Universiteit van Amsterdam

MSc. Systems and Networking Engineering
Cyber Crime and Forensics Track

Covert channel detection using flow-data

Author:
Guido Pineda Reyes
`guido.pineda@os3.nl`

Supervisors:
Pepijn Jansen
`pepijn.janssen@redsocks.nl`

July 7, 2014

Abstract

This research project is the second of two projects for the Master Education of System and Networking Engineering at the University of Amsterdam, and is the result of four weeks of research. It focuses on the detection of network-based covert channels through the study of patterns of behavior of the selected protocols within the context of historic flow-data. By analyzing the behavior of these protocols, which are ICMP, HTTP and DNS protocol, it was possible to obtain a baseline for comparing normal behavior and malicious behavior generated by the specific tools for each tunneling technique. Finally, an implementation of the proposed algorithms on a flow data-set was performed in order to verify their effectiveness, these algorithms were defined based on the analysis stage of this research. The final conclusion is that is possible to detect malicious activity generated by network-based covert channels, by establishing a baseline of normal behavior and comparing it with malicious behavior.

Acknowledgements

I would like to thank my supervisor, Pepijn Janssen for his help during this research project and Leo Willems, for his help in the read of proof of this document. I also would like to thank my family for their support and my wife Glenda for her love and care during this year. But mostly to God, who has given me this opportunity to continue with my studies and has blessed me throughout this year.

Contents

List of Figures

List of Tables

1 Introduction

Covert channels are effective mechanisms that enable communication via unauthorized methods. They can go undetected by security monitoring tools like IDS/IPS or firewalls. Network Covert Channels are based on the idea of tunneling, which allows encapsulation of any protocol within another, carrying hidden information inside specific fields of the protocol header. They can be used by an attacker for malicious purposes such as data exfiltration from a compromised system, botnet control, malware updates and many other types of usage that can be considered as illegal.

1.1 Related work

The detection of covert channel is a widely researched topic. There are several investigations into techniques for detecting covert channels. For example, through packet classification proposed by Dong Ping et al [1], in which covert information is encoded by modulating the varieties of packets on the Internet. Another technique for detecting covert channels, proposed by Jiangtao Zhai et al, who based their research on the behaviors of TCP flows which are modeled by the Markov chain composed of the states of TCP packets [2]. Another approach proposed by Vincent Berk et al, which describes a method for detecting covert timing channels based on how close a source comes to achieving that channel capacity [3]. Another approach for detecting specific DNS tunnels proposed by Wendy Ellens et al, which focuses on statistical methods to detect this kind of covert channels [4].

This research focuses on network-based covert storage channels with historic flow-data. Historical data is relevant to study because it contains information that has been collected over a period of time. And in some cases, research of this information may be needed to determine whether there has been any kind of malicious activity. In cases of forensic investigation, it may be relevant to conduct this type of research to determine if a system was compromised to some kind of suspicious or malicious attack.

1.2 Research questions

In order to perform this research project, the following question will be analyzed:

Is it possible to detect network-based covert channel malicious activity by using flow-data?

To answer this question, the following sub questions will be answered:

1. How do the selected covert channel techniques work?
2. What is the difference between normal traffic and covert channel traffic behaviour using the chosen techniques?
3. What algorithms can be used to detect covert channel traffic?
4. How can these results be validated?

1.3 Approach

The approach of this project is to investigate the chosen tunneling techniques, and use different tools in order to reproduce malicious traffic that will be analyzed in a testing environment. The characteristics of the normal traffic behaviour and the malicious traffic behaviour will be studied to determine if it is possible to detect such traffic, based on the comparison of normal and malicious behaviour. This approach will be focused on historic flow-data, therefore, previously captured network traffic considered as normal network traffic will be reproduced in the testing environment for further analysis. Furthermore, malicious traffic will be generated by the chosen tools and it will be reproduced for further analysis. This flow-data will be analyzed and based on the behaviour analysis, different algorithms should be proposed in order to detect malicious traffic. Finally, a validation procedure should be established to validate the effectiveness of the proposed algorithms.

1.4 Scope of the project

This project is focused on the research of network-based covert channels that use the ICMP, HTTP and the DNS protocol, as they are one of the most common protocols in use on the Internet, and they can be misused by an attacker to exfiltrate unauthorized information. Furthermore, the analysis will be performed on historical flow-data that will be provided by the sponsoring company. The Netflow standard that will be used in this research is the current version (v10), also known as IPFIX.

1.5 Netflow overview

Netflow is a network traffic monitoring tool, initially developed by Cisco, and now it has evolved to become an IETF standard called IP Flow Information Export (IPFIX) [5], that describes the method for a flow-collector, to export statistics about IP packets passing an observation point in a network during a certain time interval. Every IP packet that belongs to a particular flow, has a set of common properties, also called attributes, and these attributes are used to distinguish one flow from other flows. When a new packet arrives to the collector, it determines whether it belongs to the current flow, or to any other flow. The main attributes that are used to uniquely identify a flow are: source address and port number, ingress interface, destination address and port number, network layer protocol, and type of service (TOS).

Also, the accumulated traffic in bytes and packets per flow is recorded. It is relevant to note that the payload is not recorded in flow-data, therefore any type of analysis should be made based on behaviour.

IPFIX, the latest version of Netflow (v10), extends NetFlow v9 by adding new attributes. This attributes are used to gain more information about individual flows and also IPFIX allows to export bidirectional flows, which is helpful when analyzing this kind of information [6]. For example, some tools provide the ability to add DNS information such as DNS query types and responses to the flows. This information can help to get a better understanding about the flows and in general, the network traffic. However, for IPFIX, the support provided by the currently used collection framework is limited and might not be fully supported in every flow collector [7].

2 Experiments and data gathering

This section describes the experimental procedures performed in this research. First, the description of the experimental environment is made, which was used in order to conduct the experiments. Secondly, a description of the tested tunneling techniques is made and how the experiments were conducted.

2.1 Experimental environment

The system architecture of the experimental environment consists of one flow collector, the tool used in this project is nProbe™[8], which gets all the incoming network traffic from sources such as pcap format files or replayed packets by using the tool tcpreplay [9]. It is also possible to configure the flow collector to receive network traffic from a specific network card interface, but for the purpose of this research project, it is out of the scope, since it focuses on analyzing historical data. This network traffic is processed and latter it is saved in a MySQL database as flow-data format for further analysis, see Figure 1.

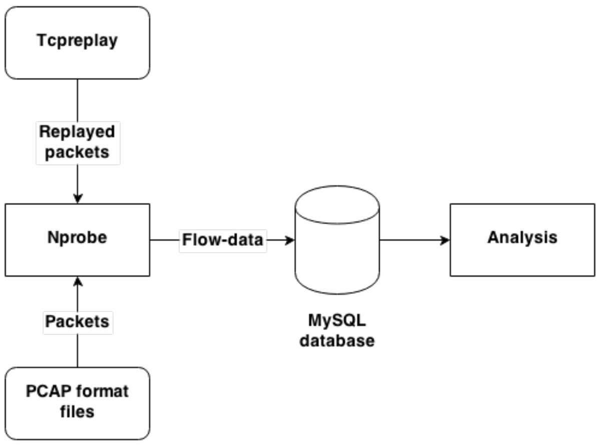


Figure 1: Experimental setup

2.2 Covert channel techniques

Covert channels can be deployed in many ways by using different kinds of protocol headers to carry hidden data. This research focuses on commonly used protocols such as the Internet Control Message Protocol (ICMP), the Domain Name System protocol (DNS) and the Hypertext Transfer Protocol (HTTP).

The following techniques were tested:

- ICMP tunneling
- ICMP reverse shell
- DNS tunneling
- HTTP reverse shell

ICMP tunneling

This tunneling technique uses ICMP echo and reply packets to carry hidden data. The architecture consists of one client, one destination and a proxy, see Figure 2. The client communicates with the proxy by using ICMP echo requests, and the proxy forwards these packets by opening a TCP connection to the destination, then, in the reply from the destination to the client, those packets are converted into ICMP replies by the proxy and then delivered to the client. All the communication is transported in the "Data" field of the ICMP packet. This technique can be exploited by malicious software in order to leak sensitive information out of the compromised machine. For the purpose of this research, the tool that was used is Ptunnel [10]. The proxy is a virtual machine running Ubuntu 14.04 LTS that can be reached over the Internet, and is not blocking ICMP messages. The destination is the same proxy server, which has a web server running on port 80 and an SSH server running on port 5022. The client connects to the proxy over the Internet, and it is running a Kali Linux as the operating system.

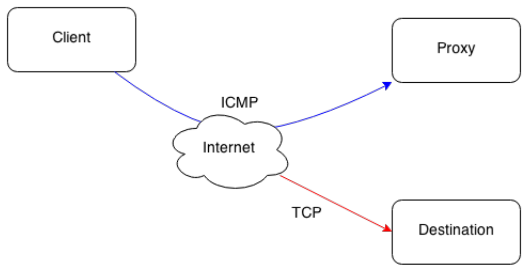


Figure 2: ICMP tunneling

This technique was tested in several ways, by performing different types of behavior. Since a web server was running, a simple login web page was set, where the client has to enter a user name and password, this checks if the user is correct and displays a successful login, otherwise, an error will display. Also, files of different sizes were downloaded from the web server. Another type of behavior was recorded with the SSH server, by typing random commands in the terminal, and also downloading files of different sizes using an SFTP client, that runs over the SSH protocol to communicate with the endpoint.

In order to capture this traffic generated by the client and the proxy, a network traffic sniffer was located at the client side. This network traffic, captured as pcap format file, will be replayed and analyzed by the flow collector to determine the characteristics that identify this type of network traffic.

ICMP reverse shell

This is a technique that uses ICMP packets to carry hidden information in the "Data" field. As well as the ICMP tunnel technique, it uses echo requests and echo replies to communicate between the two endpoints. The network architecture consists of the client which in this case would be the victim's machine and the server which in this case is the attacker's machine, see Figure 3. When the server is running, it waits for the client to connect in order to send remote commands to it.

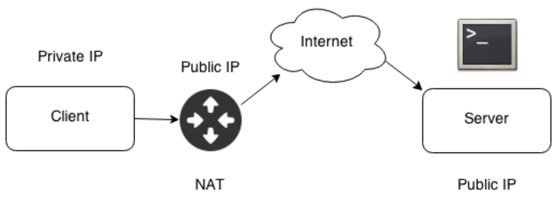


Figure 3: ICMP Reverse shell

The tool used for this purpose is ICMPsh [11]. This tool also provides a time delay between command option, that waits for a specific time between the commands sent from the server to the client. This tool was tested, while random commands were issued at the client, like getting the configuration of the network, creating and erasing files, reading the content of text files, etc, with the purpose of trying to emulate the behavior of an attacker. This setup was tested running the client on a Windows 7 and Windows 8 machines that connect to the server through the Internet. The server is running on an Ubuntu 14.04 LTS server. The client was also tested for Windows Server 2003 with no successful attempts.

The network traffic generated by this technique was recorded with a network sniffer at the client side.

DNS tunneling

This technique allows to carry hidden information by using DNS queries and replies. This technique can be exploited in systems where the DNS incoming and outgoing traffic is allowed, and this is the case for most systems, therefore it can be exploited by several types of malicious software like Feederbot (Dietrich, 2011) and Moto (Mullaney, 2011), where both use DNS TXT records for command control. The architecture of this system consists of the client side, the DNS tunnel server, which is the authoritative name server for the controlled domain, and this DNS tunnel server is typically accessible over the Internet and controlled by the client. The client side initiates a DNS request to the authoritative name server in order to start sending data, and this information included in the DNS payload, can be encoded to increase performance by using different techniques such as Base32, Base64, Hex, etc. The tool used for this research is Iodine [12], where the client side is a machine that connects to the DNS tunnel server through the Internet and there are services running like SSH server and SFTP to transfer files, see Figure 4.

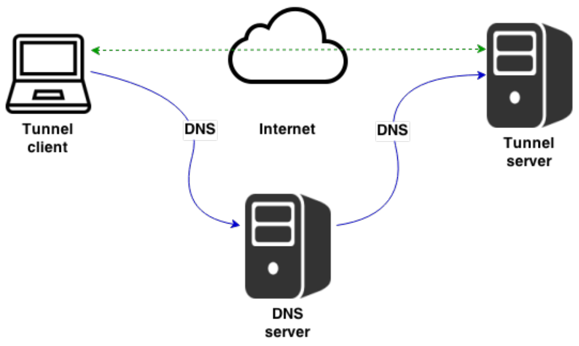


Figure 4: DNS tunneling

The DNS queries performed by the client are sent to the DNS server which is controlled by the client. This traffic is processed as a regular request, and at the end of this operation, this information is handled by the tunnel server, which retrieves the encapsulated data and replies to the DNS query by encapsulating the response in the answer section of the DNS response message.

This technique was tested by imitating behaviors like downloading files from different sizes up to 100 Megabytes and using an connecting to an SSH server and typing random commands at the tunnel server. All the network traffic generated was recorded with a network sniffer at the client side.

HTTP reverse shell

This is a tunneling technique that uses the HTTP protocol to send hidden commands in GET and POST methods. It consists of two main components, the client, which is the compromised machine, and the server, which is the attacker's machine that is listening to port 80, where the client connects. Once the client connects to the server, it polls for incoming commands to the server. For this research project the tool used is Matahari [13], which also offers several types of polling types like *insane*, *aggressive*, *normal* and others. These polling types are used to evade IDS/IPS and firewall systems since they use a time interval between requests to the server. For this project, the polling techniques that were tested are: adaptive (dynamically increases polling period when no commands are received until reaching stealth type), aggressive (25 seconds between requests), normal (60 seconds between requests), polite (5 minutes between requests) and ids-evasion (randomly selects the time between the polling intervals). The client is running on a Kali Linux machine and the server is running on an Ubuntu 14.04 LTS and they are connected over the Internet, see Figure 5. All this traffic generated by the communication between these two hosts was recorded with a network sniffer on the client side.

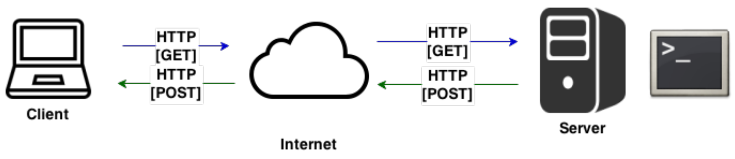


Figure 5: HTTP reverse shell

2.3 Data gathering

To be able to differentiate between regular and malicious traffic, several captures of all previously mentioned techniques were conducted in order to establish a baseline of behavior-based analysis of network traffic once it has been converted to flow-data format.

Network captures: Regular traffic

In order to have a baseline to compare the *malicious* traffic to *normal* activity from different users was captured during the period of one week. The summary of this captured traffic is shown in Table 1. To capture ICMP traffic, experimental ping messages of different types were sent, for example, messages with different sending intervals between 0.1 and 1 second message, with a larger size to the default size, which is 64 bytes, messages to firewall protected devices, messages to virtual machines, where it was observed that the messages are redirected by it the gateway to which the devices are connected. To capture DNS traffic, besides the regular DNS traffic, it also worth to take into account DNSSEC traffic, which was generated through the dig command with requests to servers that have DNSSEC enabled. Finally, HTTP traffic was captured during one week at a user's computer, recording all communication that uses HTTP.

Table 1: Captured regular network traffic summary

Protocol	Total bytes of traffic (MB)	Total packets
ICMP	698.5	3445152
DNS	1638.6	3981600
HTTP	1956.27	1818293

Network captures: Malicious traffic

Every session and network traffic was recorded by using a network sniffer for all the previously described techniques, several pcap files were generated, by performing different kinds of behavior. This traffic will be considered as *malicious* traffic. The capture time of this network traffic was not fixed, therefore, some pcap files were extensible large and others were relatively small. Table 2 shows a summary of the captured traffic.

Table 2: Captured malicious network traffic summary

Technique	Total bytes(MB)	Total packets
ICMP tunnel	3957.08	4491868
ICMP reverse shell	196.26	3481308
DNS tunnel	2746.75	3376230
HTTP reverse shell	311.39	470985

Flow-data

Once the packet capture procedure has been completed, it is possible to reproduce this traffic in order to convert it into flow-data by the flow collector. This information will be stored in a MySQL database, and this data will be analyzed later. Two different databases will be created, one for *regular* traffic, and one for *malicious* traffic.

The summary of the flow-data is shown in Table 3 for regular traffic and in Table 4 for malicious traffic.

Table 3: Flow-data summary for regular traffic

Protocol	Total bytes (MB)	Total packets	Total bidirectional flows
ICMP	698.5	3445152	169
DNS	1638.6	3981600	53490
HTTP	1956.27	1818293	40107

Table 4: Flow-data summary for malicious traffic

Technique	Total bytes (MB)	Total packets	Total bidirectional flows
ICMP tunneling	3957.08	4491868	30
ICMP reverse shell	196.2	3481308	75
DNS tunneling	2746.7	3376230	172
HTTP reverse shell	311.39	470985	166

3 Data analysis

This section describes the analysis procedure for every type of protocol used in this research in order to get a better understanding of how they work and how they can be interpreted while working with flow-data. This research project is focused only in analyzing flow-data, but a quick overview on how these protocols are used in this techniques, can help to have a better understanding of the operation of them.

3.1 Protocol level

In this section, the analysis will focus on the protocols that are used by the tunnel techniques. This analysis will allow a better understanding of how information is sent via these tunnels. Furthermore, this analysis will allow us to understand the behavior of these techniques in flow-data format.

ICMP

Since the tested tunneling technique only uses the “Echo request” and “Echo reply”” to send messages, this research is only interested in this type of messages, therefore, it is possible to filter out other types of ICMP messages. In the normal operation of this protocol, Echo request messages are generated in the source with an ICMP type 8 and subsequently, they are replied in the destination with an ICMP type 0. According to the standard specification (RFC 792), the data received in the echo message must be returned in the echo reply message [14]. Figure 6 shows the structure of an ICMP packet for echo and reply messages, where the “Type” field is the ICMP message type (8 for echo requests and 0 for reply messages). The “Code” field is always 0 when echo and reply messages are sent. The “Checksum” field is used for error controls and the “Data” field contains data according to the specific type and code values. For echo requests and echo reply messages, this field contains numbers and for other implementations this field contains letters of the English alphabet, see Appendix 1, where the “Data” field is represented by the alphabet letters in the ASCII dump.

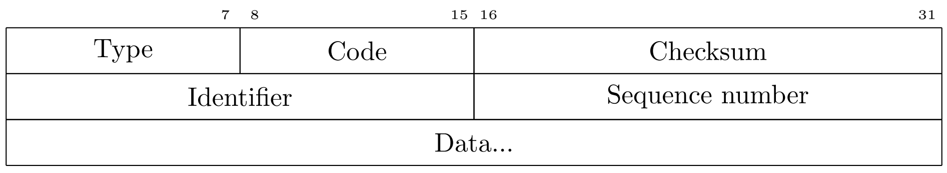


Figure 6: Echo or Reply message

ICMP tunnel

This tunneling technique uses the “Data” field of the ICMP packet header to carry information. For the tested tool (Ptunnel), this information is not encoded, compressed or encrypted, therefore it is possible to read the content of the packet with a network sniffer, see Appendix 1 for examples of how this information is transported using the ICMP headers. The length of the “Data” field can be flexible, carrying a decent amount of information.

ICMP reverse shell

This technique uses the ICMP echo request and echo reply messages to transfer information, therefore it is also possible to filter out other types of ICMP messages. The information carried in the “Data” field is sent without compression or encrypted, and it is possible to retrieve it with a network sniffer, see Appendix 1 for an example of how data is embedded in the packet header. With the specific tested tool (ICMPsh), the echo requests are performed by the attacker’s machine and the reply messages are performed by the victim’s machine. It is also noticeable that the TTL for every packet it is never less than 230, which shows strange behaviour, and this should be checked when working with the flow dataset.

DNS

In this technique the client, will send DNS requests to the tunnel server. The first requests are referred as standard queries, but later the information of the DNS packets is classified as “Unknown operation” or

"Malformed packet", indicating some suspicious behaviour, that will have to be checked in the flow-data set. When comparing this behaviour with *regular* DNS packets, for every request there is a response, this behavior is different to that found during the analysis of *malicious* traffic, since for one DNS request, there are many responses, which are assumed to be the data being transferred from the client to the tunnel server. For DNSSEC traffic, the amount of outgoing traffic is much higher than the incoming traffic, which also must be checked in the analysis of the flow dataset.

HTTP

For this technique, the client sends request messages to the server for commands to execute locally, these requests are sent in a GET method that contains information such as a URI, a protocol version, client information and the content of the message, which is encoded in Base64 for the tested tool (Matahari). The response message from the server to the client, is an HTTP response message with code 200 OK, stating that the request was successful, see Appendix 2 as an example of command requesting for the available space in the client's disk.

If for any reason, the client gets an empty message from the server, the client will not execute any command locally.

3.2 Flow level

In this section, the analysis will focus on the flow level, that is by analyzing the output of the flow-data set which was generated by reproducing the captured packets and later captured by the flow collector. For this purpose, Netflow version 10, also known as IPFIX was used. It allows a flexible way to analyze flow-data by specifying different kinds of key fields in a given template. Therefore, different types of templates, specific for each protocol were used, and this is for *regular* traffic and *malicious* traffic.

ICMP

An analysis of the ICMP protocol in the flow-data set is performed. First, the behavior of the protocol will be analyzed under normal conditions and in different situations, then the behavior of the protocol is analyzed when being used as a tunnelling technique.

The template used for ICMP has the key fields to analyze the behaviour of the flows, see Table 6.

Table 5: IPFIX template for ICMP

Field	Description
IPV4_SRC_ADDR	IPv4 source address
IPV4_DST_ADDR	IPv4 destination address
PROTOCOL	IP protocol byte
IN_BYTES	Incoming flow bytes (src ->dst)
IN_PKTS	Incoming flow packets (src ->dst)
OUT_BYTES	Outgoing flow bytes (dst ->src)
OUT_PKTS	Outgoing flow packets (dst ->src)
MIN_TTL	Min flow TTL
MAX_TTL	Max flow TTL
ICMP_TYPE	ICMP Type * 256 + ICMP code

Regular ICMP

When using the ping utility, which uses ICMP, two unidirectional flows are generated. A flow from source to destination that contains the echo request, and another flow from the destination to the source, that contains the echo reply. The tool used (nProbe) by default generates a unique bidirectional flow, therefore, in this research, for every analyzed flow, it is referred as a bidirectional flow. For tests where the Ping tool is used, messages of different lengths and even different size messages and sending interval were generated. For each message, a flow is generated, and a total of 169 flows were generated. During the analysis, it may be noted that one difference between the flows generated, is the number of packets and bytes sent and received, but always a symmetry between the number of bytes and packets sent and received is kept. That is, the packet ratio or byte ratio is almost always equal to 1. There are some cases where this value is slightly less than one, and that is due to packet loss, but for the analysis performed, this value varies from 0.9833 to 1 for the packet ratio and 0.8519 to 1 for the byte ratio. Since, the packet and byte ratios show the same behavior, for the purpose of this research, we will analyze only the packet ratio, see Figure 7a This analysis is valid only for ICMP echo request messages that have been answered back by the ICMP echo reply messages. When sending messages to a device that is blocking ICMP messages or that is offline, the number of incoming bytes is zero, because no echo reply messages are being received, therefore the rate of bytes and packets will be zero (Received packets/Sent packets). Other messages were sent to virtual machines, and the behavior that could be seen in the flow format was that the echo reply message is sent by the bridge device where the virtual machine is connected to, and this is because ICMP messages are being redirected, but the flow-data still show symmetry in the sent and received packets and bytes. Another variable, that can be used to determine regular or malicious behavior is the number of bytes per packet per flow. For regular ICMP traffic, this value is between 28 and 84 for bytes sent and received, depending on how the ping messages were generated, see Figure 7b. When the size of the message is altered by making a ping test with more than the default value, which for Linux machines is 56 bytes, which are translated to 64 ICMP data bytes when combining with the 8 bytes of ICMP header data, and for Windows machines this value is 32 bytes for a regular Ping message, which are translated to 40 ICMP data bytes. To all of these values, the IP header of 20 bytes is also added. But, it is possible that the number of bytes per packet per flow can be bigger than this range of values, when ICMP messages are being sent using different sizes of messages by altering the default value, and this is still not considered to be malicious traffic, since it can be used for troubleshooting purposes. However, it was found that the symmetry between the number of sending and received bytes is still maintained. For this analysis, the amount of packets sent is considered, since it's exactly the same as the amount of packets received. The variable ICMP_TYPE, which shows the ICMP type and the ICMP code per flow, shows the value 2048 for every analyzed flow that has an echo reply and echo response. This value represents the ICMP type times 256 plus the ICMP code, which is 8 for echo requests and 0 for echo reply, and the ICMP code is always 0 for this ICMP type. Therefore, 2048 represents a successful ping connection. The variable MIN_TTL or MAX_TTL show typical values that vary from 48 to 128 for the analyzed flows, see Figure 7c

ICMP Tunnel

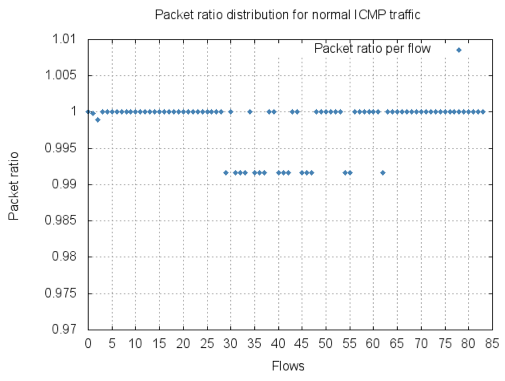
For this technique, one bidirectional flow is generated per communication attempt regardless of the amount of data being transferred. For this technique, 30 flows were generated. Since the tested tunneling technique, in order to work, must not have the ICMP messages blocked at the sender or receiver side, it is possible to filter out every flow that has the byte or packet ratio equal to zero, see Figure 8a.

For every type of behaviour that was recorded, the byte or packet ratio is higher than expected, for example, when testing a user logging into a test web page using this technique, the number of bytes received is approximately twice the number of bytes sent. Another anomaly can be found, when downloading a 5 MB size file, since the the amount of received bytes is almost 300 times the amount of the sent bytes. This behaviour was found for every test when downloading a file with this technique.

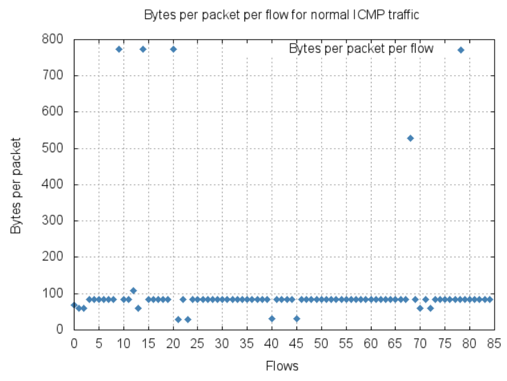
The number of bytes per packet per flow, do not show much difference from normal behaviour, ranges for outgoing bytes per packet per flow varies from 56 to 104 bytes per packet, while for the incoming bytes, this range varies from 60 to 970 bytes per packet, see Figure 8b.

The ICMP_TYPE field for all flows show a value of 2048, and as it was previously described, it represents a successful connection between two hosts, therefore it is not relevant in the analysis.

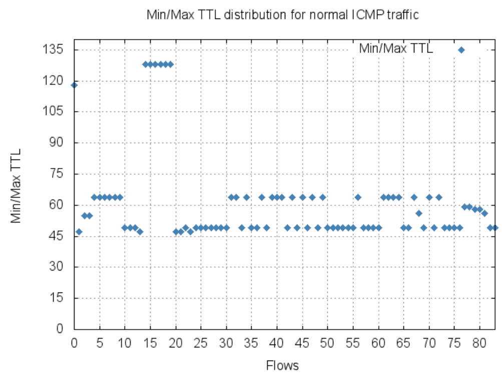
Also the variable MIN_TTL or MAX_TTL do not show any difference from regular ICMP traffic, although, its values are 128 for every flow, it is still considered to be normal behaviour, see Figure 8c.



(a) Packet ratio distribution

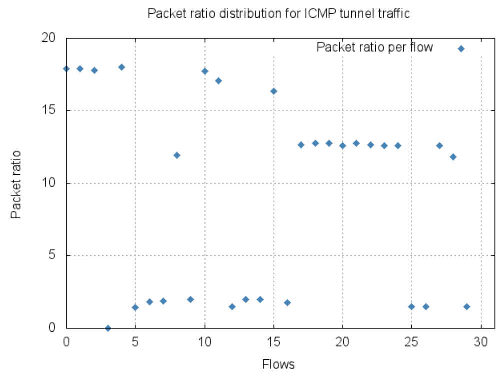


(b) Bytes per packet

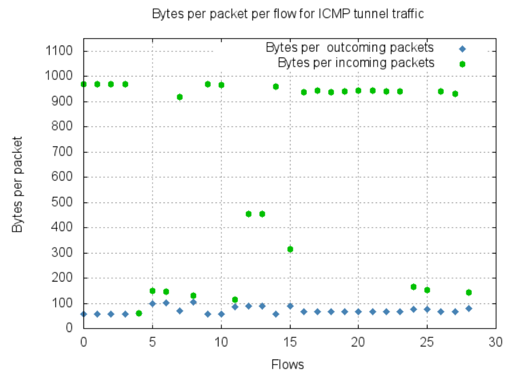


(c) Min/Max TTL

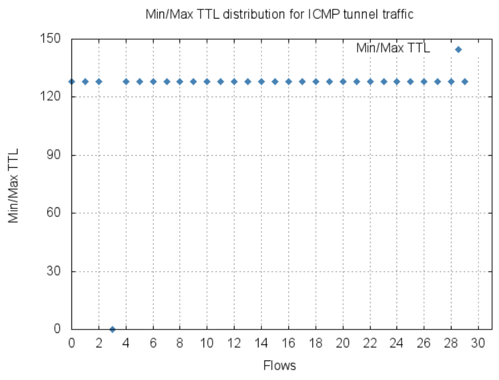
Figure 7: Regular ICMP traffic variables



(a) Packet ratio distribution



(b) Bytes per packet



(c) Min/Max TTL

Figure 8: Tunnel traffic variables

ICMP reverse shell

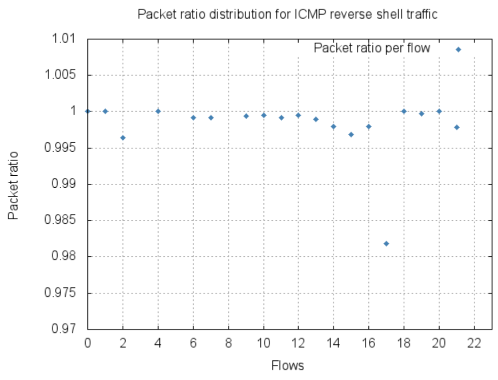
This technique may generate several flows per session. Since a reverse shell is generated on the attacker's side, it can be possible that the attacker would generate ping commands from this command shell, therefore, it was found that these messages are also part of the communication and can be recorded as flow-data. These ICMP messages are seen as regular ICMP traffic, therefore it is not interesting to analyze them. But, other flows are of interest to analyze.

When analyzing the bytes or packet ratio, there seems not to be a marked difference between regular traffic and malicious traffic, the amount of incoming bytes or packets is nearly the same as the amount of outgoing bytes or packets, therefore the packet ratio is close to one for every flow, see Figure 9a.

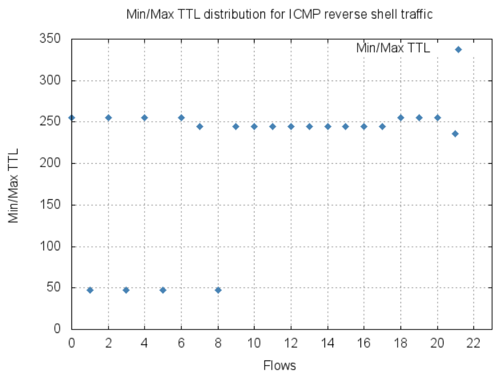
One characteristic of these flows, is that the TTL value of every flow is never less than 230, and for the same flow, the amount of sent and received bytes is particularly high, from 20 MB up to 100 MB, which is not normal behaviour for ICMP messages. This minimum and maximum TTL value being more than 230, also shows suspicious behaviour, therefore another test was performed in order to possibly determine that can be causing it. A server located 22 hops away running CentOS, shows a TTL value of 236, which is something not possible in normal conditions, because the TTL value for an echo request to the same server is 48, see Figure 9b which shows the distribution of this value for every analyzed flow.

Another pattern that can be found is that the number of outgoing bytes per packet per flow is always around 28 regardless the amount of traffic being sent. In fact, the standard deviation of this value, for all the flows, is around 0.064, which means that there is not much variation of this value for this analysis. This value of 28 bytes per packet per flow is also in the range of regular ICMP traffic as it was previously discussed, therefore it does not show any suspicious behaviour by itself, see Figure 9c which shows the distribution of this value for ever analyzed flow.

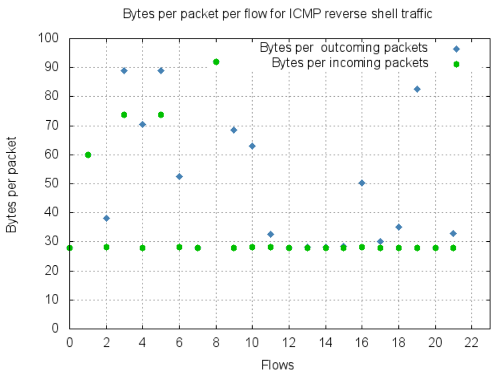
The ICMP_TYPE field for all flows show a value of 2048 as the previously described tunneling technique, therefore it is also not relevant in this analysis.



(a) Packet ratio distribution



(b) Min/Max TTL



(c) Bytes per packet

Figure 9: ICMP reverse shell variables

DNS

In this section, an analysis of the behaviour of the DNS protocol will be analyzed within the flow-data set. The template that was used for the analysis of this protocol is presented below, see Table 6.

Table 6: IPFIX template for DNS

Field	Description
IPV4_SRC_ADDR	IPv4 source address
IPV4_DST_ADDR	IPv4 destination address
PROTOCOL	IP protocol byte
IN_BYTES	Incoming flow bytes (src ->dst)
IN_PKTS	Incoming flow packets (src ->dst)
OUT_BYTES	Outgoing flow bytes (dst ->src)
OUT_PKTS	Outgoing flow packets (dst ->src)
MIN_TTL	Min flow TTL
MAX_TTL	Max flow TTL
DNS_QUERY	DNS query
DNS_QUERY_ID	DNS query transaction Id
DNS_QUERY_TYPE	DNS query type (e.g. 1=A, 2=NS..)
DNS_RET_CODE	DNS return code (e.g. 0=no error)

Regular DNS

For regular DNS traffic, the analysis shows some patterns in the normal behaviour. A total of 53490 flows, considered as regular DNS traffic were analyzed. The analysis shows that the traffic packets ratio for sent packets over the number of packets received, is always equal to 1 for 96.6% of the flows, that means that there is a degree of symmetry in the number of incoming and outgoing packets, 2.57% of the flows have a packet ratio of equal to zero, meaning that the received packets is zero, see Figure 10. However, having a packet ratio equal to 1 is not the case for the byte ratio, because the values vary depending on the type of the DNS request, eg, for DNSSEC traffic type, which generates relatively high amounts of incoming traffic, the number of incoming bytes is greater than the number of sent bytes and is on the order of about 20 times the number of outgoing bytes, but even for the same DNSSEC traffic, the packet ratio is always equal to 1. Therefore, the byte ratio will not be analyzed in this research for this specific type of network traffic. It is also interesting to note that the range for the packet ratio varies between 0.25 and 1 for regular DNS traffic.

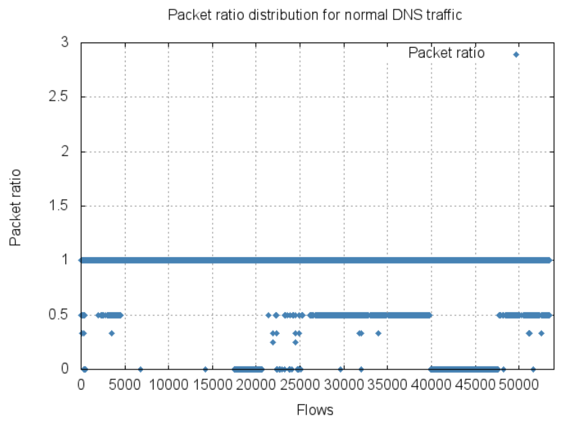
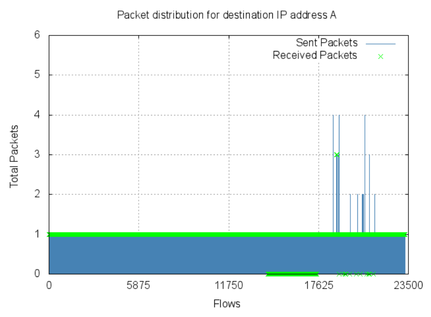


Figure 10: Packet ratio distribution for regular DNS traffic

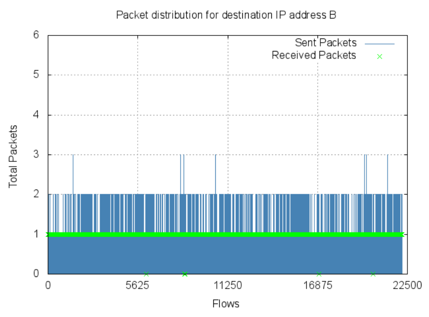
Another type of analysis will focus on determining what are the top IP addresses to which there is the greatest amount of flows. Once these destinations are identified, an analysis of the number of sent and received packets per flow will be made. The top ten flows are shown in Table 7. For example, for IP address A, 99.85% of the flows have one packet that is being sent, 0.09% of the flows have two packets that are being sent and 0.03% of the flows have 3 or 4 packets that are being sent. The same analysis was also performed for the number of received packets per flow. IP address A, shows that 99.3% of the flows have 1 incoming packet per flow, 0.009% of the flows have 3 incoming packets per flow and 0.7% of the flows have 0 incoming packets per flow. This analysis was performed for all the connections, and it shows that for the majority of the flows the number of received packets per flow is 1. It is clear to note that for the analyzed data, the maximum number of sent packets per flow is 4, the maximum number of received packets is 3, and for most flows, the number of sent and received packets per flow is 1. Figure 11 shows this packet distribution for the top four destination IP addresses. An analysis on the average and standard deviation for the packet distribution was analyzed, Table 8 shows that the standard deviation for every flow is not bigger than 0.1, for the top 4 analyzed flows, this calculation was performed for every connection with a distinct destination IP address, and the maximum value for the standard deviation is 0.5, which means that the number of packets for every flow is about the same and the values are not too separated from each other.

Table 7: Number of packets per flow distribution

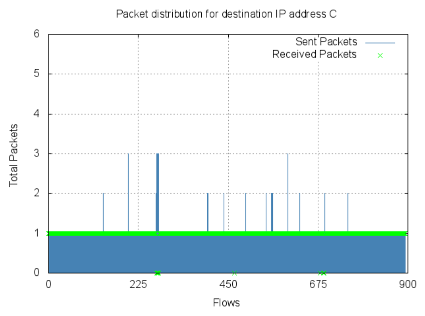
Destination IP	**Flows**	**# flows with n sent pkts**				**# flows with n received pkts**			
		1	**2**	**3**	**4**	**0**	**1**	**2**	**3**
A	23287	23251	21	8	7	163	23122	0	2
B	22190	20860	1323	7	0	6	22184	0	0
C	895	868	17	10	0	11	884	0	0
D	764	764	0	0	0	0	764	0	0
E	544	544	0	0	0	0	544	0	0
F	472	457	312	3	0	0	470	0	0
G	254	254	0	0	0	0	254	0	0
H	234	234	0	0	0	0	234	0	0
I	234	234	0	0	0	88	146	0	0
J	190	190	0	0	0	0	190	0	0



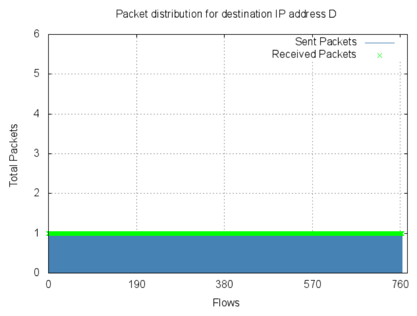
(a) Destination IP address A



(b) Destination IP address B



(c) Destination IP address C



(d) Destination IP address D

Figure 11: Packet distribution for top 4 destination IP addresses

Another approach in order to detect suspicious activity, is by analyzing the DNS fields provided by

Table 8: Packet distribution analysis

Destination IP	# Flows	Avg sent_pkts	Std dev sent_pkts	Avg received_pkts	Std dev received_pkts
A	23287	1.0025	0.075	0.9932	0.0845
B	22190	1.063	0.2396	0.9997	0.0164
C	895	1.0413	0.2490	0.9877	0.1102
D	764	1	0	1	0

the IPFIX export. The DNS_QUERY field shows all the DNS host names, the DNS_QUERY_ID field is not too interesting for this analysis, since it is a unique identifier value for the DNS query. The DNS_QUERY_TYPE for all the analyzed flows is shown in Table 9, and it shows that 75.5% of the flows are related with the "A" DNS type, 15.03% of the flows contain the "DNSKEY" DNS type, and this is because DNSSEC traffic was also generated. And the third biggest DNS type present in this analysis is the "AAAA" type which is used for IPv6 addresses. For this analysis, is clear that the majority of flows for DNS traffic contain the "A" DNS type, commonly used to map hostnames to an IP address.

Table 9: DNS_QUERY_TYPE distribution

DNS_QUERY_TYPE	# of flows	%	Type	Meaning
1	40395	75.5	A	A host address
2	1807	3.39	NS	An authoritative name server
6	4	0.007	SOA	Marks the start of a zone of authority
12	438	0.08	PTR	A domain name pointer
16	1	0.002	TXT	Text strings
28	2461	4.6	AAAA	IPv6 Address
33	18	0.03	SRV	Server Selection
43	723	1.35	DS	Delegation Signer
48	8083	15.03	DNSKEY	DNSKEY

The DNS_RET_CODE field, which indicates the resulting state of a request for the analyzed flows are shown in Table 10. It shows that for most flows (97.7%), the DNS queries were successful, and for some queries, there is a nonexistent domain response.

Table 10: DNS_RET_CODE distribution

DNS_RET_CODE	# of flows	Description
0	52272	No Error
2	39	Server Failure
3	1132	Non-Existent Domain
5	47	Query Refused

The last approach is to determine the amount of sent and received bytes per destination IP address. For a covert channel using DNS traffic, it could be expected to produce high amounts of traffic, therefore, it is relevant to consider this approach. The different domains will be identified and the amount of sent and received bytes to each IP address. For the analyzed flows, destination IP address A has a total of 7734.77 MB received, and a total of 1524.7 MB sent, destination IP address B, has a total of 1601.4 MB sent and 3449.4 MB received, Figure 12 shows the top 30 destination IP addresses.

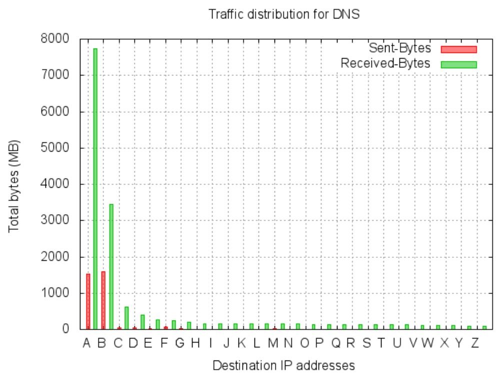


Figure 12: DNS traffic distribution for the analyzed flows

DNS tunnel

For this technique the total number of flows collected is 172. The analysis of the packet ratio shows that for 68.02% of the flows, this ratio is equal to 1, for 23.84% percent of the flows, the ratio is zero, which means that the number of received packets is zero. There are other values in the packet ratio that may raise an alarm on suspicious behaviour, and those are the values with rates that vary from 1.7 to 2.5, which means that the amount of received packets is almost twice as much as the sent bytes, see Figure 13.

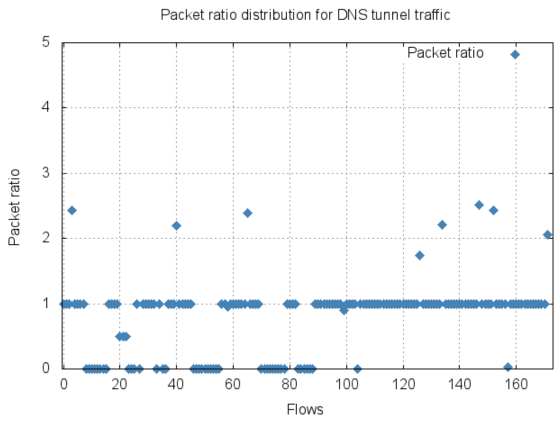


Figure 13: Packet ratio distribution for DNS tunnel traffic

Other analysis is to determine the destination IP address to which the largest amount of flows are directed to, and once this IP addresses are detected, an analysis of the packet distribution will be made in an attempt to detect suspicious behaviour. Table 11 shows the different destination IP addresses, where A is the IP address of the DNS tunnel server, B and C are IP addresses of regular DNS servers. The analysis on the standard deviation for the sent and received packets shows that for destination IP address A, where the tunnel server is, these values show suspicious behaviour. The average value suggests that a big amount of sent and received packets are detected, see Table 12.

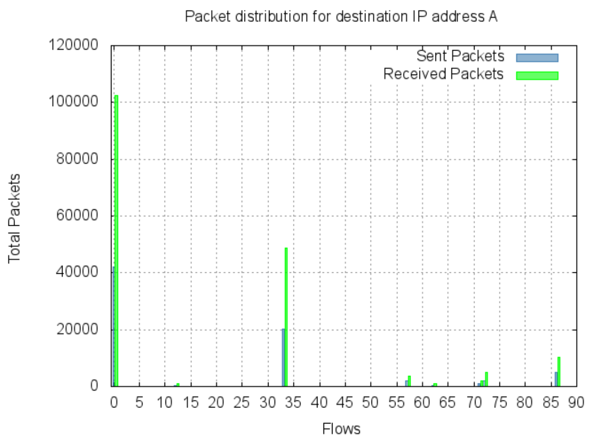
Table 11: Top destination IP addresses for DNS tunnel

Destination IP address	Flows
A	87
B	68
C	17

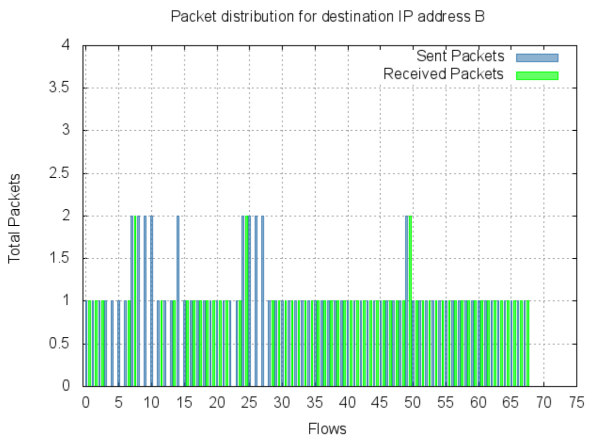
Table 12: Top destination IP addresses for DNS tunnel

Destination IP	# Flows	Avg sent_pkts	Std dev sent_pkts	Avg received_pkts	Std dev received_pkts
A	87	303.6207	877.0426	5037.4138	15680.588
B	68	1.04	0.245	0.997	0.0164
C	17	1.056	0.2789	0.9756	0.1098

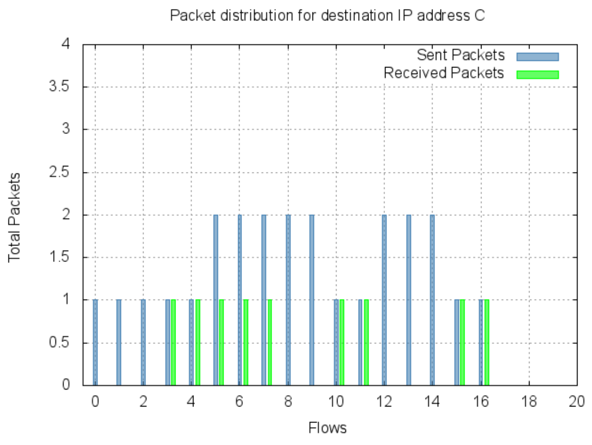
This analysis shows that the destination IP address A where the tunnel server is active has more flows. The packet distribution of all flows, where the DNS tunnel is present, show a very irregular pattern, because there are flows with suspiciously high amounts of sent and received packets. And this is not the case of the other destination IP addresses, see Figure 14, where the packet distribution is similar to the normal behaviour. At this point, irregularity of the packet distribution for the flows generated by the DNS tunnel is visible.



(a) Destination IP of the DNS tunnel server



(b) Destination IP address B



(c) Destination IP address C

Figure 14: Packet distribution

The next analysis focuses on the DNS_QUERY_TYPE field, Table 13 shows this distribution for the DNS tunnel flows. What is interesting to note here is that 13 flows are using a DNS_QUERY_TYPE of value zero, and this is a reserved value, which must never be allocated for ordinary use, according to the IANA specification [15]. This number of flows match the number of tests performed with the tool Iodine, therefore as a hypothesis, every flow that has a value in the DNS_QUERY_TYPE field of zero, can be considered as suspicious.

Table 13: DNS_QUERY_TYPE field analysis for DNS tunnel

DNS_QUERY_TYPE	**# of flows**	**%**
12	60	34.88
10	57	33.14
1	26	15.12
0	13	7.56
16	5	2.92
5	3	1.74
15	3	1.74
33	3	1.74
255	1	0.58
28	1	0.58

The DNS_RET_CODE distribution for the DNS tunnel flows, see Table 14, shows that most of the flows have a successful query response, that is DNS_RET_CODE of 0, 13 flows have a server failure response and 23 flows have a non existent domain response.

Table 14: DNS_RET_CODE field analysis for DNS tunnel

DNS_RET_CODE	**# of flows**
0	136
2	13
3	23

Now, in order to validate the previous hypothesis that states that for every flow that has a DNS_QUERY_TYPE value of 0, the flow will be marked as suspicious. In order to do this, the flow source and destination IP addresses will be obtained, and if the destination IP address corresponds to the DNS tunnel server, then the suspicious flow will correspond to malicious traffic. Also by analyzing the packet ratio and the amount of traffic being transferred, this hypothesis becomes stronger.

Once that this analysis was performed, it was determined that the destination IP addresses which have a value of 0 in the DNS_QUERY_TYPE correspond to the DNS tunnel server, and also, the amount of traffic being transferred is particularly high. The packet ratio of these flows, also show lack of symmetry, because the amount of received packets is almost twice as the amount of bytes sent.

HTTP

In this section, an analysis is made of the behaviour for HTTP within then flow-data set. The template used for this analysis, is shown in Table 15.

Regular HTTP

A total of 40107 flows were analyzed for regular HTTP traffic. This analysis shows that the packet ratio is not 1 for every flow. 40.62% of the flows have a packet ratio of 1, which means that the amount of sent and received packets is the same, 16.12% of the flows have a packet ratio of 0.5, and other values of packet ratio where found. The same analysis for the byte ratio were performed, and the conclusion is that the packet or byte symmetry is not a variable that can be used to detect any suspicious behaviour.

An analysis of the TCP_FLAGS field was performed, see Table 16 for all possible TCP_FLAGS in one flow. This field, in the flow-data set, is represented as the cumulative OR of this value for every packet in one flow, Table 17 shows the number of flows that are using a certain TCP_FLAG value. It shows that most of the flows use the value 24, which have the PUSH and ACK flags set, and they are used at the beginning and at the end of the data transfer to make sure the data segments are handled correctly. Also, the PUSH flag is used to send HTTP or other types of requests through a proxy to ensure that the requests are handled properly. TCP_FLAG of 26 uses the SYN flag to initiate TCP connections. TCP_FLAG of value 27 also use the FIN flag, used to close a TCP connection. For the other flows, the RST flags is also used, and it indicates that the remote host has reset the connection.

Table 15: IPFIX template for HTTP

Field	Description
IPV4_SRC_ADDR	IPv4 source address
IPV4_DST_ADDR	IPv4 destination address
PROTOCOL	IP protocol byte
IN_BYTES	Incoming flow bytes (src->dst)
IN_PKTS	Incoming flow packets (src->dst)
OUT_BYTES	Outgoing flow bytes (dst->src)
OUT_PKTS	Outgoing flow packets (dst->src)
MIN_TTL	Min flow TTL
MAX_TTL	Max flow TTL
TCP_FLAGS	Cumulative of all flow TCP flags
HTTP_URL	HTTP URL
HTTP_METHOD	HTTP METHOD
HTTP_RET_CODE	HTTP return code (e.g. 200, 304...)

Table 16: TCP_FLAGS options

Flag	Description	Binary	Decimal
CWR	Congestion Windows Reduced	10000000	128
ECE	ECN-Echo	01000000	64
URG	Urgent	00100000	32
ACK	Acknowledgment	00010000	16
PSH	Push	00001000	8
RST	Reset	00000100	4
SYN	Syn	00000010	2
FIN	Fin	00000001	1

Other type of analysis is made in the HTTP fields of the IPFIX template. The HTTP_URL field is not interesting, because it does not show any suspicious behaviour when analyzing it, since it only shows the HTTP URL, commonly used for web pages or other resources, such as file transfers (FTP), etc.

The HTTP_METHOD field shows the type of method used in the flow. For this analysis, 52.78% of the flows are using the GET method which is used to retrieve information from the server, 21.87% of the flows are using the HEAD method, which is used to get information about the entity implied by the request without transferring the entity-body itself. 20.49% of the flows do not show the HTTP method that was used, and this can be an issue on the collector. A minority part of the flows, a 4.84% of the flows, are using the POST method, which is used to request a web server to accept the data enclosed in the request message's body for storage, see Table 18. An interesting analysis can be to determine the amount methods that a unique destination IP address is receiving, because this could indicate that something suspicious is occurring, see Table 19. It shows the top 10 destination IP addresses with more flows.

The HTTP_RET_CODE field shows the HTTP response code used in the request-response between the source and destination address. Appendix 3 shows the detailed analysis for all flows. It shows that for most of the flows, there is a successful connection with code 200. For other flows, there is a response 0, which is not documented, but it indicates that the request was empty.

HTTP reverse shell

The analysis of this technique, is performed with a total of 166 flows.

Since a unique server was used to test this setup, there is just one destination IP address. If other servers were used, this analysis would simply have more destination IP addresses.

The analysis of the packets ratio does not show any visible suspicious behaviour as established in the normal behaviour of this protocol. Also, the amount of bytes being transferred is not high enough to raise suspicion.

Table 17: TCP_FLAGS field in analyzed protocols

TCP_FLAG	**# of flows**	**Meaning**	**%**
24	22088	ACK+PUSH	55,0727
26	10284	ACK+PUSH+SYN	25,6414
27	5039	ACK+PUSH+SYN+FIN	12,5639
19	2223	ACK+FIN+SYN	5,5427
17	163	ACK+FIN	0,4064
31	162	ACK+PUSH+RST+SYN+FIN	0,4039
30	93	ACK+PUSH+RST+SYN	0,2319
23	38	ACK+RST+SYN+FIN	0,0947
25	15	ACK+PSH+FIN	0,0374
21	1	ACK+RST+FIN	0,0025
18	1	ACK+SYN	0,0025

Table 18: HTTP_METHOD for the analyzed flows

HTTP_METHOD	**# of flows**	**%**
GET	21167	52,776
HEAD	8773	21,874
-	8217	20,488
POST	1940	4,837
PUT	10	0,025

A research on the distribution of packets per destination IP address contained per flow was performed. This approach did not make much difference from regular HTTP traffic because there are visible peaks of sent and received packets in both normal and suspicious traffic.

The analysis on the TCP_FLAGS field shows that almost 97% of the flows have a TCP_FLAGS value of 27, which indicates that every connection is sending data and closing the connection after the data has been transferred. Almost 2.5% of the flows have a value of 26, and less than 1% of the flows have a value of 31, which indicates that the connection has been reset by the server. Therefore, it is possible to filter out flows with other values than 27 in the TCP_FLAGS field.

The analysis on the HTTP_METHOD field, shows that the 49.36% of the flows are using the GET method, 48.08% of the flows are using the POST method, and 2.6% of the flows do not specify what method is being used. When analyzing how the tool works, when the client or the victim's machine, connects to the server, it requests for a command from the server with a GET method, which is replied by the server by a return code of 200 OK and the content encoded in Base64, then, the answer is later replied by the client with a POST method sending the answer of the command encoded in Base64. When the client does not get any command from the server, it will poll for commands every specific time period by using the GET method. But for every GET method there will be a POST method if the server is sending commands. If the server does not send any command back to the client, the amount of GET methods will be larger than the amount of POST methods. But in a regular malicious behaviour, the attacker will be sending commands to the client every specific time period. Therefore, it might indicate suspicious behaviour if a unique destination IP address has about the same amount of GET and POST methods, therefore a ratio of POST and GET methods should be established in order to classify these types of flows. For regular HTTP traffic, this ratio varies from 0 to 0.4, while for malicious traffic, this ratio is close to 1 (0.974), therefore, the ratio that will be used should be between 0.5 and 1.5 in order to classify a flow as suspicious.

The HTTP_RET_CODE analysis shows that 54.22% of the flows have a 200 code and indicates that the request is successful. 45.78% of the flows have a return code of 0. This analysis does not show any relevant difference between regular HTTP traffic.

3.3 Summary

In this section, a quick summary is shown about some key features that were observed and are relevant for each of the tested techniques.

Table 19: HTTP method analysis for top 10 destination IP addresses

Destination IP address	**# of Flows with method:**			
	GET	**POST**	**HEAD**	**EMPTY**
A	104	-	1722	105
B	114	-	1482	107
C	267	25	849	94
D	-	-	-	979
E	18	-	729	3
F	700	-	-	10
G	628	-	-	33
H	-	-	-	618
I	-	-	555	4
J	371	136	-	39

ICMP

Table 20: ICMP summary

Variable	**Regular ICMP**	**ICMP Tunnel**	**ICMP reverse shell**
Packet ratio	Value very close to 1. It can be 0 when it is a request without a response.	It is never less than 2.	Values are close to 1.
Bytes per packet per flow	Varies depending on how the message was generated. Packet ratio is still maintained to 1.	Incoming packets have a larger value than outgoing packets.	Similar to regular ICMP traffic.
ICMP_TYPE	2048 for every flow.	2048 for every flow.	2048 for every flow.
MIN/MAX TTL	Varies from 28 to 128 for all analized flows.	Varies from 28 to 128 for all analized flows.	High TTL values, from 230 to 255

DNS

Table 21: DNS summary

Variable	**Regular DNS**	**DNS Tunnel**
Packet ratio	Value varies from 0 to 1 for all analyzed flows.	Value varies from 0 to 1 for most of the flows, but there are some visible peaks for specific flows.
Packet distribution per unique destination IP address	Majority of the flows have 1 packet per flow. This value varies from 1 to 4 for all analyzed flows. Maximum standard deviation value of 0.5.	Irregular distribution for the IP address where the DNS tunnel server is running. High standard deviation values.
DNS QUERY TYPE	It does not show any suspicious behaviour by itself.	Reserved or restricted values are being used.
DNS RETURN CODE	It does not show any suspicious behaviour by itself.	It does not show any suspicious behaviour by itself.

HTTP

Table 22: HTTP summary

Variable	**Regular HTTP**	**DNSTunnel**
Packet ratio	Does not show a pattern for every analyzed flow.	Does not a pattern for every analyzed flow.
TCP FLAGS	Most of the flows have a value of 24. Does not show any particular suspicious behaviour.	This value is set to 27 for every analyzed flow.
POST/GET method distribution	It does not show any suspicious behaviour by itself.	About the same amount of POST and GET methods per flow.
HTTP RET CODE	It does not show any suspicious behaviour by itself.	It does not show any suspicious behaviour by itself.

4 Implementation

This section discusses about the implementation of the algorithms to be able to detect malicious traffic, based on the analysis performed in Chapter 3 of this report. Then, an implementation with data provided by the sponsoring company will be made in order test the algorithms and determine if false positives can be detected. Finally, malicious traffic generated by all the techniques previously discussed will be injected, to determine the effectiveness of the proposed algorithms by trying to detect such traffic.

4.1 Proposed algorithms

ICMP tunnel

Every flow that has a value of 1 in the PROTOCOL field must be filtered, this field represents the protocol field in the IPv4 header. After having all flows with ICMP traffic, the packet or byte ratio should not be zero, because that indicates the received packets or bytes are zero, and for the analyzed techniques, this value is never 0.

After having all this filtered flows, a threshold in the packet or byte ratio should be set, which states that the amount of received packets or bytes should not be greater than 1.5 times the amount of packets or bytes sent. This threshold was selected because in the previously analyzed normal behaviour showed that the maximum packets or bytes ratio is never more than one. Also, for malicious behaviour, the minimum packets or bytes rate is never less than 1.

Finally, the source and destination IP addresses should be checked, if these addresses are considered as unknown by the network administrator, after a deeper analysis, which is out of the scope of this research, then the flow should be considered as malicious.

The analysis of the traffic being generated is not considered in this algorithm, since it was found that for normal ICMP traffic, large amounts of traffic were found because of the different tests performed. And we can also note that a network administrator can use the Ping utility for troubleshooting and thus generate large amounts of ICMP traffic, so at this point, if large amounts of traffic are found, variables that can help to detect malicious traffic are the bytes or packets ratio.

See Appendix 4 for the SQL queries used in this algorithm.

ICMP reverse shell

For the ICMP reverse shell technique, every flow with a PROTOCOL value of 1 should be filtered. The minimum TTL value found in the ICMP reverse shell technique was 236, therefore, a threshold of 230 minimum value will be used. Thus, every packet with a minimum TTL value below 230 will be filtered out. After having checked this variable, the amount of bytes being transferred should be checked to be able to validate if the flows are considered to be malicious, and lately, check the destination IP address to determine what are the destinations of these flows.

Appendix 4 shows the SQL queries used in this algorithm. Appendix 5 shows the flow chart of how this algorithm along with the ICMP tunnel detection.

DNS tunnel

In order to obtain every flow with the DNS protocol, the "L4_DST_PORT" field should match the value of 53. After having all DNS flows, the packet ratio should be checked. For this algorithm, the threshold value will be 1.5, therefore, for every flow with a threshold value more than 1.5 will be further analyzed. The destination IP address should be extracted, and the packet distribution of these flows should be checked. If the standard deviation for this distribution exceeds a value of 2, then it will be passed for further analysis, otherwise, the flow will be discarded. The DNS_QUERY_TYPE field should also be checked, since it was found that for some flows this value is equal to 0, which is a reserved value, and it should not be used. Another check on the DNS_RET_CODE should be made in order to determine the result value. Finally, a validation of the destination IP address should identify to which server is the DNS traffic going.

Appendix 4 shows the SQL queries used in this algorithm.

HTTP reverse shell

Every flow with the L4_DST_PORT field should match the value of 80 or 8080. After having every flow with the HTTP protocol, an analysis on the TCP_FLAG field, by filtering every flow with the

TCP_FLAG value of 27. Then, out of these flows, the distinct destination IP addresses should be determined, and then analyze the percentage of GET and POST methods per destination IP address. The ratio should be between 0.5 and 1.5 for the flow to be classified as suspicious. After analyzing this, the HTTP_RET_CODE distribution should be established, in order to determine the amount of successful connections, because that means that for every GET method issued by the client, the server should respond with a 200 code.

Appendix 4 shows the SQL queries used in this algorithm.

4.2 Data-set

After having these algorithms, an implementation of real historic data was performed. This is a data-set provided by the sponsoring company, which consists of 1 day capture of different network protocols and applications. It contains HTTP traffic generated by more than 150 web crawlers, DNS traffic, which is generated along the HTTP requests, it also has ICMP traffic that was generated by the Ping utility, where random commands were issued and for different periods of time. This analysis was performed in order to detect false positives within the data-set and determine the effectiveness of the proposed algorithms by injecting malicious traffic which was previously generated.

The data-set provided by the company consists of 3.05 Gigabytes of network traffic, 7925899 packets and 370172 flows.

ICMP tunnel and ICMP reverse shell detection

For these techniques, the proposed algorithms were tested. The data-set has 12323 ICMP flows. When calculating the packet ratio, and filtering out the values that are zero, the number of valid flows is 5615. After analyzing the packet ratio for every flow, every flow has a packet ratio lower than 1, which shows normal behaviour. The TTL values are lower than 68 and the amount of ICMP traffic is lower than 4 Megabytes, which is also considered as normal behaviour therefore, any false positive was shown during this analysis.

DNS tunneling

The provided data-set has 35186 DNS flows. The packet ratio distribution of these flows is that 35165 flows, that is 99.94% of the flows have a packet ratio of 1, which is considered as normal according to the algorithm, and the other packet ratio values are less than 1.

When analyzing the packet distribution for every destination IP address, it shows that 35089, that is 99.72% of the flows have 1 packet per flow. The minimum packet per flow value is 1 and the maximum packet value is 4, which is considered as normal behaviour.

The DNS_QUERY_TYPE analysis shows that 19515, that is 55.46% of the flows have a DNS RR type of 28 (AAAA), which is used to resolve host names with IPv6 addresses. 15589, that is 44.31% of the flows have a DNS RR type of 1 (A), which is used to resolve hostnames with IPv4 addresses. The rest of DNS query types are 6 (0.02%), 12 (0.04%), 16 (0.17%), but are not considered as malicious traffic.

The DNS_RET_CODE analysis showed that 26431 flows (75.12%) have a return code of 0, which indicates no error, 189 flows (0.54%) have a return code of 2, which indicates a server failure, and further analysis for this flows do not show any suspicious behaviour. And finally, 8566 flows (24.35%) have a return code of 3, which indicates that the query has been refused.

The conclusion of this analysis is that any false positives were found, according to the proposed algorithm, since every flow seems to be normal DNS traffic.

HTTP reverse shell

This data-set has 68988 HTTP flows. When analyzing the TCP_FLAGS field, 56545 (81.93%) flows have a value of 27, other TCP_FLAGS values are not relevant for this analysis since it was determined that only TCP_FLAGS with value 27 were found when testing the HTTP reverse shell technique.

After determining the distinct IP addresses that have the TCP_FLAGS set to 27, 1679 distinct destinations were found. Table 23 shows the top five destination IP addresses for the analyzed flows. And the analysis does not show any suspicious behaviour, because the amount of GET methods are not the same as the POST methods for the distinct destination IP addresses.

Table 23: HTTP_METHOD for analyzed flows

Destination IP address	**# Flows with HTTP_METHOD**		
	POST	**GET**	**EMPTY**
A	0	43203	228
B	0	1176	0
C	2	636	1
D	0	180	0
E	0	99	6

Injecting malicious traffic

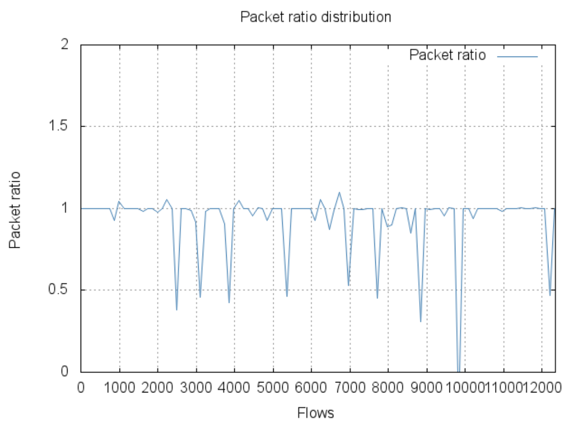
Malicious traffic, which was previously generated will be injected in the provided data-set, in order to detect it and test the effectiveness of the algorithms.

For every technique, three kinds of malicious traffic were injected. For ICMP tunneling, captured network traffic that simulates the download of an 1 Megabyte file, a simple login to a web page and a simulation of a ssh session were used. For the ICMP reverse shell, three sessions that emulate different types of behaviour will be used, one that emulates an idle session, other session with a delay of 30 seconds between commands, and other session that emulates several random commands. For DNS tunneling, three network captures that simulates the download of 1 Megabyte file, a ssh session, and access to a database will be used. And finally, for HTTP reverse shell, three network captures that simulate normal, aggressive and ids-evation behaviour will be used.

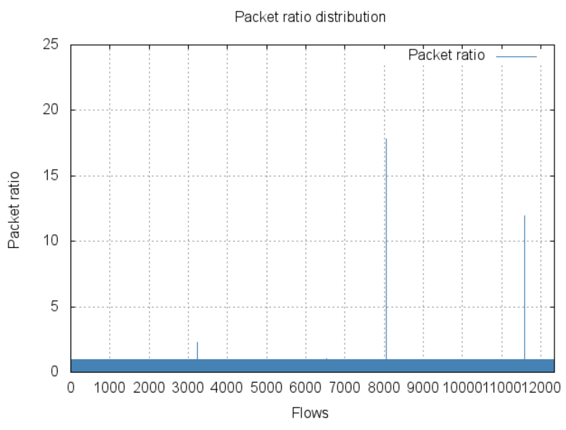
5 Results

When performing analysis on the data set provided by the company, any false positives were found that may indicate malicious activity generated by the discussed techniques. All flows analyzed showed normal behavior, however, to verify that injecting malicious traffic, this behavior can be affected.

After injecting this traffic into the data-set, the total amount of ICMP flows is 12352. When querying for every flow that has a packet ratio higher than 2, which is the threshold set in the algorithm, the output gives three flows, which are the previously injected flows, see Figure 15a. For ICMP before the malicious traffic was injected, the packet ratio for every flow is close to one, but when the malicious traffic has been injected, there are tree visible peaks, see Figure 15b.



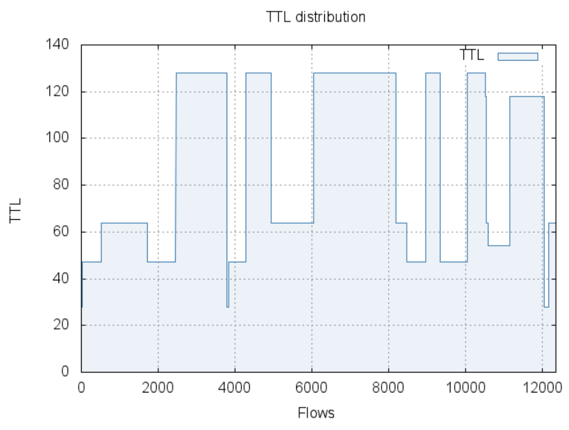
(a) normal ICMP traffic



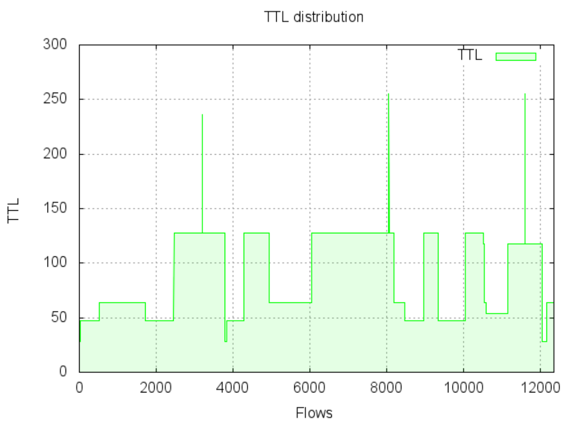
(b) ICMP tunnel traffic

Figure 15: Packet ratio distribution for ICMP traffic

When querying for every flow with a TTL higher than 230, then the output gives also three flows which correspond to the injected network traffic, see Figure 17a, Figure 17b. By analyzing the incoming bytes being transferred between the two endpoints, we can determine that suspicious activity is taking place.



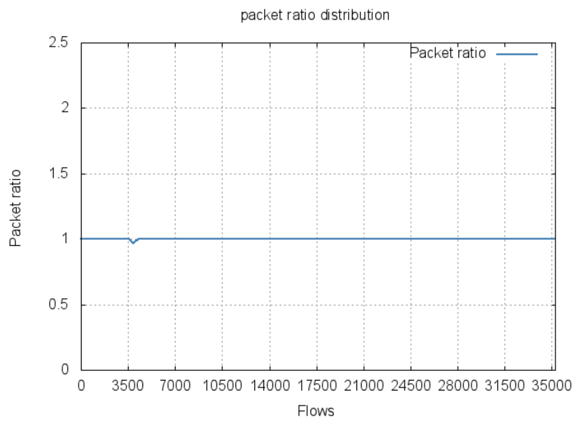
(a) normal ICMP traffic



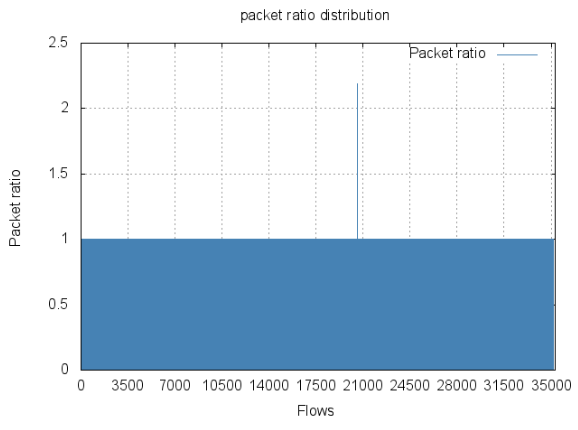
(b) ICMP reverse shell

Figure 16: TTL distribution for ICMP traffic

There is a total of 35219 DNS flows. Before injecting malicious traffic, the packet distribution shows that 99.94% of the flows have a packet ratio of 1, and the rest of flows is below 1, see Figure 17a. When implementing the algorithm to find DNS tunneling traffic, it only returns one flow that has a packet ratio more than 1.5, see Figure 17b. Then, an analysis of the packet distribution for the destination IP address of this flow was made, the standard deviation value for the incoming packets is 91.42, which marks this flow as suspicious. And the amount of sent and received bytes is particularly high (45.19 sent and 1124.38 received Megabytes).



(a) normal DNS traffic



(b) DNS tunnel

Figure 17: Packet distribution for DNS traffic

When analyzing the DNS_QUERY_TYPE field, it is possible to detect all three malicious flows, because the 0 value for this field is reserved, and for this technique, this value is used, therefore it becomes easier to detect this kind of suspicious flows.

For HTTP, there is a total of 64095 HTTP flows. When implementing the algorithm, after filtering out every flow with the TCP FLAG set to 27, the different destination IP addresses were analyzed. It becomes very difficult to find the destination IP that is using this technique because the amount of traffic being generated per flow is not high. Once analyzing every destination IP, the amount of GET and POST methods per IP were determined. This analysis shows that for the malicious flows, 34 flows use the POST methods and 69 flows use the GET method, which is a POST/GET ratio of 0.49. Figure 18 shows the top 20 destination IP addresses with the TCP_FLAGS field set to 27, in which the amount of POST and GET methods is shown. For the IP address E, it shows that the number of GET and POST methods are relatively close, therefore this destination IP address can be marked as suspicious, and in fact, this analysis showed that this is the IP address where the reverse shell client is communicating to. Even though, three types of behaviour were injected, the algorithm only shows one connection, and this is because this analysis is based on destination IP addresses, and for this technique, only one server that acts as the reverse shell server was used. The amount of flows with HTTP return code of 200 OK is 70, which are the successful connections. This analysis demonstrates that the threshold for the amount of GET and POST methods can affect on the false positives rate, because a ratio of POST over GET methods per destination IP address was set between 0.5 and 1.5, therefore, this threshold can be stated to a lower ratio.

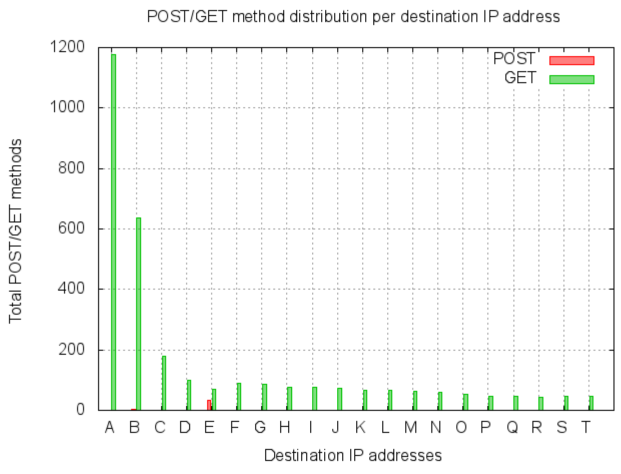


Figure 18: POST/GET methods distribution

6 Conclusions

This research has investigated how normal ICMP, DNS and HTTP traffic behaves based on analysis performed in a historic flow data-set, to further detect covert channel techniques that use these protocols to send information which is undetectable to security devices such as firewalls, IDS / IPS, etc. This analysis shows some specific patterns for each protocol, and these patterns can be used to establish a baseline to evaluate different types of behavior. Furthermore, by analyzing the behavior of these protocols, when they are misused by tunneling techniques, it is possible to differentiate between normal behavior and malicious behavior. It is important to note that this analysis was performed by analyzing specific tools, that may behave in a different way from other tools. The chosen techniques were implemented with well known tools that are commonly used, and that are not difficult to implement.

When analyzing ICMP traffic, normal behavior patterns are clearly established by using variables such as the packet ratio, which is the difference between the number of received packets by the number of sent packets. This variable is practically constant for every case of normal use of the ICMP protocol, while when used to transport encapsulated information in the data field of the ICMP, the packet ratio is very effective in detecting these behaviors. In the case of the ICMP reverse shell, the packet ratio shows no difference from the regular traffic, and thus, this variable cannot be used. During the stages of experimentation and analysis, it was found that the tool with which the experiments were performed, is using a suspiciously high TTL value compared with the normal values for other flows, therefore, this variable was used to detect such malicious traffic. Although this variable can not show any irregular activity, it is possible to mark these flows as suspicious, and later analyze them more closely and look at other variables such as the number of bytes sent and received, where indeed such flows contain malicious traffic.

The analysis of the DNS protocol shows that there are visible patterns of behavior for most of the analyzed flows. The packet ratio shows that for most flows, this is a constant value, while for malicious traffic, this variable does not show a constant value. Once flows are marked as suspicious, a classification of the different destination IP addresses, to where the DNS requests are made, and an analysis of the packet distribution is conducted. The standard deviation is used to determine how close the values of the packets are to each other are. This analysis shows that for regular traffic, the value of the standard deviation is less than 1, whereby a threshold of 2 was established to classify a flow as suspect. When working with NetFlow version 10 (IPFIX), one can use some DNS protocol related fields. An analysis in the DNS_QUERY_TYPE field, shows that for regular DNS traffic, these values are in the normal range because values are clearly established by the standard, while for malicious traffic generated by the DNS tunneling technique, it was found that the flows that carry hidden information are using prohibited or reserved values, which under normal circumstances should not be used.

For HTTP traffic, the analysis showed that the variable packets ratio cannot be used because there is no visible pattern for regular traffic and malicious traffic behavior. During testing and analysis, it was possible to detect that all flows generated by this technique, have the TCP_FLAGS field set to 27. This value is the sum of all flags for each flow, and indicates that each flow starts, sends and closes a communication. Further analysis was performed to identify the different destination IP addresses and analyze the POST/GET ratio, that is the difference between POST and GET methods per unique destination IP address. The analysis of normal HTTP traffic, showed that this rate is between the values of 0 and 0.4, while for malicious traffic, this value is close to 1, but for the algorithm, a threshold between 0.5 and 1.5 was established. This algorithm can produce false positives, because it is possible to find short duration flows, with the same amount of POST or GET methods. But this check could not be performed due to time constraints, but identifying these suspicious flows may allow to identify the destination IP address in order to maintain such flows constantly monitored. HTTP traffic can also be proxied through other known ports such as 8080, therefore, this option should also be noticed when filtering HTTP traffic.

Once the differences between regular and malicious traffic were identified, certain parameters and steps as algorithms, in order to detect such suspicious flows, were established. To verify the effectiveness of the operation of these algorithms, a dataset provided by the sponsoring company was analyzed. This data set consisted of a capture of network traffic for a day, generated by 150 web crawlers that simulate human behavior while browsing the Internet. Furthermore, this data set contains DNS traffic and ICMP traffic generated randomly through different duration ping commands. The analysis of this dataset produced no false positive, ie, any malicious traffic related to any of the previously mentioned techniques was found. Subsequently, malicious traffic was injected in this dataset, in order to verify whether the proposed algorithms can detect this traffic. It was determined that for the traffic generated by the ICMP tunnel technique, all kinds of injected traffic were detected. Similarly, the injected traffic generated by the DNS tunneling and the HTTP reverse shell technique was detected, by performing the analysis of proposed algorithms.

Finally, the analysis of flow-data may have some disadvantages compared to other types of analysis such as IDS/IPS or firewalls, since it does not look into the content of the packets. But, it is still a powerful tool to analyze anomalies based on the behaviour of specific flows. Analysis based on the packet content inspection, may also be time and resource consuming, therefore flow-data analysis should be considered as an alternative to security analysis.

6.1 Future work

This research has extensively investigated the behavior of several tunnelling techniques or covert channels. However, only specific tools were tested, therefore, it is important to investigate the behavior of such protocols with other tools, to further assess the effectiveness of the proposed algorithms regardless of the tool that was used. Moreover, the techniques of covert channels vary depending on the creativity of those who create them, therefore, it is difficult to establish a unique approach to identify each type of malicious traffic generated by these techniques. There are many other techniques and protocols that are used to transmit information in a hidden way, so it is worth looking at other protocols, tools and techniques to generate covert channels.

The proposed algorithms were tested with historical data in order to find patterns in regular and malicious traffic. The same analysis should be performed with live flow-data, to compare the results of the behavior of these flows. Also, to compare the effectiveness of the algorithms in such an environment.

For this project, database queries were used to obtain the results presented in this research. When working with large databases, these queries can take a long time to run, so one should find out how to implement the proposed algorithms by way of scripting or programming languages. Such improvements could help the analysis to be much more effective in terms of time.

In order to test whether the proposed algorithms generate false positives, several tests should be performed with bigger datasets and longer period of capture. Also, analyzing other techniques than covert channels that may use the protocols analyzed to determine if they produce the same results, or have a similar behavior as the analyzed techniques.

This analysis was performed without sampling features that allow flow-data to be sampled in a particular way. This option is widely used in high speed networks that do not collect every single packet because of performance issues and only analyze sampled packets. The observed behaviour should not differ from sampled data because there are specific patterns that are common for every packet in one flow, but it should be relevant to investigate these techniques with sampled data in order to compare the differences.

Bibliography

[1] Zhongjun Lu Ping Dong, Huanyan Qian and Shaohua Lan. A network covert channel based on packet classification. *International Journal of Network Security.*

[2] Taeshik Sohn, JungTaek Seo, and Jongsub Moon. A study on the covert channel detection of tcp/ip header using support vector machine. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, *Information and Communications Security*, volume 2836 of *Lecture Notes in Computer Science*, pages 313–324. Springer Berlin Heidelberg, 2003.

[3] Vincent Berk, Annarita Giani, George Cybenko, et al. Detection of covert channel encoding in network packet delays. *Rapport technique TR536, de lUniversité de Dartmouth. Novembre*, 2005.

[4] Anna Sperotto Harm Schotanus Michel Mandjes Erik Meeuwissen Wendy Ellens, Piotr Zuraniewski. Flow-Based Detection of DNS Tunnels. 2013.

[5] Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of IP Traffic Flow Information. `http://tools.ietf.org/html/rfc5101`.

[6] Bidirectional Flow Export Using IP Flow Information Export (IPFIX). `http://tools.ietf.org/html/rfc5103`.

[7] Petr Velan. Practical experience with IPFIX flow collectors. 2013.

[8] nprobe. `http://www.ntop.org/products/nprobe/`.

[9] Tcpreplay. `http://tcpreplay.synfin.net/`.

[10] Ping tunnel. `http://www.cs.uit.no/~daniels/PingTunnel/`.

[11] Simple reverse ICMP shell. `https://github.com/inquisb/icmpsh`.

[12] Iodine. `http://code.kryo.se/iodine/`.

[13] MATAHARI, a simple reverse HTTP shell. `http://matahari.sourceforge.net/`.

[14] RFC 792 - Internet Control Message Protocol. `http://tools.ietf.org/html/rfc792`.

[15] Domain Name System (DNS) Parameters. `http://www.iana.org/assignments/dns-parameters/dns-parameters.xhtml`.

1 ICMP tunnel and reverse shell

This appendix shows some examples of how information can be hidden inside a protocol header. Below is a capture of a regular ICMP message.

```
Offset     Hexdump                                     ASCII dump

0x0000:    c8be 198a 066e 6036 dda8 258c 0800 4500     .....n'6..%...E.
0x0010:    003c 4fe3 0000 8001 3041 c0a8 0064 9164     .<O.....0A...d.d
0x0020:    682c 0800 3862 0100 13fa 6162 6364 6566     h,..8b....abcdef
0x0030:    6768 696a 6b6c 6d6e 6f70 7172 7374 7576     ghijklmnopqrstuv
0x0040:    7761 6263 6465 6667 6869                    wabcdefghi
```

Figure 19: Echo or Reply message

Below is a capture where a HTTP GET method is being carried by the ICMP packet.

```
Offset     Hexdump                                     ASCII dump

0x0000:    0016 3e11 c1fa feff ffff ffff 0800 4500     ..>..........E.
0x0010:    0164 0000 4000 3401 969d d57c deb5 9164     .d..@.4....|...d
0x0020:    6965 0800 4cae ab6a 0001 d520 0880 0000     ie..L..j........
0x0030:    0000 0000 0000 4000 0002 0000 ffff 0000     ......@.........
0x0040:    012b 0001 ab6a 4745 5420 2f20 4854 5450     .+...jGET./.HTTP
0x0050:    2f31 2e31 0d0a 486f 7374 3a20 6c6f 6361     /1.1..Host:.loca
0x0060:    6c68 6f73 743a 3830 3830 0d0a 5573 6572     lhost:8080..User
0x0070:    2d41 6765 6e74 3a20 4d6f 7a69 6c6c 612f     -Agent:.Mozilla/
0x0080:    352e 3020 2858 3131 3b20 4c69 6e75 7820     5.0.(X11;.Linux.
0x0090:    7838 365f 3634 3b20 7276 3a32 322e 3029     x86_64;.rv:22.0)
0x00a0:    2047 6563 6b6f 2f32 3031 3030 3130 3120     .Gecko/20100101.
0x00b0:    4669 7265 666f 782f 3232 2e30 2049 6365     Firefox/22.0.Ice
0x00c0:    7765 6173 656c 2f32 322e 300d 0a41 6363     weasel/22.0..Acc
0x00d0:    6570 743a 2074 6578 742f 6874 6d6c 2c61     ept:.text/html,a
0x00e0:    7070 6c69 6361 7469 6f6e 2f78 6874 6d6c     pplication/xhtml
0x00f0:    2b78 6d6c 2c61 7070 6c69 6361 7469 6f6e     +xml,application
0x0100:    2f78 6d6c 3b71 3d30 2e39 2c2a 2f2a 3b71     /xml;q=0.9,*/*;q
0x0110:    3d30 2e38 0d0a 4163 6365 7074 2d4c 616e     =0.8..Accept-Lan
0x0120:    6775 6167 653a 2065 6e2d 5553 2c65 6e3b     guage:.en-US,en;
0x0130:    713d 302e 350d 0a41 6363 6570 742d 456e     q=0.5..Accept-En
0x0140:    636f 6469 6e67 3a20 677a 6970 2c20 6465     coding:.gzip,.de
0x0150:    666c 6174 650d 0a43 6f6e 6e65 6374 696f     flate..Connectio
0x0160:    6e3a 206b 6565 702d 616c 6976 650d 0a0d     n:.keep-alive...
0x0170:    0a00
```

Figure 20: ICMP Echo request message carrying HTTP

Below is a capture where shell commands are being transported in a ICMP message.

```
Offset     Hexdump                                     ASCII dump
0x0000:    0016 3eee 780e feff ffff ffff 0800 4500     ..>.x.........E.
0x0010:    005c 03ae 0000 f501 0cb5 57d5 629e 9164     .\........W.b..d
0x0020:    6966 0800 dc40 dcce ce68 4d69 6372 6f73     if...@...hMicros
0x0030:    6f66 7420 5769 6e64 6f77 7320 5b56 6572     oft.Windows.[Ver
0x0040:    7369 a26e 2036 2e33 2e39 3630 305d 0d0a     si.n.6.3.9600]..
0x0050:    2863 2920 3230 3133 204d 6963 726f 736f     (c).2013.Microso
0x0060:    6674 2043 6f72 706f 7261                    ft.Corpora
```

Figure 21: ICMP Echo request message carrying shell commands

2 HTTP reverse shell

The information is encoded in Base64, and represents the `df -h` command, that requests for the available space from the victim's machine. Once the client gets this response message from the server, it will answer back with a POST method, giving the information requested.

```
GET / HTTP/1.1
 Host: 145.100.105.102
 Accept-Encoding: identity
 Content-Salt: 141-1402260694.1
 Next-Polling-In: 60

HTTP/1.0 200 OK
 Server: BaseHTTP/0.3 Python/2.6.6
 Date: Sun, 08 Jun 2014 20:52:41 GMT

ZGYgLWg=


POST / HTTP/1.1
 Host: 145.100.105.102
 Accept-Encoding: identity
 Content-Length: 496
 Next-Polling-In: 60

RmlsZXN5c3RlbSAgICAgIFNpemUgIFVzZWQgQXZhaWwgVXNlJSBNb3VudGVkIG9uCi9kZXYvc2RhMSAgICA
gICAgMTlHICA3LDZHICAgMTFHICA0NCUgLwpub25lICAgICAgICAgICAgNCwwSyAgICAgMCAgNCwwSyAgID
AlIC9zeXMvZnMvY2dyb3VwCnVkZXYgICAgICAgICAgICAyLDBHICA0LDBLICAyLDBHICAgMSUgL2Rldgp0b
XBmcyAgICAgICAgICAgMzk1TSAgMSw0TSAgMzk0TSAgIDElIC9ydW4Kbm9uZSAgICAgICAgICAgIDUsME0g
ICAgIDAgIDUsME0gICAwJSAvcnVuL2xvY2sKbm9uZSAgICAgICAgICAgIDIsMEcgIDE1MksgIDIsMEcgICA
xJSAvcnVuL3NobQpub25lICAgICAgICAgICAgMTAwTSAgIDM2SyAgMTAwTSAgIDElIC9ydW4vdXNlcgo=
```

Where the information encoded represents the following output:

```
Filesystem        Size  Used Avail Use% Mounted on
/dev/sda1          19G  7,6G   11G  44% /
none              4,0K     0  4,0K   0% /sys/fs/cgroup
udev              2,0G  4,0K  2,0G   1% /dev
tmpfs             395M  1,4M  394M   1% /run
none              5,0M     0  5,0M   0% /run/lock
none              2,0G  152K  2,0G   1% /run/shm
none              100M   36K  100M   1% /run/user
```

3 HTTP_METHOD

This appendix shows the details of the HTTP_METHOD for the analyzed flows.

Table 24: HTTP_METHOD analysis

HTTP_RET_CODE	**Meaning**	**# of flows**	**%**
200	OK	20932	52,1904%
0	Not documented	9111	22,7167%
302	Found	6910	17,2289%
206	Partial Content	1105	2,7551%
304	Not Modified	1070	2,6679%
301	Moved Permanently	485	1,2093%
404	Not Found	145	0,3615%
403	Forbidden	142	0,3541%
204	No Content	61	0,1521%
400	Bad Request	47	0,1172%
201	Created	45	0,1122%
101	Switching Protocols	27	0,0673%
303	See Other	7	0,0175%
504	Gateway Timeout	7	0,0175%
503	Service Unavailable	5	0,0125%
502	Bad Gateway	5	0,0125%
500	Internal Server Error	1	0,0025%
1	Not documented	1	0,0025%
100	Continue	1	0,0025%

4 SQL queries for the proposed algorithms

This appendix shows some of the SQL queries used to implement the algorithms for the different tunneling techniques discussed in this research.

4.1 ICMP tunnel

First, a view is created for every flow in order to create temporal columns such as the packets and bytes rate, so it becomes easier to do calculations based on these fields. Then, the protocols are filtered with the value 1, which is for ICMP. After having this view, it is possible to create another view for all previous view to be able to have all flows with temporal columns in one single table.

```
CREATE VIEW ptunnel_view AS
    SELECT
        INET_NTOA(ipv4_src_addr) AS src_addr,
        INET_NTOA(ipv4_dst_addr) AS dst_addr,
        in_bytes,
        out_bytes,
        (out_bytes / in_bytes) AS bytes_ratio,
        in_pkts,
        out_pkts,
        (out_pkts / in_pkts) AS pkts_ratio,
        (in_bytes / in_pkts) AS in_bpp,
        (out_bytes / out_pkts) AS out_bpp,
        min_ttl,
        max_ttl,
        icmp_type
    FROM
        ptunnel_flow
    WHERE
        protocol LIKE 1;
```

After having a view, we can select the flows that have the packets rate higher than the threshold specified and also checking that this value is not 0.

```
SELECT
    *
FROM
    ptunnel_view
WHERE
    pkts_ratio > 2 AND out_pkts != 0;
```

4.2 ICMP reverse shell

```
CREATE VIEW rev_icmp_view AS
    SELECT
        INET_NTOA(ipv4_src_addr) AS SRC_ADDR,
        INET_NTOA(ipv4_dst_addr) AS DST_ADDR,
        in_bytes,
        out_bytes,
        (out_bytes / in_bytes) AS bytes_ratio,
        in_pkts,
        out_pkts,
        (out_pkts / in_pkts) AS pkts_ratio,
        (in_bytes / in_pkts) AS in_bpp,
        (out_bytes / out_pkts) AS out_bpp,
        min_ttl,
        max_ttl
    FROM
        rev_icmp_flow
    WHERE
        protocol = 1;
```

To filter out every flow that has a minimum TTL value less than 230:

```
SELECT
    *
FROM
    rev_icmp
WHERE
    protocol = 1
        AND (min_ttl > 245 OR max_ttl > 245);
```

4.3 DNS tunneling

```
CREATE VIEW dns_tunnel_view AS
    SELECT
        INET_NTOA(ipv4_src_addr) AS SRC_ADDR,
        INET_NTOA(ipv4_dst_addr) AS DST_ADDR,
        in_bytes,
        out_bytes,
        (out_bytes / in_bytes) AS bytes_ratio,
        in_pkts,
        out_pkts,
        (out_pkts / in_pkts) AS pkts_ratio,
        (out_bytes / in_bytes) AS in_bpp,
        (out_bytes / out_pkts) AS out_bpp,
        min_ttl,
        mx_ttl,
        dns_query,
        dns_query_id,
        dns_query_type,
        dns_ret_code
    FROM
        dns_tunnel_flow
    WHERE
        l4_dst_port = 53;

SELECT
    dst_addr, COUNT(*) AS counter
FROM
    dns
GROUP BY dst_addr
ORDER BY counter DESC;

SELECT
    MIN(in_pkts),
    MAX(in_pkts),
    AVG(in_pkts),
    STDDEV(in_pkts),
    MIN(out_pkts),
    MAX(out_pkts),
    AVG(out_pkts),
    STDDEV(out_pkts)
FROM
    dns;
```

4.4 HTTP reverse shell

Creating view to filter every HTTP connetion.

```
CREATE VIEW rev_http_view AS
    SELECT
        INET_NTOA(ipv4_src_addr) AS src_addr,
        INET_NTOA(ipv4_dst_addr) AS dst_addr,
        in_bytes,
        out_bytes,
        (out_bytes / in_bytes) AS bytes_ratio,
        in_pkts,
        out_pkts,
        (out_pkts / in_pkts) AS pkts_ratio,
        (in_bytes / in_pkts) AS in_bpp,
        (out_bytes / out_pkts) AS out_bpp,
        min_ttl,
        max_ttl,
        tcp_flags,
        http_url,
        http_method,
        http_ret_code
    FROM
        rev_http_flow
    WHERE
        l4_dst_port = 80
    OR
        l4_dst_port = 8080;
```

Select the distinct IP addresses.

```
SELECT
    src_addr, dst_addr, COUNT(dst_addr) AS contar
FROM
    normal.rev_http_view
WHERE
    tcp_flags = 27
GROUP BY dst_addr
ORDER BY contar DESC;
```

5 Algorithms flow diagram

This appendix, shows the flow diagrams for every detection algorithm.

5.1 ICMP-based tunneling techniques

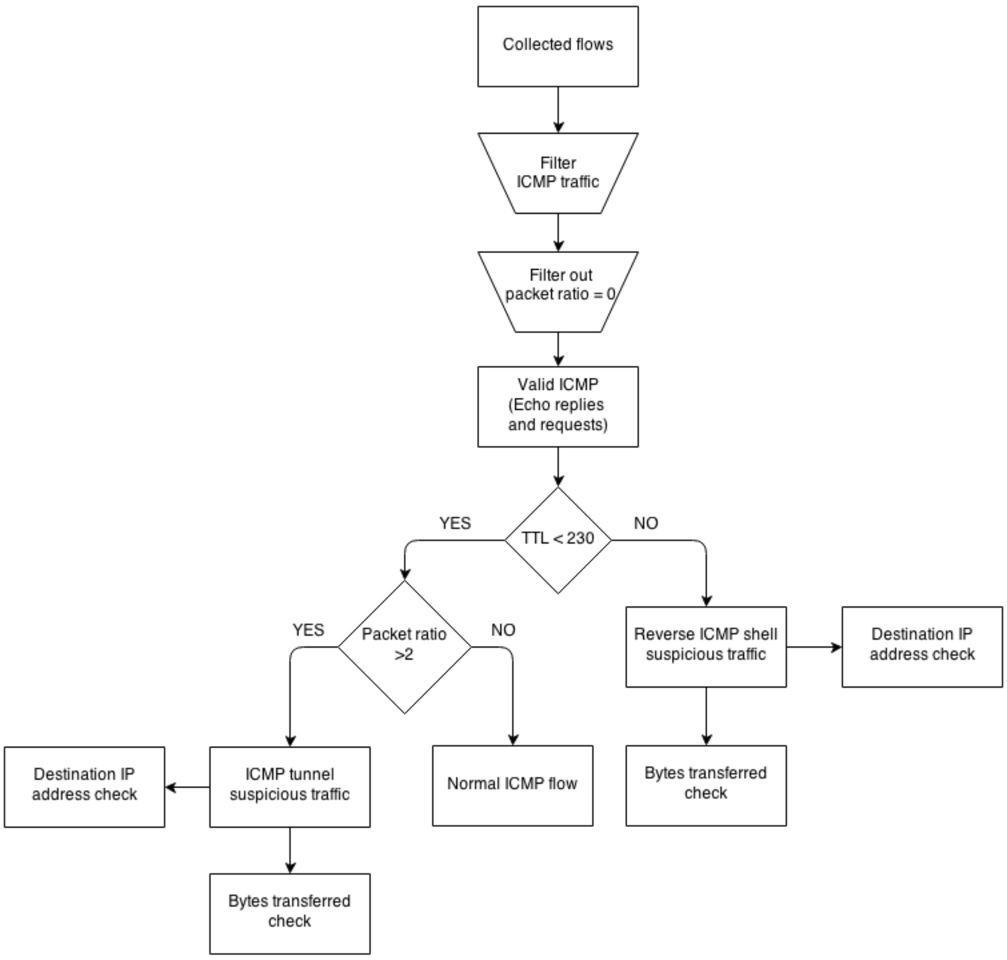

Collected flows
Filter ICMP traffic
Filter out packet ratio = 0
Valid ICMP (Echo replies and requests)
TTL < 230
YES
NO
Packet ratio >2
YES
NO
Reverse ICMP shell suspicious traffic
Destination IP address check
Destination IP address check
ICMP tunnel suspicious traffic
Normal ICMP flow
Bytes transferred check
Bytes transferred check

Figure 22: ICMP-based tunneling techniques

5.2 DNS-based tunneling techniques

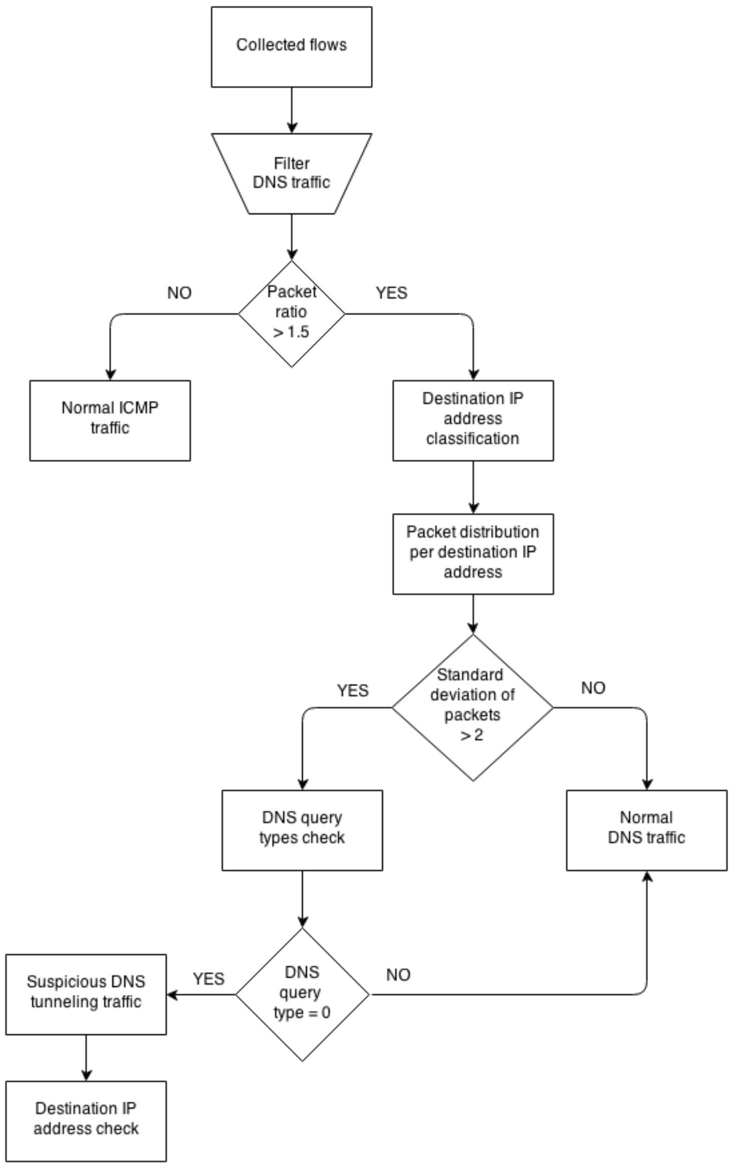
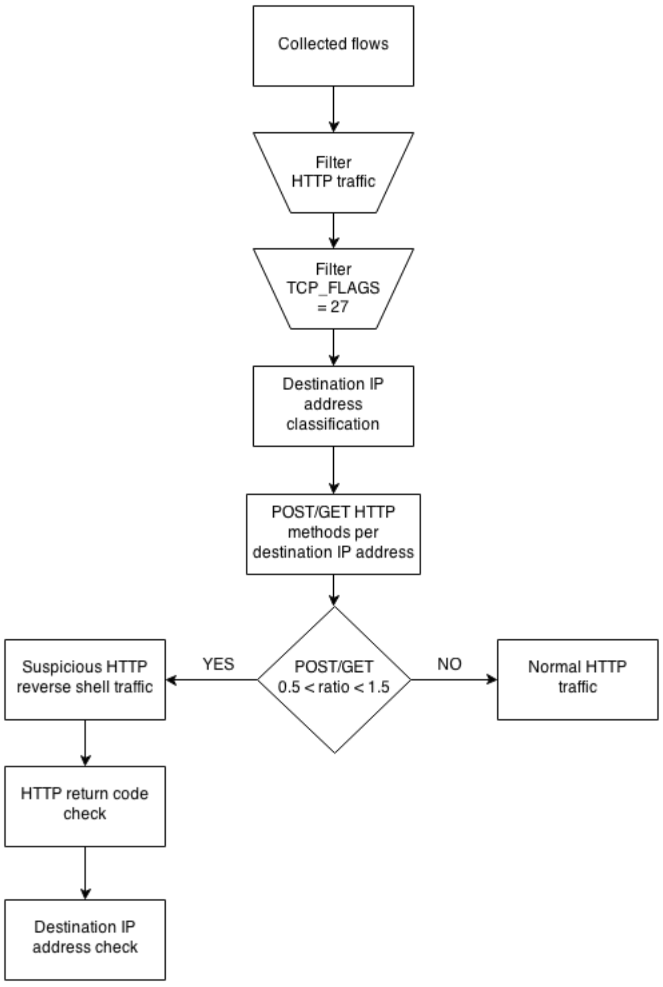

Collected flows
Filter DNS traffic
Packet ratio > 1.5
NO
YES
Normal ICMP traffic
Destination IP address classification
Packet distribution per destination IP address
Standard deviation of packets > 2
YES
NO
DNS query types check
Normal DNS traffic
DNS query type = 0
YES
NO
Suspicious DNS tunneling traffic
Destination IP address check

Figure 23: DNS-based tunneling techniques

5.3 HTTP-based tunneling techniques



Collected flows
Filter HTTP traffic
Filter TCP_FLAGS = 27
Destination IP address classification
POST/GET HTTP methods per destination IP address
POST/GET 0.5 < ratio < 1.5
YES
NO
Suspicious HTTP reverse shell traffic
Normal HTTP traffic
HTTP return code check
Destination IP address check

Figure 24: HTTP-based tunneling techniques