APPLYING COVERT CHANNEL IN TCP FAST OPEN (TFO)

MOHAMED AZRAN BIN AZIZ

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

APPLYING COVERT CHANNEL IN TCP FAST OPEN (TFO)

MOHAMED AZRAN BIN AZIZ

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA
KUALA LUMPUR

2019

UNIVERSITY OF MALAYA

ORIGINAL LITERARY WORK DECLARATION

Name of Candidate: Mohamed Azran bin Aziz

Registration/Matric No: WGA140039

Name of Degree: Master of Computer Science

Title of Project Paper/Research Report/Dissertation/Thesis ("this Work"):

Applying Covert Channel in TCP Fast Open (TFO)

Field of Study: Information Security

I do solemnly and sincerely declare that:

(1) I am the sole author/writer of this Work;
(2) This Work is original;
(3) Any use of any work in which copyright exists was done by way of fair dealing and for permitted purposes and any excerpt or extract from, or reference to or reproduction of any copyright work has been disclosed expressly and sufficiently and the title of the Work and its authorship have been acknowledged in this Work;
(4) I do not have any actual knowledge nor do I ought reasonably to know that the making of this work constitutes an infringement of any copyright work;
(5) I hereby assign all and every rights in the copyright to this Work to the University of Malaya ("UM"), who henceforth shall be owner of the copyright in this Work and that any reproduction or use in any form or by any means whatsoever is prohibited without the written consent of UM having been first had and obtained;
(6) I am fully aware that if in the course of making this Work I have infringed any copyright whether intentionally or otherwise, I may be subject to legal action or any other action as may be determined by UM.

Candidate's Signature Date:

Subscribed and solemnly declared before,

Witness's Signature Date:

Name:

Designation:

APPLYING COVERT CHANNEL IN TCP FAST OPEN (TFO)

ABSTRACT

Covert channel is one of the techniques that is used in information hiding. It uses communication channel as a medium for transmitting hidden information. There are two main categories in covert channel namely storage covert channel and timing covert channel. Storage covert channel basically manipulate existing data and/or encode hidden messages within legitimate data. Whereas, timing covert channel intentionally manipulate timing behaviour of resources e.g. delaying between packets to create codes. There are many implementations of covert channel in TCP that use various fields in the TCP header such as Sequence Number, Urgent Pointer and reserved fields. Techniques such as field replacement, create intended delays and manipulating random values are used in implementing covert channel in TCP. Moreover, covert channel implementations also extended to optional fields such as Maximum Segment Size (MSS) and Timestamps. From time to time these optional fields (TCP Options) get evolved (e.g. Quick-Start Response - 2007, TCP Authentication Option – 2010 and TCP Fast Open -2014) and thus more potential covert channel implementations can be discovered. TCP Fast Open (TFO) is one of the latest TCP options that offers faster transmission performances between nodes. It utilises up to 16 bytes in allocated options field in TCP header as its message authentication code (MAC). Previous covert channel implementations cover various fields in the TCP header but not TFO. The aim of this study is to introduce covert channel in TFO by manipulating allocated options field in the TCP header known as TFO cookie. Subsequent to this, observation on performances are investigated as to detect any changes in semantic as well as syntax of TFO transactions. To conduct this study, tools are built to manipulate incoming and outgoing packet transactions and create covert content in allocated options field in TCP header. Further, performance test is conducted to observe any changes in transactions between implemented covert channel TFO and ordinary TFO. The results of the tests show covert content is transferred successfully between receiver and sender without breaking TFO transaction. Moreover, the results also show there are no significance performance degradation when applying covert channel into TFO. These results indicate that covert channel can be created in TFO and works normally as ordinary TFO. On this basis, it would make covert channel in TFO as one of latest alternative methods in implementation of covert channel in TCP.

Keywords: covert channel, TCP Fast Open, network steganography, network security, information hiding

PENERAPAN SALURAN TERSELINDUNG DIDALAM TCP *FAST OPEN* (TFO)

ABSTRAK

Saluran terselindung adalah salah satu cara yang digunakan dalam penyembunyian maklumat. Ia menggunakan saluran komunikasi sebagai medium untk menghantar maklumat tersembunyi. Terdapat dua kategori utama di dalam saluran terselindung iaitu saluran terselindung storan dan saluran terselindung bermasa. Saluran terselindung storan biasanya memanipulasikan data yang sedia ada dan/atau mengedkod masej tersembunyi di sebalik data yang sah. Manakala saluran terselindung bermasa secara sengaja memanipulasikan perilaku masa sesuatu sumber sebagai contoh membuat lengahan masa tertentu untuk penjanaan sesuatu kod. Terdapat banyak kaedah membuat saluran terselindung di dalam TCP yang menggunakan pelbagai medan di dalam kepala TCP seperti *Sequence Number*, *Urgent Pointer* dan lain-lain. Kaedah seperti penggantian medan, pembinaan lengahan sengaja, manipulasi nilai rawak dan lain-lain digunakan didalam pelaksanaan saluran terselindung di dalam TCP. Selain itu, saluran terselindung juga meliputi beberapa opsyen TCP seperti di *Maximum segment size* (MSS) dan *Timestamps*. Dari semasa ke semasa, opsyen TCP berkembang (contohnya *Quick-Start Response* - 2007, TCP *Authentication Option* – 2010 and TCP *Fast Open* -2014) dan oleh itu terdapat lebih banyak potensi dalam membina saluran terselindung untuk diterokai. TCP *Fast Open* (TFO) adalah salah satu opsyen baru didalam TCP yang memberi prestasi pantas di dalam transmisi paket. Ia menggunakan sehingga 16 *bytes* di dalam medan opsyen diperuntukan dalam kepala TCP sebagai *message authentication code* (MAC). Implemntasi saluran terselindung mencakupi pelbagai medan tetapi tidak di dalam TFO. Tujuan utama kajian ini adalah untuk memperkenalkan saluran tersembunyi ke atas TFO dengan cara memanipulasi medan opsyen yang dikenali sebagai kuki TFO. Lanjutan itu, pencerapan ke atas prestasi juga dikaji untuk melihat sebarang perubahan dari segi semantik mahupun sintaks transaksi TFO. Bagi mengendali kajian ini, satu alatan dibina bagi membuat manipulasi transaksi keluar masuk paket serta juga membina maklumat terselindung di dalam medan opsyen di dalam kepala TCP. Selanjutnya, ujian prestasi dilakukan bagi mencerap sebarang perubahan di dalam transaksi antara TFO saluran terselindung dan TFO biasa. Hasil daripada ujian mendapati maklumat terselindung berjaya dihantar di antara penghantar dan penerima tanpa merosakkan transaksi TFO. Lanjutan itu, hasil juga menunjukan tiada pengurangan prestasi secara signifikasi apabila saluran terselindung digunakan di dalam TFO. Ini menunjukan saluran terselindung boleh dibina di dalam TFO dan berfungsi normal sebagaimana TFO biasa. Di atas asas ini, ia menjadikan saluran terselindung di dalam TFO sebagai salah satu alterntatif terkini dalam pelaksanaan saluran terselindung di dalam TCP.

Kata Kunci: saluran terselindung, TCP *Fast Open*, steganografi rangkaian, keselamatan rangkaian, penyembunyian maklumat

ACKNOWLEDGEMENTS

Alhamdulillah, all praises be to Allah and salawat on prophet Muhammad. My appreciation to my supervisor Associate Professor Dr. Ainuddin Wahid bin Abdul Wahab and Associate Professor Dr. Rosli bin Salleh, my mother, my father, my family, my lecturers, my gurus, my friends, open source developers and those who help me indirectly.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

3DES	:	Triple Data Encryption Algorithm
AES	:	Advanced Encryption Standard
AW	:	Active Warden
BGP	:	Border Gateway Protocol
CC	:	Covert Channel
CPU	:	Central Processing Unit
CWR	:	Congestion Window Reduced
DF	:	Don't Fragment
DNS	:	Domain Name Server
ECE	:	ECN- echo
ECN	:	Explicit Congestion Notification
FIN	:	Finish
IoT	:	Internet of Things
IP	:	Internet Protocol
IPsec	:	Internet Protocol Security
ISCSI	:	Internet Small Computer Systems Interface
ISN	:	Initial Sequence Number
MD5	:	Message Digest algorithm 5
ms	:	Milliseconds
NAT	:	Network Address Translation
NIC	:	Network Interface Card
PDU	:	Protocol Data Unit
PSH	:	Acknowledgment
RFC	:	Request for Comments

RST	:	Reset
SDN	:	Software-defined Networking
SYN	:	Synchronise
SYN_RCVD	:	Synchronise received
SYN_SENT	:	Synchronise sent
TCP	:	Transmission Control Protocol
TFO	:	TCP Fast Open
URG	:	Urgent Pointer
VPN	:	Virtual Private Network

CHAPTER 1: INTRODUCTION

1.1 Background

Communication channel is an instrument to transmit data from one or several sources to one or several destinations. In general, channels depend on standards or protocols that consist of syntaxes that understood by both parties and bounded by communication policies. These syntaxes comprise a form of format or structure of encapsulated data in communication e.g. first 12-bit is reserved for the header.

On the other hand, there are also alternate channels that are made to breach communication policies without or least disrupt the whole communication system and this type of channel known as a covert channel. Covert channel occurs when communication mechanisms that appear out of context from standard or overt communication channel mechanisms. These covert channel mechanisms are derived from various factors such as design, architecture or nature of an overt communication system. In other words, covert channel is communication mechanisms that seem unintentionally, that never intended to be a valid communication channel and not supposed to be allowed to communicate (Lampson, 1973).

At the early stage of covert channel discovery, covert channel was viewed as a security flaw in a communication channel that can lead to data leakages or can be extended to remote execution mechanisms. For instance, an incognito device that equipped with embedded code and later on an attacker can activate by remotely to accomplish a particular task (Clark & Levin, 2009). In the latest approaches, covert channel can be combined with other data leakage technique such as side channel attack that can create a catastrophe that is hard to resolve. For example, in the case of spectre and meltdown attack, an exploited side effect in out-of-order execution on processors can create a side-channel attack which privileged data can be collected. This unauthorised collected data later is transferred by using covert channels to expose to the outside world. To utterly annihilate the attack, it requires restructuring, reengineering and redesigning of the processor itself and of course, this requires substantial efforts (Hamburg et al., 2018; Kocher et al., 2018).

In contrast, some studies also showed covert channel could also be applied for security application purposes. For example, in dubious and limited network capability environment such as in ZigBee or personal area network, covert channel can be used to send secret data between communicating parties on top of conventional cryptographic approaches (Hussain et al., 2016).

Numerous studies have found multiple covert channels exist in various network protocols. One of the undisputable protocols that globally used is Transmission Control Protocol (TCP). Since the birth of the internet, TCP plays a significant role in global internet traffic. This trend becomes more tremendous when more TCP based applications are being introduced such as internet video, online gaming and mobile applications. In fact, according to Cisco Visual Networking Index, by the year 2012, internet video traffic alone will dominate 82 percent of all consumer internet traffic by 2021 (Cisco, 2017).

In TCP, most of the implementations of covert channel primarily are focusing on mandatory fields in TCP header but less in TCP option fields (Mileva & Panajotov, 2014; Kumar et al., 2011). The main purpose of TCP Option fields is to accommodate various functionality that does not provide by ordinary TCP, and some of these options may change the behaviour of regular TCP. From time to time, the usage of TCP Option fields is expanding, for instance, since 2010, there are three new options were introduced namely TCP Authentication Option (TCP-AO), Multipath TCP (MPTCP) and TCP Fast Open Cookie (TFO) (Kay et al., 2017).

Table 1-1 shows TCP header option fields that have potential to be used widely in TCP with their year published from Request for Comments (RFC). Among the latest TCP options are TFO, MPTCP and TCP-AO. These three are based on RFC7413, RFC6824, RFC5925 and at the year of 2014, 2013 and 2010 respectively with some of them are currently pursuing to become a standard track. Moreover, these three TCP options have the largest in terms of payload compared to other TCP options as shown in APPENDIX A.

Table 1-1: TCP header Option Fields and Year published (Kay et al., 2017)

No.	Field Name	Year
1.	Maximum Segment Size	1983
2.	Window Scale	1988
3.	SACK Permitted	1996
4.	SACK	1996
5.	Timestamps	1992
6.	Quick-Start Response	2007
7.	TCP-AO	2010
8.	MPTCP	2013
9.	TFO	2014

TCP-AO is a replacement for the obsoleted Message-Digest algorithm 5 (MD5) signature option of RFC 2385 (TCP MD5). It specifies the use of stronger Message Authentication Codes (MACs) and more secure compared to TCP MD5. In terms of usage, TCP-AO is designed to replace TCP MD5 and targets for specific network equipment such as for Border Gateway Protocol (BGP) router (Touch et al., 2010).

MPTCP allows devices to use multiple network interfaces that have different network segments for a single TCP connection session. Thus, this benefits devices such as mobile phone and cloud servers to use this option (Ford et al., 2013). However, MPTCP is only supported by selected operating systems. According to D. (Murray et al., 2017), only FreeBSD, Mac OSX and IOS operating system are integrated with the kernel. While, for Linux, it needs an extra module in kernel to be compiled with. Even though the MPTCP are growing, the current deployment of MPTCP also introduces new challenges that require congestion control mechanisms which can assess and equitably share available resources across multiple paths. (D. Murray et al., 2017).

TFO permits data to be sent at the initial stage of the three-way handshake. Thus, this makes TFO saves one round trip time and can perform faster transmission as compared to ordinary TCP (Cheng et al., 2014). In terms of implementations, TFO is supported by many operating systems. According to (D. Murray et al., 2017) TFO is already supported by Linux, FreeBSD, IOS, Android, Mac OSX and Microsoft Windows. However, not all software on these operating systems supports TFO. For example, only Google Chrome browser on Linux and Android are TFO capable but not to others (D. Murray et al., 2017).

Further, although both MPTCP and TFO relatively new, a survey conducted by (D. Murray et al. 2017), TFO has more slightly higher tendency usage than MPTCP in terms of network traffic trending. Therefore, the study is focusing on exploring covert channel in TFO.

1.2 Motivation and Statement of Problem

Covert channel is one of the techniques in information hiding. Depends on purposes, some viewed covert channel as security treats. In particular, covert channels can expose sensitive data where it led to data breaches (Lampson, 1973). According to Cost of Cybercrime Report (Accenture, 2018), the average number of data breaches is growing 27.4% each year, and about 21.2 million USD is estimated regarding cybercrime in the United States alone.

On the other hand, some studies showed a covert channel is useful when transferring message securely in an untrusted ad-hoc network such as personal area networks (Hussain et al., 2016). Whether covert channel is used for security threats or vice versa, the importance of covert channel is unavoidable. Implementation of covert channels are also found in latest trend technologies such as in Internet of Things (IoT), IPv6 and cloud computing (Lewandowski et al., 2006; Hussain et al., 2016; Betz et al., 2017). Thus, the tendencies of covert channel to expand in the latest technology is promising and should cover various areas such as the latest options in TCP.

Moreover, TCP is core protocol in internet traffic. Samples internet traffic that done by (D. Murray et al., 2017) showed 87.56% of internet traffic is made up from TCP. Thus, these push factors have made TCP one of the preferred implementations of covert channel.

Therefore, the introduction of covert channel to new TCP technologies such as TFO can help to widen in creating more alternative in implementing covert channel. As yet, to the best of the author knowledge, there is no covert channel implementations have been done in TFO so far.

Hence, the primary research questions are "*Can we implement covert channel in TFO? If it can be implemented, does the covert channel implementation is keep intact with the TFO objective which is performance for data transferring*."

1.3 Aims and Objectives

This study aims to introduce covert channel to TFO. To achieve this aim, the primary objectives of this research as follows:

- To report covert channel and implementation TFO that used in practice;
- To implement covert channel in TFO; and
- To evaluate correctness and performance of covert channel in TFO.

1.4 Scope of Study

In this study, the approach is applied as much as possible that imitate as in a real-world scenario. Thus, the study applies covert channel in TFO as in client-server in a web environment that valid in practice. However, this only limited on one type of specific traffic and does not represent all types of traffics. Further, some of the processes and test require some assumptions that are unavoidable. In this study, list of assumptions will be covered in Chapter 3.

1.5 Thesis Outline

This thesis comprises of five chapters which consist of:

- Chapter 1 presents the introduction of the study, background of covert channels, problem statement, thesis objectives and the study scopes.
- Chapter 2 presents the literature review on covert channels, design concepts of covert channel in general, covert channel in TCP protocol, understanding TCP protocol and TFO.
- Chapter 3 presents the methodology and design of this study to achieve the objectives.
- Chapter 4 presents the implementation and execution of covert channel tools. It also introduces how tests were conducted.
- Chapter 5 presents discussion about the experiment results and outcome.
- Chapter 6 concludes the thesis by theoretical considerations and practical implications of the method. Finally, the limitation of this research is discussed and some future works are proposed.

CHAPTER 2: LITERATURE REVIEW

This chapter explains a literature review and studies the current approach in covert channel as fundamental ideas which include how covert channel can be designed. This followed by TCP as overview background subject based on RFC 793 before entering TFO as the specific target for this study. In this section, it includes differences TCP and TFO and also implementation TFO in Linux. Next section, is focusing on the previous approaches covert channel in TCP which generally include on techniques and payload sizes.

2.1 Covert Channel

U.S Department of Defense defines covert channel is "*any communication channel that can be exploited by a process to transfer information in a manner that violates the system's security policy*" (Latham, 1986) . Covert literally means hidden, whereas channel means medium of communication. Thus, covert channel simply can be described as hidden communication.

Covert channels are divided into two categories namely, storage covert channel and timing covert channel. Storage covert channel manipulates storage to create covert content (Gray, 1994). This may include payload, header and reserved fields in data that are potential to create covert content. Whereas, timing covert channel manipulate timing behaviour to create code that represents messages (Zander et al., 2007).

Further, covert channel also exists in cryptographic algorithm, where it exploits the undetermined value such as random values in cryptographic algorithm. This sub category is known as subliminal channel. Further, combining various techniques create hybrid covert channel such as in (Mazurczyk et al., 2009; Simmons, 1993). The overall covert channel can be illustrated as in Figure 2-1:

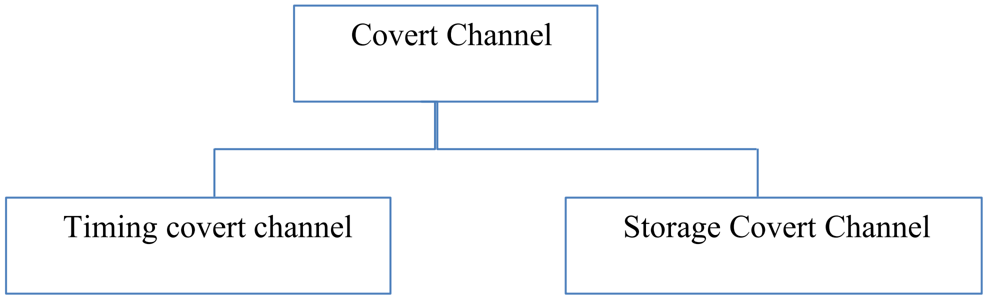

Figure 2-1: Categories in covert channel

2.1.1 Communication Model

There are several types of communication model in covert channel. It can be described using the famous prisoner problem that introduced by (Simmons, 1984). The problem is about Bob and Alice were thrown into prison and wanted to escape. Wendy a warden prisoner always monitors any communication between prisoners and Bob and Alice want to communicate without her notice. Bob and Alice must communicate using innocuous messages that contains hidden information to prevent Wendy's doubt. Any messages must go thru Wendy and each message can be read or modified by her.

Extending from this scenario, there are at least four type communication situations that possible to create (Zander et al., 2007). Assume Bob and Alice are overt channel users and Mallory and Mallet are covert channel users where Mallory and Mallet communicate secretly using Alice and Bob communication without they notice. Mallet and Mallory are covert channel implementer where they can Alice and Bob themselves or another person. Table 2-1 showed various communication can be made in implementing cover channel. The first one includes both Alice / Mallory and Bob / Mallet are located at the same site respectively, where covert activity are being done at same location (at the same host). Whereas second approach, Mallory located at the middle of overt channel. In this case, Mallory intercept every Alice communication sessions and injected covert content in overt channel. Similarly, the third approach are the reverse of second approach where

Mallet intercept Bob communication sessions and injected covert content in overt channel. While fourth approach, Mallory and Mallet are both outside of Alice and Bob's location. In real word, it would be the covert channel users are located at network device such as router and network proxies.

Table 2-1: Various Communication Models

No.	Overt Sender	Channel	Overt Receiver
1.	Alice	Overt channel	Bob
	Mallory	Covert channel	Mallet
2.	Alice	Overt channel	Bob
		Mallory, Covert channel	Mallet
3.	Alice	Overt channel	Bob
	Mallory	Covert channel, Mallet	
4.	Alice	Overt channel	Bob
		Mallory, Covert channel, Mallet	

2.1.2 Covert Channel Techniques

According to (Wendzel et al., 2015) covert channel techniques can be generalised into eleven common categories. Four of them, are belong to time covert channel and other fall under storage covert channel. Some of these techniques may have hindrance on the structure which can be categorised into three factors namely syntax, semantic and noise. Syntax is about the structure format of PDU such as first 8 bytes field in structure are reserved for header. While semantic is a logical of interpretation of the PDU. For example, IP timestamp option is to measure *catenet* delays which if any misconfigured value may lead into wrong interpretation and sometime would interrupt the whole process. While noise is a form of overhead that occurs from unwanted condition such as bit corruptions or packet loss which may disturb transaction. In conclusion, from all patterns listed, only random value pattern is generally preserved syntax, semantic and noiseless of channels. Table 2-2 explains the category techniques with their caveats (Wendzel et al., 2015).

Table 2-2: Covert Channel Techniques (Wendzel et al., 2015)

	Technique	Type	Syntax	Semantic	Noise	Description	Example
1	Size Modulation Pattern	Storage	√			Size manipulation of a header element or Protocol Data Unit (PDU) to store covert content.	Manipulating padding field's size in IEEE 802.3
2	Sequence Pattern	Storage	√			Order manipulation of header or PDU to store covert content.	Manipulating options or position or number of options in DHCP option
3	Add Redundancy Pattern	Storage	√			Additional area within header element or PDU to store covert content.	Adding additional fields in HTTP headers.
4	PDU Corruption/Loss Pattern	Storage	√		√	Corrupted PDU generation to store covert content.	Transferring corrupted frames in IEEE 802.11
5	Random Value Pattern	Storage				Covert content inserting in random value of a header element.	Manipulating identification field in IP header
6	Value Modulation Pattern	Storage		√	√	Selects one of *n* values a header element.	Manipulating Least Significant Bit (LSB) into the IPv4 timestamp option
7	Reserved/Unused Pattern	Storage		√		Use reserved/unused to store covert content.	Utilizing unused fields in IPv4 or TCP
8	Inter-arrival Time Pattern	Timing			√	Timing intervals manipulation to encode covert content.	Creating delays into inter-arrival times of SSH packets
9	Rate	Timing			√	Data rate manipulation in traffic flow to encode covert content.	Sending specific commands in serial communication port to alter throughput
10	PDU Order Pattern	Timing	√		√	Artificial PDU order creation to encode covert content.	Re-ordering of IPSec Authentication header (AH) packets
11	Re-Transmission Pattern	Timing			√	Intentionally re-transmit PDU to encode covert content.	Retransmitting intended unacknowledged packets

Further, according to (Lewandowski, 2011) there are two essential requirements in creating covert channel need to be considered. First, any covert channel transactions as much as possible should comply with channel syntax. Any violation of the channel syntax may result in channel failure or exposing the covert channel. For example, imbedding covert content in reserved fields sometimes can be successful but in environment that applies strict rules in firewall may discard the packet or can expose the content. Finally, covert channel also should consider channel semantic. Ineffective of preserving the semantic part may result anomaly situation or may amend the meaning of traffic. For instance, applying covert channel in reserved field flags unproperly can change the packet behaviour such as IP flags set into DF which may result the following packet can be dropped. Thus, to increase reliability of covert channel requires channel syntax and semantics knowledge deeply (Lewandowski, 2011).

2.1.3 Wardens

Extending from the Section 2.1.1, traditionally there are two types of wardens: passive warden and active warden. Passive warden is passive way of detecting covert channel and can be time consuming (Zawawi et al., 2012). It operates by monitoring traffic and create alarm when it detects any suspicious activities. Numerous techniques such as statistical, probabilistic and machine learning as described in (Murdoch et al., 2005; Tumoian et al., 2005; Zhai et al., 2010) are used in passive warden. Some of these techniques requires large samples of packets to analyse for instance, (Kumar et al., 2011) used 50 samples to detect anomaly traffic pattern in their findings.

On the other hand, active wardens are actively preventing covert channel, these include normalising packet operation such as dropping, correcting or modifying content in a packet. In certain techniques, all inspections are assumed content hidden messages. In other words, active warden is more on preventing rather than detecting as passive warden does (Lewandowski et al., 2006).

(Fisk et al., 2002) introduced Minimal Requisite Fidelity (MRF) concept that try to balance between end user needs and create distortion to covert communication. It implemented various covert channel countermeasure techniques at inline at network level in a system similar to a firewall, network proxy or intrusion prevention system. In fact, it performs similarly to a covert channel by replacing noise as the hindrance. The idea of this approach is to disturb suspicious covert area while maintaining semantics and syntax of a structure. The principles of MRF can be described as follows:

1. Value correction on known values.

Any value must follow standard such as change all reserves bits to default value for example: zero or recalculate checksum if the checksum is false.

2. Eliminate value on unknown values.

If the value is unknown for such as ID that uses random value. Replace with new value and create bijective mapping to the source or simply drop it.

3. Scramble value between known and unknown values.

Applying noise to received value which may contents ambiguous area such as in least significant bit or intended streaming in network traffic (Fisk et al., 2002). Extended work for MRF, (Lewandowski et al., 2006) enhanced the method by adding information surrounding network in in order to decrease the entropy present in IPv6 protocol fields.

Furthermore, some of these approaches are also found in other implementation but for different purposes, for example OpenBSD's *pf* is capable to detect invalid flag combinations in order to remove ambiguities in packet (OpenBSD, 2005). Furthermore, some studies in protecting from various network attacks are using similar concept for example (Kreibich et al., 2001) presented packet normalisation in order to protect from numerous attack such as *stateholding* attacks.

2.2 Transmission Control Protocol (TCP)

TCP protocol is a common protocol use in internet. It designed to be end-to-end reliable protocol and works in multi-layer network. TCP protocol is based on RFC 793 (Postel, 1981) where it defines standards and protocol specifications. TCP comprises header segment that contains information fields and data segment which carries user data.

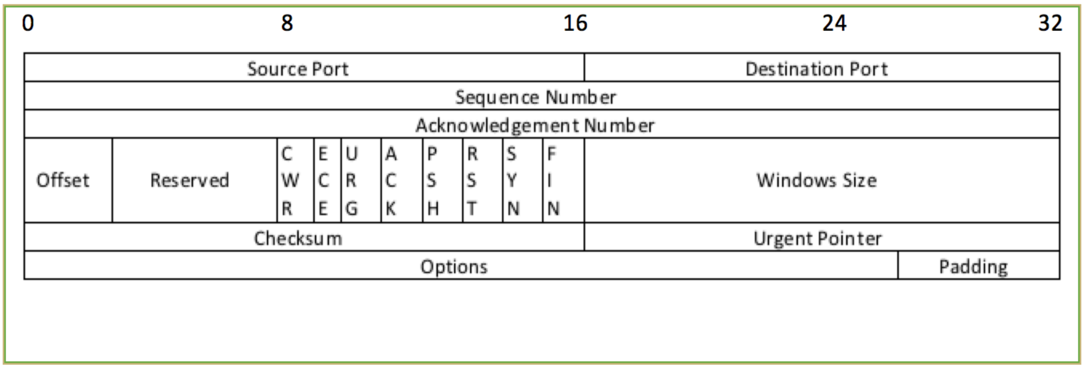

Figure 2-2: TCP Header Format

TCP header encompass ten mandatory fields without padding and an option as shown in Figure 2-2.

- Source Port: 16 bits

  Contains sender port number.

- Destination Port: 16 bits

  Contains receiver port number.

- Sequence number: 32 bits

  Contains the sequence number of the sender. It starts with initial sequence number where SYN in control bit is equals to 1.

- Acknowledgment Number: 32 bits

  Contains the sequence number from the receiver. It gets increased when data get transmitted.

- Data Offset: 4 bits

  Contains TCP header size. The size can range from 20 bytes up to 40 bytes where it calculated using word or bits value multiply by 4

- Reserved: 4 bits

  Reserved flag bit for future use. RFC 3540 (Spring et al., 2003) occupied one bit make it reduced into 3 bits left.

- Control Bits: 8 bits

  Consists of 9 1-bits flags as follows:

  a. CWR: 1 bit
  b. ECE: 1 bit
  c. URG: 1 bit
  d. ACK: 1 bit
  e. PSH: 1 bit
  f. RST: 1 bit
  g. SYN: 1 bit
  h. FIN: 1 bit
  i. Window: 16 bits

  Contains the size usually in bytes of receive window size

- Checksum: 16 bits

  Consists of checksum of source IP address, destination IP address, protocol number and the length of TCP header including payload.

- Urgent Pointer: 16 bits

  Contains byte position of data that should be process immediately.

- Options: 0 -320 bits (variable)

  TCP option contains verities of options and each of them has specific purposes. The list of TCP options is summarised in APPENDIX A. The define option in TCP option is either in a single octet or multiple octets. In multiple octets, it consists of Option-kind, Option-length and Option-data. Option-kind contents kind number that indicate TCP option kind for example kind number two is belong to Maximum Segment Size. Option-length is the size of payload in kind number for instance, the size of Maximum Segment Size is four. Meanwhile Option-data is the payload of kind number itself where its content information of TCP option operation such as timestamp value in timestamp option in TCP. On the other hand, single octet only consists 1 byte that currently applied to End of operation List and No-Operation. End of operation List has been used to indicates the end of the option list. While No-Operation is used for filler between options (Postel, 1981).

- Padding: (variable)

  The padding is zero bits filler to ensure the size of TCP header to make it an even multiple of 32 bits in TCP header especially when TCP header contains TCP options.

2.2.1 Three-way handshake in TCP

Based on RFC 793 (Postel, 1981), TCP is a connection-oriented protocol and uses three-way handshake to establish connection. In general, TCP operates under three processes which are open connections, close connections and established connection for data transferring purposes in a pair of sockets. Each of the processes involving specific or combination of the ACK, SYN, PSH, FIN and RST flags. The whole operations in TCP are based on transition states that occur in socket as showed in Figure 2-3 and described briefly in Table 2-3.

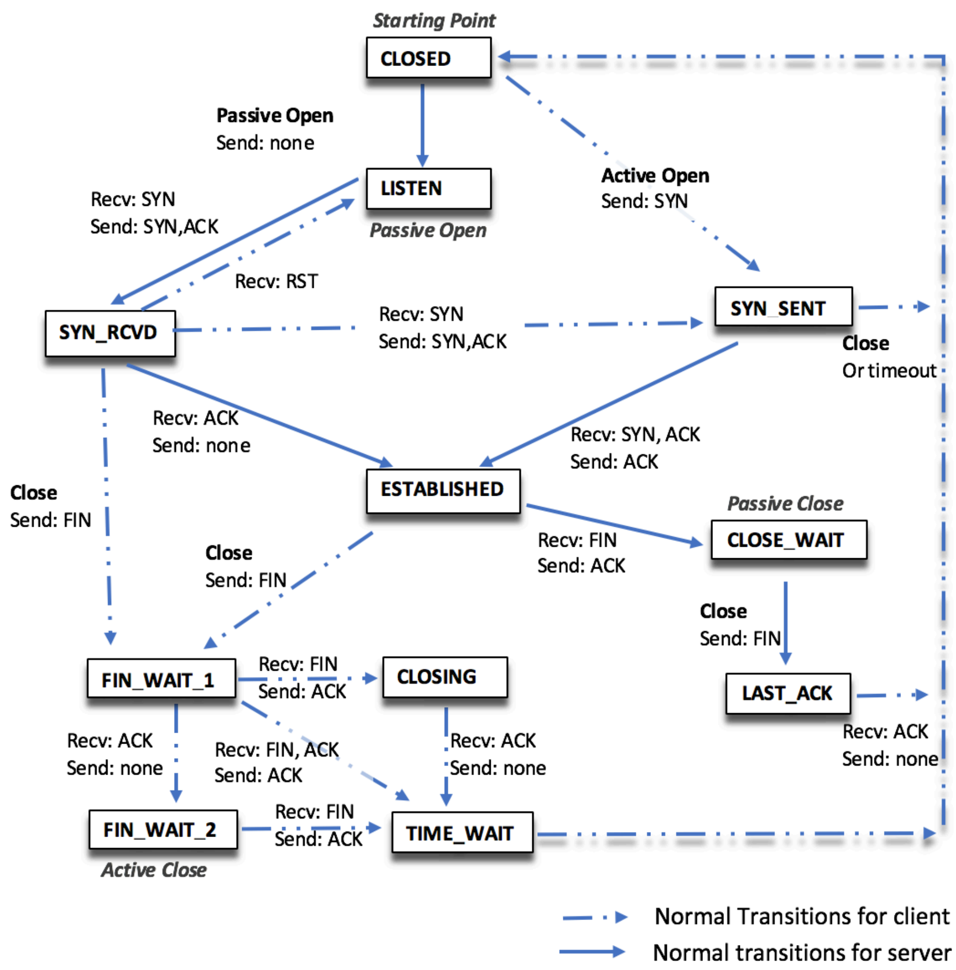

Figure 2-3: Transition State in TCP (Stevens, W., Fenner, B., & Rudoff, A., 1999)

Table 2-3: TCP state descriptions (IBM Knowledge Center. (2017)

TCP Connection State	Description
LISTEN	Waiting for a connection request
SYN-SENT	Waiting for an acknowledgment from the remote node after having sent a connection request.
SYN-RCVD	Received a connection request and sent an acknowledgment.
ESTABLISHED	Data transfer phase of the connection.
FIN-WAIT-1	Waiting for an acknowledgment of the connection termination request or for a simultaneous connection termination request from the remote node.
FIN-WAIT-2	Waiting for a connection termination request from the remote TCP after this node has sent its connection termination request.
CLOSE-WAIT	The node has received a close request from the remote node and at this state it waits for a connection termination request.
CLOSING	Waiting for a connection termination request acknowledgment from the remote node. This state is entered when this node receives a close request from the local application, sends a termination request to the remote node, and receives a termination request before it receives the acknowledgment from the remote node.
LAST-ACK	Waiting for an acknowledgment of the connection termination request previously sent to the remote node. This state is entered when this node received a termination request before it sent its termination request.
TIME-WAIT	Waiting for enough time to pass to be sure the remote node received the acknowledgment of its connection termination request.
CLOSED	Represents no connection state at all.

An example of full simple cycle of TCP states is shown in Figure 2-4 and can be described as follows:

1. Initially, server is set to LISTEN state and client is under CLOSED state.
2. When client want to make connection, it sends request to server and create initial sequence number as reference number that to be used later. The sequence number field is filled up with the sequence number. During this stage, control flag for SYN is set, other flags must be unset, TCP state for client = SYN_SENT, server =SYN_RCVD and no data in data payload.
3. Server replies by sending its own first sequence number in sequence number field and client's sequence number + 1 in acknowledge number field. During this stage, control flag for SYN and ACK is set, other flags must be unset, TCP state client = ESTABLISHED, server =SYN_RCVD and no data in data payload.
4. Client replies by sending its own sequence number in sequence number field and server's sequence number + 1 in acknowledge number field. During this stage, control flag for SYN and ACK is set, other flags must be unset, TCP state client = ESTABLISHED, server = ESTABLISHED and no data in data payload. After this, data start to exchange and uses PSH, ACK and SYN flags until the client send FIN flag.
5. When client starts to send FIN flag and received by server, server will change to CLOSE_WAIT state and it replies back ACK to the client.
6. Upon receive, client change the state to FIN_WAIT_2 and wait server to reply FIN ACK packet. Once the client received FIN ACK from server, it changes to TIME-WAIT and reply ACK to server. The purpose of having TIME-WAIT is to make sure server has adequate time to receive the ACK packet and eventually client back to CLOSED state as its initial state.

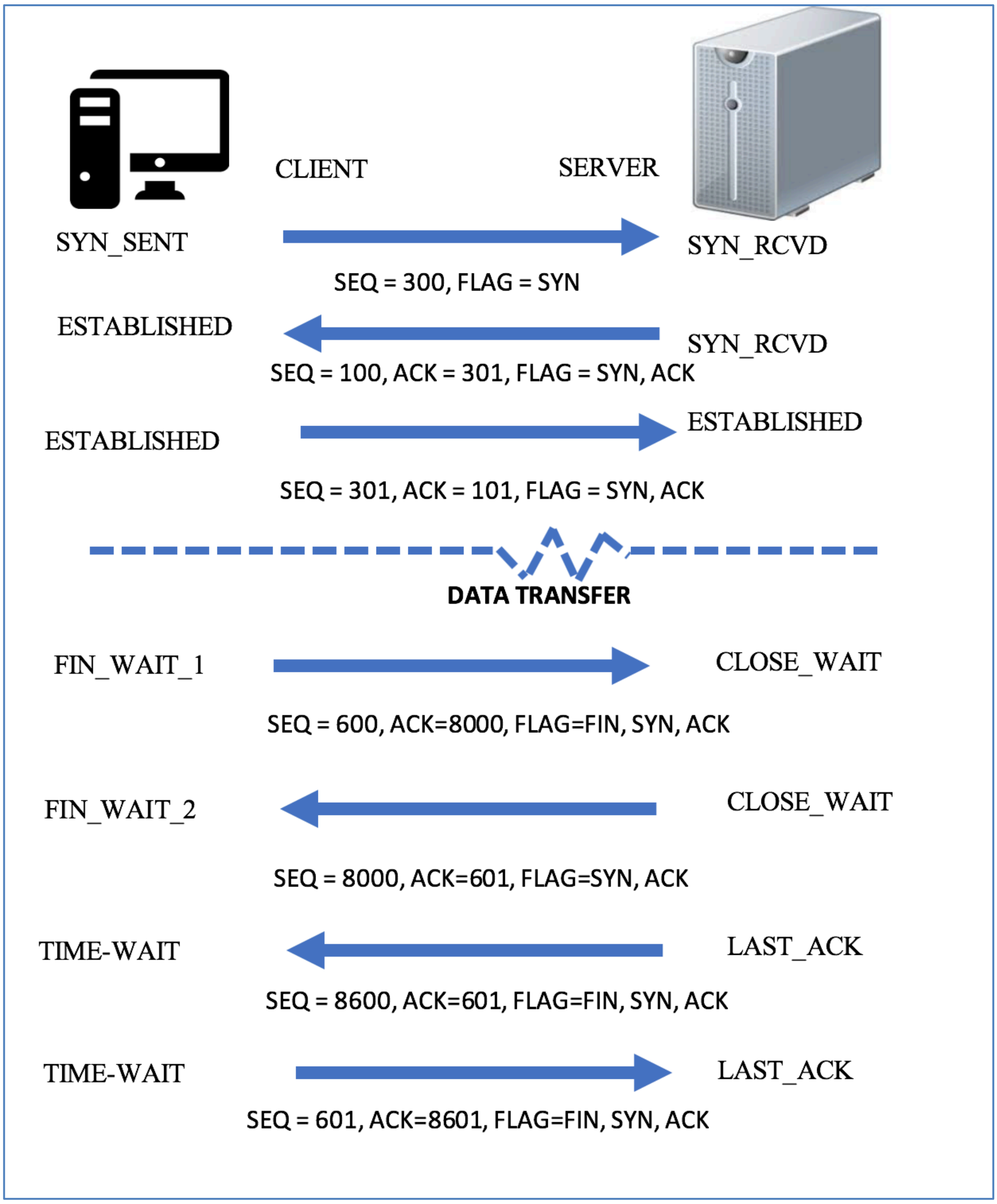

Figure 2-4: Example of three-way handshakes operation

2.3 TCP Fast Open (TFO)

(Radhakrishnan et al., 2011) introduced TFO as one of the TCP options in TCP protocol. It aims to utilise data exchange in TCP by starting to send data at three-way handshake stage. For this reason, TFO can create more efficient in data transferring; thus, this can give faster transmission. In terms of implementation, RFC 7413 (Cheng et al., 2014) defines on TFO specification and explanation in this section is based on RFC 7413. Previously, Transactional Transmission Control Protocol or T/TCP was introduced by

(Braden, 1994) which it bypass three-way handshake in TCP. However, T/TCP does not have flood protection mechanism which it cannot apply TCP SYN cookie, hence, this will lead and prone to flood attack (*Phrack Magazine,* 1998). Due to this security factor, T/TCP implementation was abandoned and it has been changed into historic status in RFC (Duke et al., 2006).

Based on RFC 7413 (Cheng et al., 2014), TFO has similar aim as T/TCP but enhanced with security features. It equipped with TFO cookie where it works like a credential and contains unique value for each client IP addresses. TFO does not drop or reject packets, if any invalid cookie the server will revert to normal TCP. This will mitigate resources from exhaustion and diminishes amplification attack such as flood attack.

Another feature in TFO, it has pending requests threshold; wherein some occasions, the server may be overloaded and unable serve requests. For instance, if the valid cookie gets comprised from trojan attack, it still subject to defined limit on pending request. Once the pending request exceeds the limit, the server temporarily disables TFO and uses normal TCP. This will allow SYN flood countermeasure techniques take place such as SYN cookies (Cheng et al., 2014).

TFO operates at three-way handshake stage as shown in Figure 2-5, in first connection session, client creates a TCP request along TFO option with empty TFO cookie. Then the server transfers TFO cookie to client as token and normal TCP proceed normally. After that, in second connection the client sends request along with TFO cookie and data. Then, upon receive, the server validates the cookie, if the cookie is valid, then the server replies SYN-ACK where ACK is total of client's SYN + 1 and size of received data. However, in case of invalid cookie the ACK value is just client's SYN + 1. In case of valid TFO, instead of waiting ACK from the client, the server starts to send data while the client is also replying request to server. Whereas in case of invalid TFO, the server must wait ACK from the client and proceed to normal TCP. Finally, in both situations, at this stage the transactions are resume as normal TCP transaction.

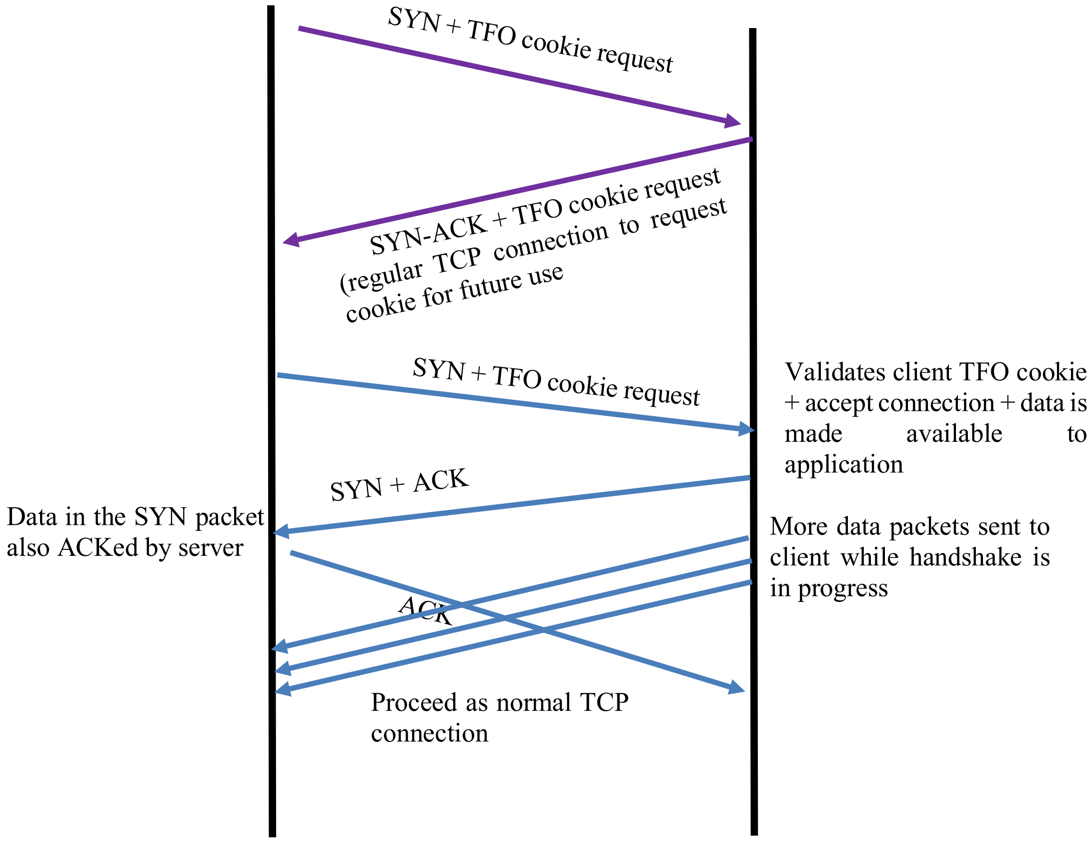

Figure 2-5 : TFO connectivity

2.3.1 TFO Structure

RFC 7413 (Cheng et al., 2014) divides TFO structures into three compartments namely, kind number which is equals to 32, length which indicates the length of TFO segment and cookie as payload of the segment as shown in Figure 2-6. The length can be various, it ranges from 2 up to 18 and must be even. Further, TFO operates in three-way handshake stage, thus SYN flag must always be set and failure to comply TFO must be ignored. The cookie size is calculated by length value subtracted by 2. TFO cookie as suggested is a message authentication code (MAC) that comprise following properties (Cheng et al., 2014):

1. Only server can generate TFO cookie and cannot be produced by other parties.
2. As suggested, it generated by encrypting IP address with AES 128 and IP address can be IPv4 or IPv6.
3. Upon verification, server reproduce the cookie and make comparison with client's IP address.
4. Encryption key can be changed manually or periodically by server or can be expired as security consideration.

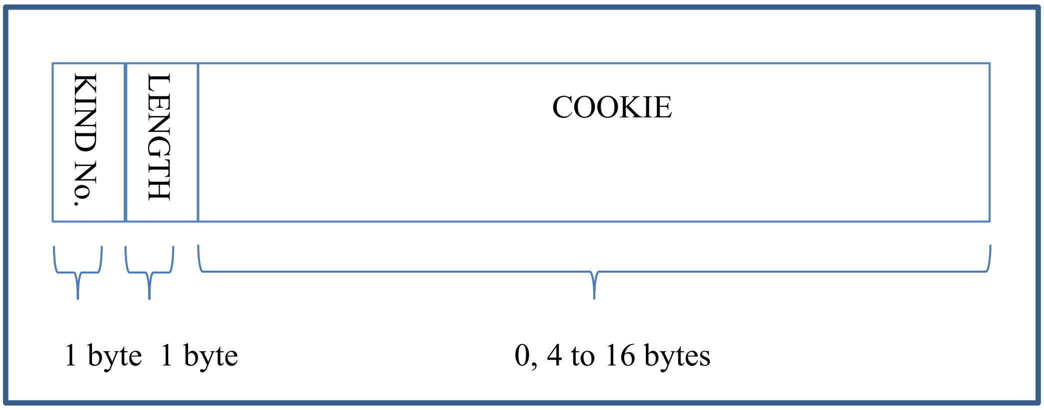

Figure 2-6:TFO structure with length size

2.3.2 TFO Implementations

There are many applications that TFO enable that cover many major aspects in internet applications and operating system. Some of them are still in the beta mode and other can be used in productions. Some of the list as shown in Table 2-4 .

Table 2-4: List of TFO Applications

	Software Name	Purposes	References
1	Linux (starting from 3.6)	Operating System	Kerrisk, M. (2012)
2	FreeBSD	Operating System	Kelsey, P. (2018)
3	Windows Server 2016	Operating System	Huitema, C. (2016)
4	IOS and Mac OS	Operating System	Prabhakar, L., & Cheshire, S. (2015)
5	Nginx	Web Server / HTTP proxy	Nginx (2017)
6	HaProxy	TCP Proxy/Load Balancer	Tarreau, w. (2016)
7	Google Chrome	Web Browser	Chrome. (2017)
8	Bind	DNS	BIND. (2016)
9	Exim	Mail Server	Harris, J. (2016)
10	Juniper	Network Operating System	Juniper. (2017)

The first implementation of TFO was based modification on Linux Operating System and used chrome and Apache as client server software (Radhakrishnan et al., 2011). Since this is the first implementation of TFO, the detail of the implementation is in next section.

2.3.3 TFO in Linux

In Linux, it started with 3.6 kernel TFO for client (Kerrisk, 2012) and 3.7 version (Vaughan-Nichols, 2012) for server. In this study, Linux kernel version 3.10 and reference in (Linux Kernel Documentation, 2017) is used to conduct the study. There are two components TFO in Linux, key configuration and mode options in TFO. The key can be maintained (renew) via two methods, the first one by restarting the operating system and the other manual key in via command line.

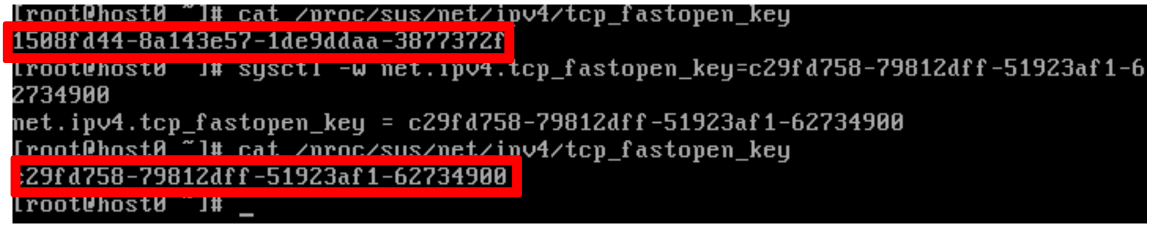

Figure 2-7: TFO key is displayed and get changed in Linux environment

By default, the key is using random values and it is stored in */proc/sys/net/ipv4/tcp_fastopen_key*. The TFO key also can be changed manually via *sysctl* command and the key size must be in 16 bytes, which is equivalent to Advanced Encryption Standard (AES) 128 bit key. The key format is 32-character hex strings, broken into 4 blocks and separated by dashes as shown in Figure 2-7.

As general, creation TFO cookie can be described in Equation 2-1 . Although according to RFC 7413 (Cheng et al., 2014) the TFO cookie size is not fixed, but it turns out that in Linux the cookie is truncated to 64 bits. This can be confirmed thru observing TFO cookie in network traffic such as in Figure 2-8 and snippet kernel's source codes as in Figure 2-9.

TFO Cookie = TRUNCATE(Ek(IPsrc , IPdst , PAD(0)),64) **2-1**

where

Ek = Encrypt (AES 128)

IPsrc = Source IP Address

IPdst = Destination IP Address

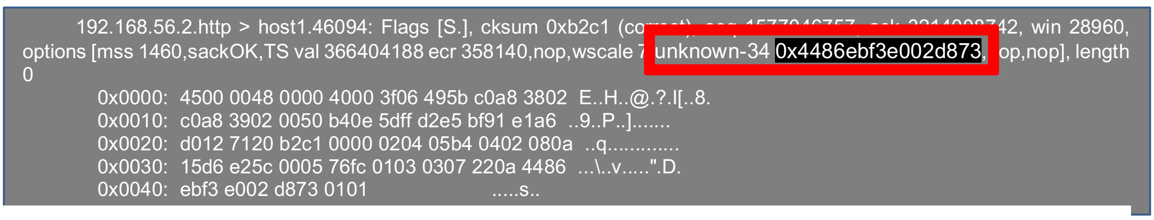

Figure 2-8: Sample of TFO traffic, unknown-34 yields kind number followed by 16 hexadecimals of TFO cookie

```
-- tcp.h --

   /* TCP Fast Open */

   #define TCP_FASTOPEN_COOKIE_MIN     4          /* Min Fast Open Cookie size
in bytes */

   #define TCP_FASTOPEN_COOKIE_MAX     16         /* Max Fast Open Cookie size
in bytes */

   #define TCP_FASTOPEN_COOKIE_SIZE 8  /* the size employed by this impl. */

--tcp_fastopen.c--

   void tcp_fastopen_cookie_gen(__be32 addr, struct tcp_fastopen_cookie *foc)

   {

        __be32 peer_addr[4] = { addr, 0, 0, 0 };

        struct tcp_fastopen_context *ctx;

        rcu_read_lock();

        ctx = rcu_dereference(tcp_fastopen_ctx);

        if (ctx) {

                crypto_cipher_encrypt_one(ctx->tfm,

                                          foc->val,

                                          (__u8 *)peer_addr);

                foc->len = TCP_FASTOPEN_COOKIE_SIZE;

        }

        rcu_read_unlock();

   }
```

Figure 2-9: TFO cookie yields the size = 8 bytes ("Linux Kernel Documentation", 2017)

Although, in practise TFO cookie is reduce into 8 bytes, yet it is still remains one of the largest content random values in contemporary TCP fields (Postel, 1981; Kay et al., 2017). In Linux, there are five options in TFO that can be used as stated in Table 2-5. From the table, TFO can be into two situations; for trusted environment such as in Internet

Small Computer Systems Interface (ISCSI) or in internal communication among nodes in cloud environment, option 0x4 and 0x200 can be used. And under normal circumstances and according to RFC 7413 (Cheng et al., 2014) standard, option 0x1 and 0x2 should be used which it requires valid TFO cookie as authentication. These TFO options is stored in */proc/sys/net/ipv4/tcp_fastopen* and to set the value manually *sysctl* command is used or it can be set in */etc/sysctl.conf* as permanent configuration.

Table 2-5: TFO modes in Linux

	Value	Meaning
1	0x1	Enables client-side support
2	0x2	Enables server-side support
3	0x4	Enables TFO at client side with or without TFO cookie
4	0x200	Enables TFO at server accept with or without TFO cookie
5	0x400	Enables all listeners to support TFO automatically in socket

2.4 Covert Channel in TCP

According to study done by (Mileva & Panajotov, 2014), there are 13 identified covert channel techniques that have been identified in TCP. From there, 69.23% of the approaches are categorised as storage covert channel and most of their approaches channel utilised TCP header fields as its covert carrier such as shown in Table 2-6 with additional from works from (Kumar et al., 2011, Efanov et al., 2017) . Further, 55.56% from that, there are related to TCP sequence number such as ACK and initial sequence number (ISN).

Table 2-6: Covert Channel with covert content size

	Paper/tool/Solutions	Years	TCP Fields / behaviours	Payload size	Type
1	Covert TCP	1997	ISN & ACK	64	Storage
2	Hintz	2002	Urgent pointer	16	Storage
3	Abad	2001	Header checksum	16	Storage
4	NUSHU	2004	ISN	32	Storage
5	Lantra	2005	ISN	32	Storage
6	Allix	2007	Reserved N packet	4	Storage
7	CLACK	2009	ACK	32	Storage
8	RSTEG	2010	Retransmission	Max IP segment	Storage
9	ACKLeaks	2011	ACK	32	Storage
10	Giffin et al.	2002	TCP Timestamp	1	Timing
11	Chakinala et al.	2005	Segment Reordering	Log_2 n!	Timing
12	Cloak	2007	X TCP flows	n	Timing
13	TCP scripts	2008	TCP Bursts	n	Timing
14	Kumar et al.	2011	Maximum Segment size (MSS) & ISN	n	Storage
15.	Efanov et al.	2017	Port	16	Storage

Note. Information no. 1 to 13 from (Mileva & Panajotov ,2014), for 14 from (Kumar et al., 2011) and 15 from (Efanov et al., 2017).

The main of the function of TCP sequence number is to mitigate from off-path attacks e.g. trust-relationship exploitation and denial-of-service attacks (F.Gont, 2012). However, this will resulted ambiguous situations where sequence number is hard to distinguish between encrypted value and genuine generated sequence number (Fisk, et al. 2002). As a result, many storage covert channel implementations are based on this property.

From the survey, it indicates many implementations of covert channel in TCP header have a tendency towards implementing on mandatory fields with some exception implementation that use TCP options such as timestamp using weakness in Least Significant Bits techniques and mixture between sequence number and Maximum Segment size (MSS). Also, it is suggested, storage covert channel is preferable than time covert channel as in (Mileva & Panajotov, 2014).

2.5 Summary

This chapter presented a various overview of covert channel, current TCP, TFO and covert channel in TCP. Section 2.1 explains covert channels in generals, ranging from concepts to attack against covert channel. Although implementations of a covert channel are varied and depend on how syntax and semantic in overt channel works, the key philosophies in building covert channel much more the same. Section 2.2 presented a comprehensive overview of TCP to describe the structure and operation of the whole TCP process. In Section 2.3, explains the state-of-art of TFO comprehensively, it comprises concepts, structures, differences as compared to standard TCP and implementations. In particular, the study covered Linux as a real-life TFO implementation that used in practice. It turns out that, some of the TFO properties in Linux are different from RFC standard; in particular the size of TFO cookie is truncate to 64 bits rather than 128 bits; as well as verities mode of using TFO cookie. Section 2.4 presented covert channels that have been implemented in TCP protocol. The Conclusion of this section showed implementations of covert channels in TCP are have more favourable to use storage type and make used random properties. Hence, theoretically it is possible to apply the similar approach to TFO by using TFO cookie as covert career.

CHAPTER 3: RESEARCH METHODOLGY AND DESIGN

This chapter provides methods that describe activity of research in order to answer research questions. The thesis is separated into four phases namely, building covert communication model, create assumptions, prototyping and tests. Building covert communication defines which communication model that used in the study. Next, create assumptions explains components and factors that must be followed in order to make prototype works. Then create prototype which resembles creating tools in environment. Finally, designing the test in order to test the running tools and to get data collection.

3.1 Building Covert Communication Model

There are many types of covert communication models as mentioned in Section in **2.1**, in this study, again the thesis illustrates it by using the famous prison problem where Alice is thrown to prison and the only person Alice can communicate is Bob. The communication they can use are normal TCP communication only. Every session of communication is filtered by warden. As security measurement, warden will inspect and block anything that suspicious and implement known active warden techniques. Later, to increase efficiencies, the prison department has implemented TFO in their network communication.

At the same time Alice wants to send covert message to Mallory as a secret receiver over TFO channel. Suppose, long before that, Alice and Mallory have an identical shared key and they also have information that recently the department has implemented TFO in the network. From here, a communication model can be designed as a conceptual flow so that Alice and Mallory enable to communicate as follows:

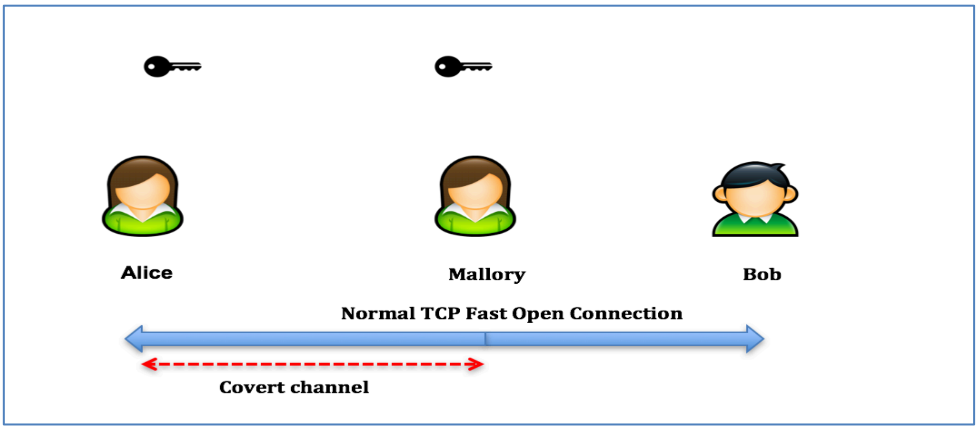

Figure 3-1: Covert Channel communication model in TFO

1. Alice and Bob use normal TFO communication.
2. Suppose Mallory and Alice have secret communication.
3. Alice and Mallory already have secret key.
4. Alice request data from Bob and Bob replies.
5. Mallory intercept Bob's packet and replace TFO cookie with encrypted secret message using secret key.
6. Alice receive packet from Bob and proceed normal operation.
7. Alice extract message from TFO cookie using secret key

For implementation purposes, Alice and warden reside in the same location (node) as well as Bob and Mallory.

3.2 Create Assumptions

In order to investigate and create a scenario case, assumptions must be made and must in line with Section 3.1 and follow RFC 7413. Assumptions are carefully designed so that it resembles real implementation scenario and do not disturb any existing process in TFO. In this research assumptions are based on the following scenarios:

1. TFO runs without any blocking

To create overt channel all nodes and network traffic need to unblock any TFO activities and traffic. It includes enabling TFO setting on both nodes and ensure there are no host middlebox (e.g. firewall, Intrusion Prevention System and router) blocks. Further, network environments that require TCP level packet modification such as NAT, reverse proxy and VPN. need to be avoided in order to maintain transaction that carries TFO properties work well. Thus, only the regular routed network is used to ensure TFO can be transmitted without having any problems.

2. Fixed covert content size

Covert content size needs to be in fixed length, due to the content depends on TFO cookie (covert carrier) size. Any changes in covert carrier (TFO cookie) size can lead to anomaly structure in TFO syntax. Although in RFC 7413 stated TFO cookie can be range 6 to 18 bytes, in practice it fixed into 64-bit size TFO cookie (Cheng et al., 2014).

3. Node specification, bandwidth and latencies are uniformed

In order to create a stable environment, bandwidth and latencies are fixed to avoid unbalanced results. Moreover, this also applies to the host's specification namely, CPU speed, memory capacity and NIC speed factors.

4. Covert content is fixed and one-time simplex communication

Covert carriers are embedded in TFO cookie and it uses the same TFO cookie that exists in SYN-ACK state in TCP from server and SYN state from a client. In a typical situation, TFO cookies are infrequently changed. Thus, creating non-static covert message would lead to anomaly pattern in TFO cookie. However, if TFO cookie get changed due to some occasion such as system reboot, intentionally change TFO cookie key, cookie expiry, etc. the system must able to adapts to avoid any breaking semantics in TFO session.

5. Covert content is receivable within some period of times

For increment reliability of transferring message, it is appropriate to maintain the message for some amount of time before it gets dissolved. It makes covert receivers do not require to receive transactions exactly same time as covert sender send the covert message.

6. Covert receiver has adequate keys to extract covert content.

Covert receiver has sufficient requirements to do decryption and extract the message from TFO cookie. Since TFO cookies might change from time to time, it is important extraction process in aware and adapts the situation.

3.3 Prototyping

The main goal is to create confirmation against Section 3.1 , thus in this study, the involved techniques consist of creating algorithm, creating covert channel and simulate performance observation. Furthermore, in order to create the tools, TFO as overt channel is configured and two nodes as data sink and data source are involved. For performance simulation, the concept is based on setup that had been done in (Radhakrishnan et al., 2011) delays or latencies in network are created to simulate various conditions. Thus, an intermediate node is required to perform delay controller. In this study, the delays is set to 0 millisecond, 5 milliseconds, 10 milliseconds and 15 milliseconds respectively in order to create throttling in traffic.

For TFO adoption, this study uses Linux operating system to create nodes namely, client, server and router. The software of both client and server are chosen based on TFO supported that suitable to create web environment give near real life example scenario. The following are software that used in the study:

- mget as web client,
- nginx as web server,
- Built in Linux function as router; and
- Other tools such as ip command, netstats and tcpdump as observer

In this case, the author uses page replicate the main page in http://ips.um.edu.my as material for website in nginx and each runs mget fetch the content as downloaded files at client side.

3.3.1 Designing Traffic Flow

From the packet traffic flow perspective, there are two types of traffics which handle any incoming packet (inbound) and outgoing packet (outbound) as illustrated in Figure 3-2. This to ensure that any incoming or outgoing are only filtered by specific states in the packet.

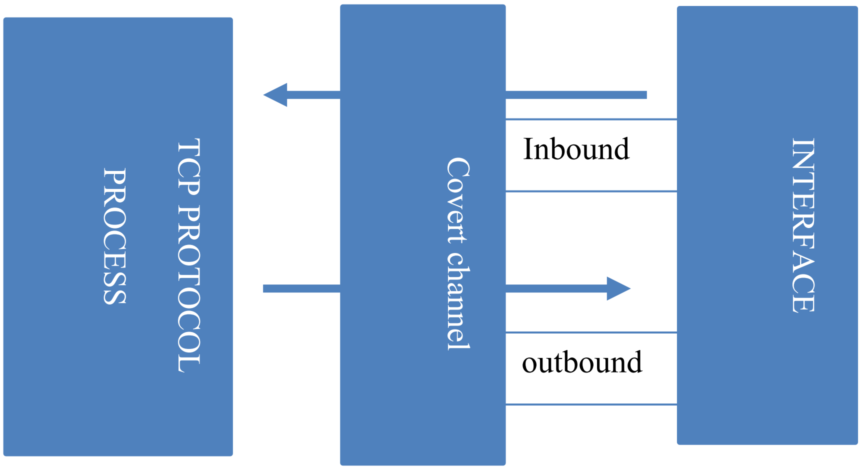

Figure 3-2: Traffic flow between interfaces

Moreover, by concentrating specific traffic, the modification of the packet can create minimum changes as the process should be fast to avoid anomaly pattern in latencies. Coincidentally, in practise packet modifications are common; since there are many implementations of modification in TCP/IP packet such as Network Address Translation (NAT), Internet Protocol Security (IPsec) and Virtual Private Network (VPN).

3.3.2 Designing Covert Tool.

To apply covert channel, preservation of TFO syntax is very important. Hence, bijective transformation is used to preserve any output/input data at data source level. To achieve this, covert channel is implemented as middlebox or man-in-middle that can be located at various places such as at host sender itself or at network service level such as firewall, router, etc. For message hiding in covert content, TFO cookie is replaced with encrypted 3DES message. A 3DES has 64 bits block size that perfectly match with the size of TFO cookie.

Only TCP packets that contain SYN, SYN-ACK with TCP option 34 are involved and taken out from packet. In outbound section, TFO cookies are replaced by 'XORed' 3DES encrypted message with Initialisation Vector (IV) as stated in Equation 3-1.

$$Covert_{message} = E_{3DES}(\text{message}) \oplus \text{IV} \quad \textbf{3-1}$$

Any key changes against TFO cookie are addressed by using different IV. On the other hand, in inbound section, contaminated TFO cookie is replace back with original TFO cookie and send back to sender (server). The Pseudocode 3-1 for covert channel is shown below:

```
OUTBOUND
Create IV[N];
For every SYN or SYN-ACK packet and TCP-OPTION== 34
N=0;
If length > 2
if connection not exist
Store TFO cookie
Replace TFO cookie with convert content = 3DES(message)  ⊕ IV[N];
Update checksum;
Else // If key changed
If connection is exit and previous cookie is not equal to current
cookie
Store TFO cookie
Replace cookie with convert content = 3DES(message)   ⊕ IV[N+1];
Update checksum;
End if
End if
End For
```

```
INBOUND
For every SYN or SYN-ACK packet and TCP-OPTION== 34
If length > 2
Replace cookie with TFO cookie
Update checksum;
```

Pseudocode 3-1: Covert Channel in TFO

Further, to extract message TFO cookie, simply XOR with shared IV and decrypt the output using 3DES as illustrated in Equation 3-2.

$$Message = D_{3DES}(Covert_{message} \oplus \text{IV}) \quad \textbf{3-2}$$

3.4 Testing Model

In this section, tests are designed to create proof against statement in Section 1.3. There are two types of tests namely, deliverable test & TFO behavioural test and performance test. Deliverable test & TFO behavioural test is to create proof that covert channel is workable and comply with preserving TFO syntax. Deliverable test is about get secret message via encrypted TFO cookie and behavioural test is about condition or situation that might occurred in TFO. In general, there are two situations that normal setting TFO can be fall under, first when TFO cookie is valid then proceed TFO transaction and lastly when TFO cookie is invalid resend new TFO cookie and proceed TFO transaction on next transaction. Thus, there are two main objectives that can be defined:

1. Successful retrieval covert content in covert channel mode.
2. No anomaly traffic in TFO transaction.

Meanwhile, performance test is to validate that implemented solution can get cope with semantic of TFO. Performance test is to observe implementation of covert channel that would impact to performance. The scope of performance test is to recreate situations that have similar concepts as explained in Section 3.3.

Moving forward, further analysis to resolve the following questions.

1. Performances relations between covert channel in TFO and normal TCP.
2. Performances relations between covert channel in TFO and normal TFO.

From there, in order to produce relationship from above questions, the null hypothesis can be created as follows:

- Hypothesis 1 (null hypothesis): There are no differences in performance between covert channel in TFO, normal TFO and TCP.
- Hypothesis 1 (alternative hypothesis): There are differences in performance between covert channel in TFO, normal TFO and TCP.

As general in order to support the study, the thesis should proof to accept alternatives hypothesis 1.

3.5 Summary

This chapter discussed methods that involving four stages namely, building covert communication model, create assumptions, prototyping and tests. Building covert communication model which apply the famous concept of the prison's problem with adaptation on TFO environment. Next, create assumptions explains that consideration that needed in order to get covert channel and TFO works, this includes consideration on both part either on covert channel or TFO limitation that can distract the whole process in this study.

On create prototype section there are two elements involved which are designing traffic flow and designing covert content. Traffic flow describes on how the data is divided into inbound and outbound and their purposes. While designing covert content involve how covert content can be created base on behaviour of TFO. In this part, explanation deeply and proposed pseudocodes are included. On the final section, testing model presents how it reflex to research problems and produce detail hypothesis.

CHAPTER 4: IMPLEMENTATION

This chapter discusses the implementation and its execution according to its main concept and ideas. It divided into two main group, creating tools and applying test. Creating tools explain how the tool can be deployed and it includes creating environment that use real situation application. Lastly, applying test explain the procedure to execute works.

4.1 Creating Tools

A. Groundwork Setup

In order to create groundwork setup, identified nodes are set with specific IP addresses. Further, to add supplementary realistic environment, the setup is divided into two network segments namely 192.168.57.0/24 and 192.168.56.0/24. A router acts as gateway to link both segments. In addition, the router also acts as traffic engineering tool which supplies intended network latencies. The full diagram is shown as in the Figure 4-1.

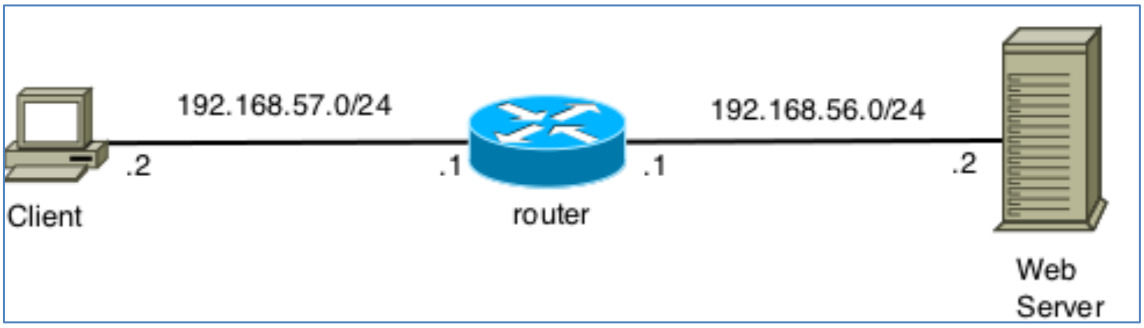

Figure 4-1: Network Diagram between web server and client

In order to reduce implementation complexity, web server, router and client are run in virtual environment and have identical specification as follows:

1. CPU: x86 4-core CPU 2.20GHz
2. Memory: 1024 MB
3. Drive Space: 30 GB
4. Operating System: CentOS 7.2
5. Kernel Version: 3.10.0

B. Building and execute tools

Since the server is the covert message source, the tool is run on the server side (web server). In this study, development tools language is chosen as follows:

- C/C++ with *libnetfilter_queue*, *libnftnl* to intercept and manipulate TCP packet
- C/C++ with openssl to encrypt and decrypt secret message.
- iptables utility to filter desire packet.

The main reason C/C++ is selected because it is the core language in user space in Linux environment. It also produces relatively high-performance on native application. *libnetfilter_queue* and *libnftnl* are libraries software used as packet filtering framework known as netfilter. The main functions of netfilter are to manage and administering queued network packet before or after reach Linux kernel. Netfiler also provides user space utility program such as *iptables* and *nftables* that give similar functions but with less flexibility compared to *libnetfilter_queue* and *libnftnl* in terms of manipulating packet. On the other hand, openssl is a software library that provide secure communication and encryption tools. It builds on C language and assembly language and runs on many platforms such as Unix-like platform and Microsoft windows.

For implementing the tools, the study uses iptables acts as middle-man between network interface and kernel in host. From here, iptables filters out TCP packets that contain SYN or SYN-ACK with TCP option 34 only to capture TFO cookies. On the inbound traffic, it captures SYN packet only that has option 34. While on the outbound traffic, it captures any packet that has option 34 since first SYN packet that carry TFO cookie also contains ACK packet from previous received packet from the client. Figure 4-2 showed a configuration of iptables and covert channel tool (NFQUEUE) that used to capture targeted packets. Note that INPUT indicates the inbound traffic while OUTPUT indicates the outbound traffic.

```
iptables -I INPUT -p tcp  --syn --tcp-option 34 -j NFQUEUE --queue-num 0
iptables -I OUTPUT  -p tcp  --tcp-option 34 -j NFQUEUE --queue-num 0
```

Figure 4-2: Example running iptables with Covert Channel tool

4.2 Applying Tests

Tests are divided base on covert message deliverable, syntax and semantic of TFO transaction. There are three set of tests as follows:

- TFO Deliverable Test - To ensure covert message is transferred successfully.
- TFO Behavioural Test - To test communication syntax in TFO. As discussed early, there are possibilities TFO cookie might change.
- TFO Performance Test – To ensure TFO objective or semantic of TFO is preserved.

The first two tests are categorised into correctness test due the tests are related to deliverable and syntax of the TFO transaction. While performance test is to assess the nature (semantic) of TFO transaction.

4.2.1 Deliverable Test and TFO Behavioural Test (Correctness Test)

In order to implement the tool, observation must be made on TFO traffic to create baseline and as comparison against covert channel mode. TFO traffic are captured and get through on two conditions namely, initial normal traffic TFO and changed TFO cookie key. The process of normal TFO with and without changed key comprises as follows:

1. When first TFO established
   - TFO sends request with no data in TFO payload.
   - Upon received, TFO payload filled with 64 bits TFO cookie.
2. Subsequence TFO connectivity
   - TFO sends request with 64 bits TFO cookie in TFO payload.
   - Upon received, no data in TFO payload TFO payload.
3. Subsequence TFO connectivity with different key at server
   - TFO sends request with 64 bits TFO cookie in TFO payload.
   - Upon received, TFO payload filled with new 64 bits TFO cookie.
   - For next subsequence TFO connectivity, it uses new 64 bits TFO cookie and proceed same as number two process.

The scenarios start with ordinary TFO traffic that involve normal transaction TFO starts with at first session client requests TFO cookie and server replies with TFO cookie, on second session client uses TFO cookie and proceed as normal transaction in TFO. On third session, unlike normal TFO traffic, the client uses expired TFO cookie and the server replies with new TFO cookie and proceed as normal TFO transaction. In terms of observations, all TCP states are recorded.

On covert channel part, identical situations are repeated but with running implemented covert channel tool. The procedure of running deliverable test & TFO behavioural test are shown below:

1. Run web server service (nginx)
2. Execute covert channel tool (if applicable).
3. Run mget client to fetch content from website.
4. Observe the output via netstat and ip command.

4.2.2 Performance Test

The tests are basically loop test performance between web client and web server. It uses use same setup as deliverable test & TFO behavioural test in order to keep persistency across the tests. In terms of creating comparisons, there are three modes which consist of normal TCP, normal TFO and covert channel TFO. Although the main target here is to find likeness between covert channel in TFO and normal TFO; normal TCP is also included as control element between two modes of TFO's.

To ensure all test are non-bias, the tests are conducted using the same nodes without modification according to the mode as mentioned before. In order to create high reliabilities, 99 loops test are conducted via running python script to create sampling. Moreover, each of tests are rebooted to ensure there are no caching process in operating system. Finally, each of completed tests are recorded for analysis. Below is the procedure when conducting the performance test:

1. All nodes (client, server and router) are rebooted when perform each cycle.
2. Run web server service (nginx).
3. Run performance script which loop mget client to fetch content from website.
4. Capture the output.

From the captured output, means or averages with difference percentages are collected. Further to proof performance are equal or unequal the thesis uses T-Test with actual population is unknown. The significance level α in this study is set to 0.05 and equal variance is assumed due to same data points in this study. The following are the elaborated hypothesis from 3.4 with means equations.

Hypothesis 1 (null hypothesis): There are no differences in performance between covert channel in TFO and TCP

$$\boldsymbol{H_0 = \mu_1 - \mu_2 = 0} \qquad \textbf{4-1}$$

Where:

μ_1 = TCP means

μ_2 = Covert channel TFO means

Hypothesis 1 (alternative hypothesis): There are differences in performance between covert channel in TFO and TCP

$$\boldsymbol{H_A = \mu_1 - \mu_2 \neq 0} \qquad \textbf{4-2}$$

Where:

μ_1 = Average of TCP

μ_2 = Covert channel TFO means

Hypothesis 2 (null hypothesis): There are no differences in performance between covert channel in TFO and TFO

$$\boldsymbol{H_0 = \mu_1 - \mu_2 = 0} \quad \textbf{4-3}$$

Where:

μ_1 = Average of TFO

μ_2 = Average of covert channel TFO

Hypothesis 2 (alternative hypothesis): There are differences in performance between covert channel in TFO and TFO

$$\boldsymbol{H_A = \mu_1 - \mu_2 \neq 0} \quad \textbf{4-4}$$

Where:

μ_1 = Average of TFO

μ_2 = Average of covert channel TFO

4.3 Summary

This chapter discussed the implementation and its execution according to its main concept and ideas. On the first section, it explains about creating tool comprises of background which includes network environment, nodes that involved web server, router and client. Then on specific part of building tool, programming languages that are used and some command utilities are explained. On applying test part, it consists of procedures that must be followed in order to capture the data. Command line utilities are used to endorse and confirm the output data. Further, statistical tests are deployed to confirm hypothesis.

CHAPTER 5: RESULTS AND DISCUSSIONS

There are two main output results that produced from implementation which consist of correctness test and performance test. Correctness test is to test covert content retrieval while observing any variations on TFO behaviours. Performance test is an observation of performance changes during covert channel implementation.

5.1 Correctness Test

Deliverable tests consist of capabilities test to retrieve (covert channel) without disturb or change TFO behaviours. The study makes used normal setting of TFO which applies two conditions as follows:

1. When received TFO cookie is not equal to previous, then resume as normal TCP.
2. When received TFO cookie is equal to previous, then starts TFO operations.

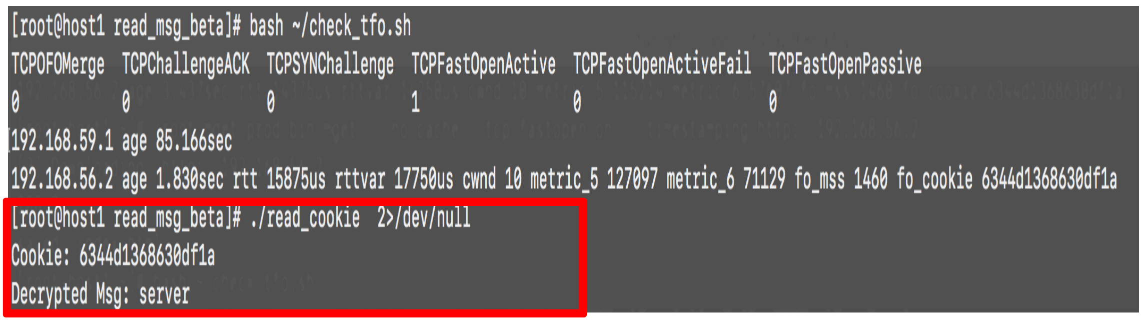

Figure 5-1: Client (host1) is successful retrieved message from web server (host0)

Detail results can be found in APPENDIX C where it showed the detail traffic and covert content retrieval in TFO. Only SYN_SENT, ESTABLISHED and SYN_RCVD states are involved where it targeted on TFO cookie movement. Meanwhile, on key changing part, it uses script key generator to test in four different situations which apply to covert channel TFO. However, only one-time key changed are observed since the key changing is not very common due to it occurs only on specific occasion such as system restart or intentionally setting. Figure 5-1 showed covert channel mode, message is successful retrieved and resume as normal TFO after the key is changed as in Figure 5-2 and Figure 5-3.

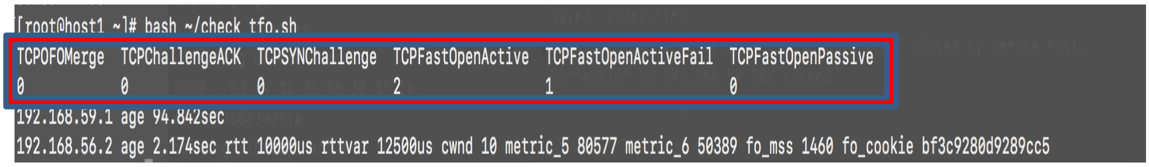

Figure 5-2: Covert Channel in TFO resumes normal TFO after TFO's key is changed

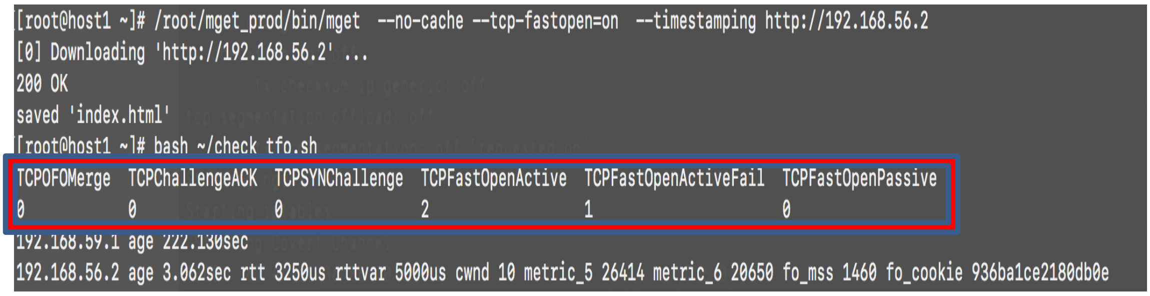

Figure 5-3: Normal TFO after TFO's key is changed

As overall, the results indicate that:

1. The message has transferred successfully.
2. No anomaly (key changing) is observed when applying covert channel in TFO.

5.2 Performance Test

The performance of covert channel in TFO is determined by comparing its means value against means of normal TCP and normal TFO. The setups are described as in 4.2 where it based on (Radhakrishnan et al., 2011) with different set of latencies and instruments. Each of tests are bounded by added network latencies namely, 0, 5, 10, 15 and 20 milliseconds (ms) and run into 99 times each. Table 5-1 showed the result of covert channel TFO performance against normal TCP and normal TFO.

Table 5-1: Averages of Covert Channel TFO performances against TCP & TFO

Modes		0 ms	5 ms	10 ms	15 ms	20 ms
TCP	Average (ms)	6.45733	22.99211	39.52326	53.88282	69.69715
	Std. Deviation	0.00918	0.00363	0.00642	0.00693	0.01336
TFO	Average(ms)	5.54876	18.05402	28.12153	38.51943	49.51002
	Std. Deviation	0.00493	0.00426	0.00464	0.00560	0.00819
	TCP (%)	14.07037	21.47734	28.84814	28.51260	28.96407
	P-value	0.19515	0.0000	0.0000	0.0000	0.0000
CC_TFO	Average(ms)	5.65176	18.75157	28.32758	38.82142	48.19316
	Std. Deviation	0.00698	0.01285	0.00389	0.00682	0.00824
	TCP (%)	12.47522	18.44346	28.32682	27.95214	30.85348
	P-value	0.24561	0.00088	0.0000	0.0000	0.0000

In this test, CC_TFO is used to represents Covert Channel TFO. From the results, at 0 ms, all performances have dissimilarity between 0.34% up to 14.07. Both CC_TFO and TFO indicate dissimilarity 12.47% and 14.07% respectively. However, although dissimilarities on both TFO indicated above 10%, but from hypothesis test perspective, there was no significant difference in the scores for TCP (Mean=6.45733, Standard Deviation=0.00918), TFO (Mean=5.54876, Standard Deviation =0.00493), CC_TFO (Mean=5.65176, Standard Deviation =0.00698) conditions; TFO p-value = 0.19515 and CC_TFO p-value = 0.24561. These results suggest that at 0 ms TFO and CC_TFO do not have an effect on performance compared to normal TCP. Specifically, the results suggest that at 0ms, all performances in the tests have no significant performances.

At 5 ms, all performances have dissimilarity between -4.35% up to 21.48% where CC_TFO and TFO indicate dissimilarity 18.44% and 21.48% respectively. Further it indicates, there were significant differences in the scores for both TFO (Mean=18.05402, Standard Deviation =0.00426) and CC_TFO (Mean=18.75157, Standard Deviation =0.01285) conditions; TFO p-value = 0.0000 and CC_TFO p-value = 0.00088. These results suggest that at 5 ms TFO and CC_TFO do have an effect on performance compared to normal TCP. Specifically, the results suggest that at 5 ms, both TFO and CC_TFO have strong significant performances.

At 10 ms, all performances have dissimilarity between -1.65 up to 28.85% where CC_TFO and TFO indicate dissimilarity 28.33% and 28.85% respectively. Further it indicates, there were significant differences in the scores for both TFO (Mean=28.12153, Standard Deviation =0.00464) and CC_TFO (Mean=28.32758, Standard Deviation =0.00389) conditions; TFO p-value = 0.0000 and CC_TFO p-value = 0.0000. These results suggest that at 10 ms TFO and CC_TFO do have an effect on performance compared to normal TCP. Specifically, the results suggest that at 10 ms, both TFO and CC_TFO have strong significant performances.

At 15 ms, all performances have dissimilarity between -8.82 up to 28.51% where CC_TFO and TFO indicate dissimilarity 27.95% and 28.51% respectively. Further it indicates, there were significant differences in the scores for both TFO (Mean=38.51943, Standard Deviation =0.00560 and CC_TFO (Mean=38.82142, Standard Deviation =0.00682) conditions; TFO p-value = 0.0000 and CC_TFO p-value = 0.0000. These results suggest that at 15 ms TFO and CC_TFO do have an effect on performance compared to normal TCP. Specifically, the results suggest that at 15 ms, both TFO and CC_TFO have strong significant performances.

At 20 ms, all performances have dissimilarity between -0.59 up to 30.85% where CC_TFO and TFO indicate dissimilarity 30.85% and 28.96% respectively. Further it indicates, there were significant differences in the scores for both TFO (Mean=49.51002, Standard Deviation =0.00819) and CC_TFO (Mean=48.19316, Standard Deviation

=0.00824) conditions; TFO p-value = 0.0000 and CC_TFO p-value = 0.0000. These results suggest that at 20 ms TFO and CC_TFO do have an effect on performance compared to normal TCP. Specifically, the results suggest that at 20 ms, both TFO and CC_TFO have strong significant performances.

Moreover, the study further up to determine either CC_TFO has significant performance differences against TFO. Table 5-2 showed an extension from Table 5-1 where the percentage differences and P-Values between TFO and CC_TFO are calculated.

Table 5-2: Covert Channel TFO performances against normal TFO

Modes		**0 ms**	**5 ms**	**10 ms**	**15 ms**	**20 ms**
TFO	Average(ms)	5.54876	18.05402	28.12153	38.51943	49.51002
	Std. Deviation	0.00493	0.00426	0.00464	0.00560	0.00819
CC_TFO	Average(ms)	5.65176	18.75157	28.32758	38.82142	48.19316
	Std. Deviation	0.00698	0.01285	0.00389	0.00682	0.00824
	TFO (%)	-1.85634	-3.86370	-0.73268	-0.78401	2.65979
	P-value	0.44882	0.30305	0.32859	0.33453	0.08390

The results indicate CC_TFO percentage values at 0 ms, 5 ms, 10 ms, 15 ms and 20 ms were -1.86%, -3.86%, -0.73%, -0.78% and 2.66%, respectively, against mean values of TFO. These relative small values are supported by hypothesis test that showed there were no significant differences in all scores for TFO (Latencies= 0ms, Mean=5.54876, Standard Deviation =0.00493), (Latencies= 5 ms, Mean=18.05402, Standard Deviation =0.00426), (Latencies= 10 ms, Mean=28.12153, Standard Deviation =0.00464), (Latencies = 15 ms, Mean=38.51943, Standard Deviation =0.00560) & (Latencies = 20ms, Mean=49.51002, Standard Deviation =0.00819) and CC_TFO (Latencies= 0 ms, Mean=5.65176, Standard Deviation=0.00698) conditions; CC_TFO p-value = 0.44882), CC_TFO (Latencies= 5 ms, Mean=18.75157, Standard Deviation=0.01285) conditions; CC_TFO p-value = 0.30305), CC_TFO (Latencies= 10 ms, Mean=28.32758, Standard Deviation=0.00389) conditions; CC_TFO p-value = 0.32859), CC_TFO (Latencies= 15 ms, Mean=38.82142, Standard Deviation=0.00682) conditions; CC_TFO p-value = 0.33453) & CC_TFO (Latencies= 20 ms, Mean=48.19316, Standard Deviation=0.00824) conditions; CC_TFO p-value = 0.08390) respectively. The full details are described in APPENDIX B.

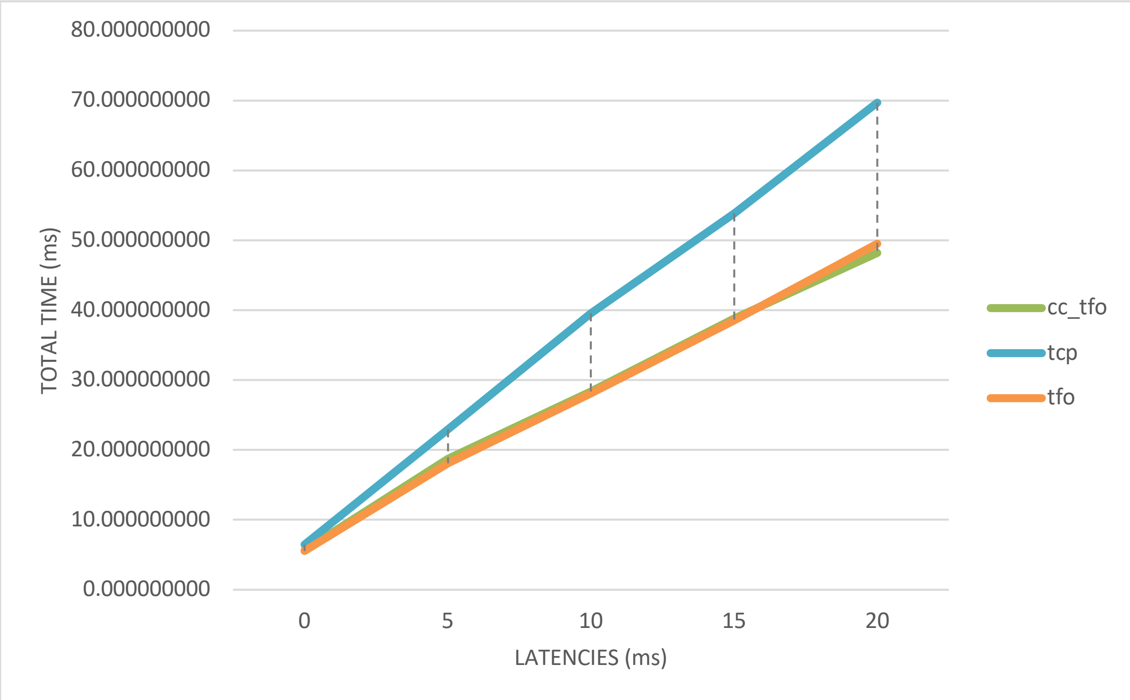

Figure 5-4: Performance of Covert channel in TFO tend to align with TFO

To illustrate trending, Figure 5-4 shows how network latencies patterns effect performances. The graph indicates when higher network latencies are applied, TFO-CC is aligned with TFO but not TCP. This showed there is only small overhead impact when implementing covert channels.

As overall, the results indicate that:

1. At 0 ms, TCP, CC_TFO and TFO have no significant performance dissimilarities.
2. Other than 0 ms, only TFO and CC_TFO showed significant performance dissimilarities against TCP.
3. For TFO and CC_TFO have no significant performance dissimilarities at all network latencies.

5.3 Discussion

The study found that covert contents are successful implemented in TFO. Packet from web server (sender) equipped with covert channel tool modifies TFO cookie to create covert content is successful transferred. On receiver part, the web client is successfully retrieved hidden information. Moreover, during the test, no errors or unsuccessfully transmission are found during the sessions.

Thus, this indicates both activities are aligned with the same procedures as normal TFO session that summarised in Table 5-3.

Table 5-3: Covert Channel in TFO overall results

Tools	Deliverable Test	Behavioural Test	Performance	Covert message.
Covert Channel	√	√	Similar as TFO	Retrieved

Further, four simulation performances are tested in different environments consist of normal TCP, normal TFO, TFO with covert channel. Certainly, implementation of covert channel gives no implication differences in terms of performance. Thus, overhead of creating covert channel is minimal.

Moreover, having these results and extension from Table 2-6. Covert channel in TFO provides one of the largest payloads as shown in Table 5-4.

Table 5-4: Comparison Covert Channel Payload Size

	TCP Fields	Papers/tool/Solution	Payload size	Type
1	ISN & ACK	Covert TCP	64	Storage
2	Urgent pointer	Hintz	16	Storage
3	Header checksum	Abad	16	Storage
4	ISN	NUSHU, Lantra	32	Storage
6	Reserved N packet	Allix	4	Storage
7	ACK	CLACK, ACKLeaks	32	Storage
8	Retransmission	RSTEG	Max IP segment	Storage
10	TCP Timestamp	Giffin et al.	1	Timing
11	Segment Reordering	Chakinala et al.	Log_2 n!	Timing
12	X TCP flows	Cloak	n	Timing
13	TCP Bursts	TCP scripts	n	Timing
14	Maximum Segment size (MSS) & ISN	Kumar et al.	n	Storage
15	Port	Efanov et al.	16	Storage
16	TFO	Covert Channel in TFO	64	Storage

Note. Information no. 1 to 13 from (Mileva & Panajotov, 2014), for 14 from (Kumar et al., 2011) and 15 from (Efanov et al., 2017).

5.4 Summary

This chapter discussed the output results of the implementation which consist of correctness test and performance test. All the features as described from previous chapter were implemented to collect the output results and suit the tests. Further, the collected data is discussed and described in the last section of the chapter. The analysis shows that the covert channel is successful implement in TFO and maintain the regular properties. The performance and behavioural of TFO with covert channel identically with the ordinary TFO.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Introduction

This chapter presents the conclusion of the study. It discusses the achievement of the study objectives, and the contributions made. There are also recommendations for future work to be considered.

6.2 Accomplishment of Objectives

A covert channel is one of the techniques to transfer message secretly without having to use ordinary procedures for data transferring. This study aims to introduce covert channel in TFO by hiding message in TFO cookie. Section 1.3 points out three objectives of this study. Thus, this section aims to answer the following questions:

Q1) Does covert channel in TFO can be implemented

Q2) Does the covert channel implementation is keep intact with the TFO objective which is performance for data transferring.

Objective 1: To report covert channel and implementation of TFO in practice.

This objective provides a clear understanding of TFO and covert channel fundamental. It explains preliminary parts that required in Q1 before covert channel implementation can be conducted. In order to achieve this objective, a detail discussion about TFO and covert channel in literature review is conducted, it also includes on how TFO works in practice by using Linux operating system as an example. Moreover, it also gives us information about of syntax and semantic of TFO that becomes the essential factor in building covert channel.

Objective 2: To implement covert channel in TFO

This objective aims to address Q1. It was initiated by creating covert communication concept that is illustrated in communication model. Then assumption factors must be included in order to create a scenario case. Further, the thesis uses prototyping approach to interpret scenario case. It consists of selecting suitable resources namely software, nodes, tools, along with traffic flow simulation and implementation of covert tool. The covert tool was tested and successfully send a hidden message between nodes.

Objective 3: To evaluate correctness and performance of covert channel covert channel in TFO

This objective aims to address Q2 by testing what have done in the previous objective. The tests consist of behavioural test and performance test that based on syntax and semantic in TFO communication. The Behavioural test is a test that when TFO key is changed and the effects against covert channel. Meanwhile, the performance test is a compression speed test against normal TFO. It uses T Test to measure the similarities. The findings of all tests confirmed that covert channel in TFO maintains as regular TFO properties.

6.3 Contributions

The introduction of covert channel in TFO creates new covert channel application in TCP. It can be recapped into three points.

1. New technique of creating covert channel.

This new approach in creating covert channel in TCP would widen on domain knowledge and create an alternative to existing approaches.

2. Efficient and Practicality in usage

The proposed implementation showed it aligned with TFO objective whereby performance and behavioural of TFO with covert channel confirmed to have identical results with ordinary TFO.

3. Offer large covert content payload

In this study, the usage of 64 bits covert payload always can be benefits in terms of transferring message.

6.4 Future Work

The proposed implementation and simulation are work only in IPv4 environment. Having latest trend different environment such as in IPv6, Zigbee and in Software-defined Networking (SDN) may create different scenarios and outcomes. Moreover, the equipment that used in this study are based on ordinary client server environment, thus the usage of Internet of Thing (IoT) or wireless sensor network devices should be considered in order to keep intact with latest real-approach application.

REFERENCES

Accenture. (2018). 2017 Cost of Cyber Crime Study. Retrieved 28 Aug, 2018 from https://www.accenture.com/my-en/insight-cost-of-cybercrime-2017

BIND. (2016). Release Notes for BIND Version 9.11.0. Retrieved 28 Aug, 2018, from https://ftp.isc.org/isc/bind9/9.11.0/RELEASE-NOTES-bind-9.11.0.html

Braden, R. (1994). T/TCP--TCP extensions for transactions functional specification (No. RFC 1644).

Betz, J., Westhoff, D., & Müller, G. (2017). Survey on covert channels in virtual machines and cloud computing. Transactions on Emerging Telecommunications Technologies, 28(6), e3134.

Cheng, Y., Chu, J., Radhakrishnan, S., & Jain, A. (2014). TCP fast open (No. RFC 7413).

Cisco System, Inc. (2017). The Zettabyte Era: Trends and Analysis. Retrieved 28 Aug, 2018, from https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/vni-hyperconnectivity-wp.html

Clark, W. K., & Levin, P. L. (2009). Securing the information highway. Foreign Aff., 88, 2.

Chrome. (2017). Chrome - Chrome Performance Tweaks and Text Enhancements. Retrieved 28 Aug, 2018 from https://wiki.mikejung.biz/Chrome#Enable_Chrome_TCP_Fast_Open_.28Linux_.2F_Android_Only.29

Duke, M., Braden, R., Eddy, W., Blanton, E., & Zimmermann, A. (2015). A roadmap for transmission control protocol (TCP) specification documents (No. RFC 7414).

Efanov, D., & Roschin, P. (2017). The Port-in-Use Covert Channel Attack. In First International Early Research Career Enhancement School on Biologically Inspired Cognitive Architectures (pp. 239-244). Springer, Cham.

Fisk, G., Fisk, M., Papadopoulos, C., & Neil, J. (2002). Eliminating Steganography in Internet Traffic with Active Wardens *Information Hiding* (pp. 18-35): Springer Berlin Heidelberg.

Ford, A., Raiciu, C., Handley, M., & Bonaventure, O. (2013). TCP extensions for multipath operation with multiple addresses (No. RFC 6824).

Gont, F., & Bellovin, S. (2012). Defending against sequence number attacks (No. RFC 6528).

Gray III, J. W. (1994). Countermeasures and tradeoffs for a class of covert timing channels. Hong Kong University of Science and Technology Technical report. Harris, J. (2016). Exim 4.88 released. Retrieved 28 Aug, 2018, from https://lists.exim.org/lurker/message/20161225.101705.4bbe7ae8.en.html

Huitema C. (2016). Building a faster and more secure web with TCP Fast Open, TLS False Start, and TLS 1.3. Retrieved https://blogs.windows.com/msedgedev/2016/06/15/building-a-faster-and-more-secure-web-with-tcp-fast-open-tls-false-start-and-tls-1-3/

Hussain, I., Negi, M. C., & Pandey, N. (2018). Security in ZigBee Using Steganography for IoT Communications. In System Performance and Management Analytics (pp. 217-227). Springer, Singapore.

IBM Knowledge Center. (2017). TCP Connection Status. Retrieved 28 Aug, 2018, from https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.1.0/com.ibm.zos.v2r1.halu101/constatus.htm

Juniper. (2017). Configuring TFO - TechLibrary - Juniper Networks . Retrieved 28 Aug, 2018, from https://www.juniper.net/documentation/en_US/junos/topics/task/configuration/tfo-configuring.html

Kay, J., Scott, K., Bridges, M., Knowles, S., Bellovin, S., Subramaniam, S., & Sukonnik, V. (2016). Transmission Control Protocol (TCP) Parameters. Retrieved from https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml

Kelsey, P. (2018). [base] Revision 330001. Retrieved from https://svnweb.freebsd.org/base?view=revision&revision=330001

Kerrisk, M. (2012). TCP Fast Open: expediting web services. Retrieved 28 Aug, 2018, from https://lwn.net/Articles/508865/

Kocher, P., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., ... & Yarom, Y. (2018). Spectre attacks: Exploiting speculative execution. arXiv preprint arXiv:1801.01203.

Kreibich, C., Handley, M., & Paxson, V. (2001). Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In Proc. USENIX Security Symposium (Vol. 2001)

Kumar, V. S., Dutta, T., Sur, A., & Nandi, S. (2011). Secure network steganographic scheme exploiting TCP sequence numbers. In International Conference on Network Security and Applications (pp. 281-291). Springer, Berlin, Heidelberg.

Lampson, B. W. (1973). A note on the confinement problem. *Communications of the ACM, 16*(10), 613-615. doi:10.1145/362375.362389

Latham, D. C. (1986). Department of defense trusted computer system evaluation criteria. Department of Defense. Retrieved 28 Aug, 2018 from http://www.iwar.org.uk/comsec/resources/standards/rainbow/5200.28-STD.html

Lewandowski, G., Lucena, N. B., & Chapin, S. J. (2006). Analyzing Network-Aware Active Wardens in IPv6 *Information Hiding* (pp. 58-77): Springer Berlin Heidelberg.

Lewandowski, G. (2011). Network-aware Active Wardens in IPv6 (Doctoral dissertation).

Linux Kernel Documentation. (2017). Retrieved 28 Aug, 2018 from https://www.kernel.org/doc/

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., ... & Yarom, Y. (2018, August). Meltdown: Reading Kernel Memory from User Space. In 27th {USENIX} Security Symposium ({USENIX} Security 18). USENIX} Association.

Martins, D., & Guyennet, H. (2010). Steganography in {MAC} Layers of 802.15. 4 Protocol for securing Wireless Sensor Networks. In IWNS 2010, 2nd IEEE Int. Workshop on Network Steganography.

Mazurczyk, W., Smolarczyk, M., & Szczypiorski, K. (2009). Hiding information in retransmissions. arXiv preprint arXiv:0905.0363.

Mileva, A., & Panajotov, B. (2014). Covert channels in TCP/IP protocol stack - extended version. *Open Computer Science, 4*(2). doi:10.2478/s13537-014-0205-6

Murray, D., Koziniec, T., Zander, S., Dixon, M., & Koutsakis, P. (2017). An analysis of changing enterprise network traffic characteristics. In Communications (APCC), 2017 23rd Asia-Pacific Conference on (pp. 1-6). IEEE.

Murdoch, S. J., & Lewis, S. (2005). Embedding covert channels into TCP/IP. In International Workshop on Information Hiding (pp. 247-261). Springer, Berlin, Heidelberg.

Nginx. (2016). Module ngx_http_core_module. Retrieved 28 Aug, 2018 from http://nginx.org/en/docs/http/ngx_http_core_module.html

OpenBSD. (2005). PF: Scrub (Packet Normalization). Retrieved 28 Aug, 2018, from http://ftp.tuwien.ac.at/.vhost/www.openbsd.org/www/faq/pf/scrub.html

Postel, J. (1981). TRANSMISSION CONTROL PROTOCOL. Retrieved 28 Aug, 2018, from https://tools.ietf.org/html/rfc793

Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., & Raghavan, B. (2011, December). TCP fast open. In Proceedings of the Seventh Conference on emerging Networking Experiments and Technologies (p. 21). ACM.

Simmons, G. J. (1984). The prisoners' problem and the subliminal channel. In Advances in Cryptology (pp. 51-67). Springer, Boston, MA.

Simmons, G. J. (1993). Subliminal communication is easy using the DSA. In Workshop on the Theory and Application of of Cryptographic Techniques (pp. 218-232). Springer, Berlin, Heidelberg.

Spring, N., Wetherall, D., & Ely, D. (2003). Robust explicit congestion notification (ECN) signaling with nonces (No. RFC 3540).

Stevens, W., Fenner, B., & Rudoff, A. (1999). UNIX network programming. Boston: Addison-Wesley/Prentice-Hall.

Wendzel, S., Zander, S., Fechner, B., & Herdin, C. (2015). Pattern-based survey and categorization of network covert channel techniques. ACM Computing Surveys (CSUR), 47(3), 50.

Phrack Magazine. (1998) T/TCP Vulnerabilities. Retrieved 28 Aug, 2018 from http://phrack.org/issues/53/6.html.

Prabhakar, L., & Cheshire, S. (2015) Your App and Next Generation Networks. Retrieved 28 Aug, 2018, from https://developer.apple.com/videos/play/wwdc2015/719/.

Tarreau, W,. (2016). HAProxy Configuration Manual. Retrieved 28 Aug, 2018 from http://cbonte.github.io/haproxy-dconv/configuration-1.5.html#5.1-tfo.

Touch, J., Mankin, A., & Bonica, R. (2010). The TCP authentication option (No. RFC 5925).

Tumoian, E., & Anikeev, M. (2005). Network based detection of passive covert channels in TCP/IP. In Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on (pp. 802-809). IEEE.

Vaughan-Nichols, S. J. (2012). Linux 3.7 arrives, ARM developers rejoice. Linux and Open Source. Retrieved 28 Aug, 2018 from https://www.zdnet.com/article/linux-3-7-arrives-arm-developers-rejoice/

Zawawi, M. N., Mahmod, R., Udzir, N., Ahmad, F., & Desa, J. M. (2012). *Active warden as the main hindrance for steganography information retrieval*. Paper presented at the 2012 International Conference on Information Retrieval & Knowledge Management. Retrieved 28 Aug, 2018 from http://dx.doi.org/10.1109/infrkm.2012.6204989

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

Zhai, J., Liu, G., & Dai, Y. (2010). *A Covert Channel Detection Algorithm Based on TCP Markov Model*. Paper presented at the 2010 International Conference on Multimedia Information Networking and Security. Retrieved 28 Aug, 2018 from http://dx.doi.org/10.1109/mines.2010.190