight_request [Page 39.]

[41] Microsoft. Graph API Microsoft. Accessed: 9/5/2023. [Online]. Available: https://learn.microsoft.com/en-us/graph/api/resources/teams-api-overview?view=graph-rest-1.0 [Page 39.]

[42] Mitre. CWE list scoring system for weaknesses. Access on 24/5/2023. [Online]. Available: https://cwe.mitre.org/cwss/cwss_v1.0.1.html [Page 40.]

[43] Microsoft. (2008) What is a card MS Teams card. Accessed: 7/5/2023. [Online]. Available: https://learn.microsoft.com/en-us/microsoftteams/platform/task-modules-and-cards/what-are-cards [Page 47.]

[44] D. Chopra, H. Schulzrinne, E. Marocco, and E. Ivov, "Peer-to-peer overlays for real-time communication: security issues and solutions," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 1, pp. 4–12, 2009. doi: 10.1109/SURV.2009.090102 [Page 48.]

[45] B. H. Tay and A. L. Ananda, "A survey of remote procedure calls," vol. 24, no. 3, p. 68–79, jul 1990. doi: 10.1145/382244.382832. [Online]. Available: https://doi.org/10.1145/382244.382832 [Page 48.]

[46] Cloudfare. (2023) Glossary what is lateral movement. Accessed: 3/5/2023. [Online]. Available: https://www.cloudflare.com/it-it/learning/security/glossary/what-is-lateral-movement/ [Page 51.]

[47] L. Prechelt, "An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program." 04 2000. [Page 58.]

[48] M. Jonker, A. Sperotto, R. Rijswijk-Deij, R. Sadre, and A. Pras, "Measuring the adoption of DDoS protection services," 11 2016. doi: 10.1145/2987443.2987487 pp. 279–285. [Page 61.]

[49] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting botnet command and control servers through large-scale NetFlow analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: Association for Computing Machinery, 2012. doi: 10.1145/2420950.2420969. ISBN 9781450313124 p. 129–138. [Online]. Available: https://doi.org/10.1145/2420950.2420969 [Page 68.]

[50] Microsoft. (2022) Microsoft 365 and office 365 URLs and IP address ranges. Accessed: 2/3/2023. [Online]. Available: https://learn.microsoft.com/en-us/microsoftteams/office-365-urls-ip-address-ranges [Page 68.]

[51] A. ONeal. (2021) How many bits of entropy in Base64, Hex, etc. Accessed: 30/5/2023. [Online]. Available: https://therootcompany.com/blog/how-many-bits-of-entropy-per-character/ [Page 68.]

[52] N. Provos and P. Honeyman, "Hide and seek: an introduction to steganography," *IEEE Security & Privacy*, vol. 1, pp. 32–44, 2003. doi: 10.1109/MSECP.2003.1203220 [Page 68.]

[53] Microsoft. Report and analytics message logging. Access on 29/5/2023. [Online]. Available: https://learn.microsoft.com/en-us/microsoftteams/teams-analytics-and-reports/teams-reporting-reference [Page 69.]

[54] R. 5389. (2008) RFC 5389. Accessed: 3/5/2023. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5389 [Page 69.]

[55] allwork. Remote work has increased 159% in 12 years. Access on 30/5/2023. [Online]. Available: https://www.allworknow.com/remote-work-has-increased-159-in-12-years [Page 71.]

TRITA-EECS-EX-2023:608