

**HOST-BASED BOTNET MITIGATION
METHODOLOGY**

BY

Ahmed Mansoor Ayedh Al-Ameri

A Thesis Presented to the
DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER NETWORKS

May, 2016

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **Ahmed Mansoor Ayedh Al-Ameri** under the direction his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN COMPUTER NETWORKS**.



Dr. Muhammad Y. Mahmoud
(Advisor)



Dr. Ahmad Almulhem
Department Chairman



Dr. Marwan H. Abu-Amara
(Member)



Dr. Salam A. Zummo
Dean of Graduate Studies



Dr. Yahya E. Osais
(Member)

22/1/17

Date

© Ahmed Mansoor AL-Ameri

2016

DEDICATION

I dedicate this work to all my family members, especially my parents, my wife and my children, Mansoor and Mohammed.

ACKNOWLEDGMENT

First and foremost, I am grateful to Allah, the Most Merciful and the Most Gracious, for his guidance and blessings without which this thesis would not have been possible. My greatest appreciation to my thesis advisor, Dr. Muhammad Y. Mahmoud for his guidance, patience, and valuable help to complete this thesis work. A special thanks to my committee members for their support and valuable criticism.

I would like to express my great thanks and appreciation to Hadhramout Establishment for Human Development for their support during my study.

TABLE OF CONTENTS

ACKNOWLEDGMENT	IV
TABLE OF CONTENTS	V
LIST OF FIGURES	X
THESIS ABSTRACT	XV
ملخص الرسالة	XVI
CHAPTER 1 INTRODUCTION	17
1.1 BOTNET LIFE CYCLE:.....	19
1.2 BOTNET ARCHITECTURE:	20
1.2.1 CENTRALIZED BOTNET ARCHITECTURE:.....	21
1.2.1.1 IRC BOTNETS:.....	22
1.2.1.2 HTTP BOTNET:	23
1.2.1.3 POP3 BOTNET:	23
1.2.2 DECENTRALIZED BOTNET ARCHITECTURE:	24
1.2.2.1 PEER-TO-PEER BOTNETS:.....	25
1.3 THESIS MOTIVATION:	25
1.4 THESIS OBJECTIVE:.....	26
1.5 THESIS ORGANIZATION:	26
CHAPTER 2 BOTNET DETECTION AND MITIGATION METHODOLOGIES	28
2.1 BOTNET DETECTION:	28

2.1.1	<i>Host-based Detection Techniques:</i>	29
2.1.2	<i>Network-Based Detection Techniques:</i>	33
2.2	BOTNET MITIGATION:	41
2.2.1	<i>Precautionary:</i>	41
2.2.2	<i>Corrective:</i>	41
CHAPTER 3 PROBLEM DESCRIPTION		43
3.1.	PROBLEM STATEMENT	43
3.2.	PROPOSED APPROACH AND RESEARCH METHODOLOGY	44
CHAPTER 4 TOOL DEVELOPMENT AND TESTING		46
4.1.	PCTOOL VS.1.0 DEVELOPMENT:	46
4.1.1.	<i>Common Experiments Setup</i>	48
4.1.2.	<i>Testing PCTool v 1.0</i>	49
4.1.3.	<i>Discussion:</i>	56
4.2.	PCTOOL VS.2.0 DEVELOPMENT:	59
4.2.1.	<i>Thread (1):</i>	59
4.2.2.	<i>Thread (2):</i>	59
4.2.3.	<i>Thread (3):</i>	60
4.2.4.	<i>Experiment Setup:</i>	62
4.2.5.	<i>Discussion:</i>	64
4.2.6.	<i>Data set:</i>	64
CHAPTER 5 RESULT AND DISCUSSION		65
5.1.	MALICIOUS - BENIGN PROCESS CLASSIFICATION:	67
5.2.	EVALUATION METRICS:	69

5.2.1. Receiver Operating Characteristic (ROC) curve:	69
5.2.2. Area under the Curve (AUC):.....	70
5.2.3. Confusion matrix:	70
5.3. RESULTS:	73
5.3.1. $\Delta T = 100-500$ Milliseconds:.....	73
5.3.2. $\Delta T = 600-1000$ Milliseconds:.....	79
5.3.3. $\Delta T = 1100-1500$ Milliseconds:.....	84
5.3.4. $\Delta T = 1600-2000$ Milliseconds:.....	89
5.3.5. $\Delta T = 2100-2500$ Milliseconds:.....	94
5.3.6. $\Delta T = 2600-3000$ Milliseconds:.....	99
5.3.7. $\Delta T = 3100-3500$ Milliseconds:.....	104
5.3.8. $\Delta T = 3600-4000$ Milliseconds:.....	109
5.3.9. $\Delta T = 4100-4500$ Milliseconds:.....	114
5.4. SUMMARY AND DISCUSSION:.....	119
CHAPTER 6 CONCLUSION	123
6.1. SUMMARY.....	123
6.2. FUTURE WORK	124
REFERENCES.....	125
VITAE	136

LIST OF TABLES

TABLE	Page
Table 1: A summary of all dissection points for the three experiment used PCTool v1.0.....	58
Table 2 : Show sample of features used for training	66
Table 3: A guide for classifying accuracy	70
Table 4: Contingency table show the arrangement of TP, FP, FN, and TN which are the output of the confusion matrix.....	70
Table 5: how to calculate TP, FP, FN, and TN	71
Table 6: How to calculate TPR, FPR, TNR, and FNR.....	72
Table 7: Summary of AUCs for Ten-Folds where $\Delta T = 100 - 500$	74
Table 8: TPR, FPR, FNR, and TNR where $\Delta T = 100-500$	75
Table 9: Summary of AUCs for Ten-Folds where $\Delta T = 600 - 1000$	79
Table 10: TPR, FPR, FNR, and TNR where $\Delta T = 600-1000$	80
Table 11: Summary of AUCs for Ten-Folds where $\Delta T = 1100 - 1500$	84
Table 12: TPR, FPR, FNR, and TNR where $\Delta T = 1100-1500$	85
Table 13: Summary of AUCs for Ten-Folds where $\Delta T = 1600 - 2000$	89
Table 14: TPR, FPR, FNR, and TNR where $\Delta T = 1600-2000$	90
Table 15: Summary of AUCs for Ten-Folds where $\Delta T = 2100 - 2500$	94
Table 16: TPR, FPR, FNR, and TNR where $\Delta T = 2100-2500$	95
Table 17: Summary of AUCs for Ten-Folds where $\Delta T = 2600 - 3000$	99

Table 18: TPR, FPR, FNR, and TNR where $\Delta T = 2600-3000$	100
Table 19: Summary of AUCs for Ten-Folds where $\Delta T = 3100 - 3500$	104
Table 20: TPR, FPR, FNR, and TNR where $\Delta T = 3100-3500$	105
Table 21: Summary of AUCs for Ten-Folds where $\Delta T = 3600 - 4000$	109
Table 22: TPR, FPR, FNR, and TNR where $\Delta T = 3600-4000$	110
Table 23: Summary of AUCs for Ten-Folds where $\Delta T = 4100 - 4500$	114
Table 24: TPR, FPR, FNR, and TNR where $\Delta T = 4100-4500$	115
Table 25: Summary of results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT vary from 100 milliseconds to 4500 milliseconds	120
Table 26: Results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT randomly selected (1000, 2000, 3000, and 4000)	121
Table 27: Results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT vary from 2500 milliseconds to 3300 milliseconds	122

LIST OF FIGURES

Figure	Page
Figure 1: Botnet Life Cycle.....	19
Figure 2 : Centralized Botnet Architecture	22
Figure 3 : Decentralized Botnet Architecture.....	24
Figure 4: Illustration of PCTool V 1.0	47
Figure 5: Packets sent and received by DDoser Process.....	51
Figure 6: Packets sent and received by Zues Process.....	53
Figure 7: Packets sent and Received by YassinOx Process	55
Figure 8: Illustration of PCTool V.2.0	61
Figure 9: Packets sent and received by YassinOx Process.....	63
Figure 10: Illustration of how data divided to K parts (K=10), each part. One part for testing and other 9 parts are for training. We run the model for k times. In each run, the testing part is changed.	68
Figure 11: ROC Curve when $\Delta T = 100-500$ Millisecond , and $K=1$	76
Figure 12: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=2$	76
Figure 13: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=3$	76
Figure 14: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=4$	76
Figure 15: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=5$	77
Figure 16: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=6$	77
Figure 17: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=7$	77
Figure 18: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=8$	77
Figure 19: ROC Curve when $\Delta T = 100-500$ Millisecond , and $K=9$	78

Figure 20: ROC Curve when $\Delta T = 100\text{-}500$ Millisecond, and $K=10$	78
Figure 21: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds , and $K=1$	81
Figure 22: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=2$	81
Figure 23: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=3$	81
Figure 24: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=4$	81
Figure 25: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=5$	82
Figure 26: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=6$	82
Figure 27: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=7$	82
Figure 28: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=8$	82
Figure 29: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=9$	83
Figure 30: ROC Curve when $\Delta T = 600\text{-}1000$ milliseconds, and $K=10$	83
Figure 31: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds , and $K=1$	86
Figure 32: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=2$	86
Figure 33: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=3$	86
Figure 34: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=4$	86
Figure 35: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=5$	87
Figure 36: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=6$	87
Figure 37: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=7$	87
Figure 38: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=8$	87
Figure 39: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=9$	88
Figure 40: ROC Curve when $\Delta T = 1100\text{-}1500$ milliseconds, and $K=10$	88
Figure 41: ROC Curve when $\Delta T = 1600\text{-}2000$ milliseconds , and $K=1$	91
Figure 42: ROC Curve when $\Delta T = 1600\text{-}2000$ milliseconds, and $K=2$	91
Figure 43: ROC Curve when $\Delta T = 1600\text{-}2000$ milliseconds, and $K=3$	91

Figure 44: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=4$	91
Figure 45: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=5$	92
Figure 46: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=6$	92
Figure 47: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=7$	92
Figure 48: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=8$	92
Figure 49: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=9$	93
Figure 50: ROC Curve when $\Delta T = 1600-2000$ milliseconds, and $K=10$	93
Figure 51: ROC Curve when $\Delta T = 2100-2500$ milliseconds , and $K=1$	96
Figure 52: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=2$	96
Figure 53: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=3$	96
Figure 54: ROC Curve when $\Delta T = 2100-2500$ milliseconds , and $K=4$	96
Figure 55: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=5$	97
Figure 56: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=6$	97
Figure 57: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=7$	97
Figure 58: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=8$	97
Figure 59: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=9$	98
Figure 60: ROC Curve when $\Delta T = 2100-2500$ milliseconds, and $K=10$	98
Figure 61: ROC Curve when $\Delta T = 2600-3000$ milliseconds , and $K=1$	101
Figure 62: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=2$	101
Figure 63: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=3$	101
Figure 64: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=4$	101
Figure 65: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=5$	102
Figure 66: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=6$	102
Figure 67: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=7$	102

Figure 68: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=8$	102
Figure 69: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=9$	103
Figure 70: ROC Curve when $\Delta T = 2600-3000$ milliseconds, and $K=10$	103
Figure 71: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=1$	106
Figure 72: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=2$	106
Figure 73: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=3$	106
Figure 74: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=4$	106
Figure 75: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=5$	107
Figure 76: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=6$	107
Figure 77: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=7$	107
Figure 78: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=8$	107
Figure 79: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=9$	108
Figure 80: ROC Curve when $\Delta T = 3100-3500$ milliseconds, and $K=10$	108
Figure 81: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=1$	111
Figure 82: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=2$	111
Figure 83: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=3$	111
Figure 84: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=4$	111
Figure 85: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=5$	112
Figure 86: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=6$	112
Figure 87: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=7$	112
Figure 88: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=8$	112
Figure 89: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=9$	113
Figure 90: ROC Curve when $\Delta T = 3600-4000$ milliseconds, and $K=10$	113
Figure 91: ROC Curve when $\Delta T = 4100-4500$ milliseconds, and $K=1$	116

Figure 92: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=2$	116
Figure 93: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=3$	116
Figure 94: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=4$	116
Figure 95: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=5$	117
Figure 96: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=6$	117
Figure 97: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=7$	117
Figure 98: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=8$	117
Figure 99: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=9$	118
Figure 100: ROC Curve when $\Delta T = 4100\text{-}4500$ milliseconds, and $K=10$	118

THESIS ABSTRACT

Name: Ahmed Mansoor Ayedh Al-Ameri

Title: Host-Based Botnet Mitigation Methodology

Major Field: Computer Networks

Date of Degree: May 2016

Botnets are state-of-the-art malware where most frauds and attacks activities on internet are carried out. Bot detection is an active research area in recent times. Most proposals are network based detection methods which have lots of drawbacks, where botnet traffics and attacks have negligible network visible effects [1], and any minor modification on botnet traffic such as command and control (C&C) encryption allows botnet to evade detection. This thesis introduce host-based botnet mitigation technique. Our approach is statistical anomaly based detection using machine learning classification technique. The mitigation methodology is by blocking bot process from sending/receiving network traffic to/from its botmaster. Our experiments are based on observing C&C traffic in local machine. Special tool used to monitor C&C communications. We named it, Packet Capturing Tool (PCTool). Collected data from PCTool are further processed to find statistical features of botnet C&C traffic. Then, Support Vector Machine (SVM) training using those features to classify malicious and benign process. We had 94.11% success detection with false positive rate 0.14 %.

ملخص الرسالة

الاسم: أحمد منصور عايض العامري

عنوان الرسالة: [اكتب عنوان الرسالة]

التخصص: شبكات حاسب آلي

تاريخ التخرج: مايو 2016

Botnet هي مجموعة من البرمجيات الخبيثة و التي تشكل شبكة عبر الانترنت ، حيث تتم معظم عمليات الاحتيال والأنشطة/الهجمات على شبكة الانترنت من خلالها. تحديد/كشف هذه البرمجيات الخبيثة هو مجال بحوث نشط في الآونة الأخيرة. معظم المقترحات/الحلول المقدمة قائمة على طرق كشف Botnet في الشبكات المحلية والتي لها الكثير من السلبيات. بحيث أن حركة تراسل Botnet على الشبكات مقارنة بالتطبيقات الأخرى تصعب ملاحظتها و تكاد لا تذكر، وأن أية تعديلات طفيفة على حركة تراسل Botnet مثل تشفير أوامر القيادة والسيطرة (C & C) يسمح لل Botnet التهرب من الكشف. هذه الأطروحة تقدم تقنية للكشف عن Botnet في الأجهزة المحلية. نهجنا هو الكشف على الشذوذ الإحصائي باستخدام تقنية التعلم الذاتي للآلة للقدرة على تصنيف الشذوذ من عدمه. منهجية التخفيف هي من خلال منع Botnet من إرسال / استقبال بيانات عن طريق الشبكة من / إلى Botmaster. تستند تجاربنا على مراقبة حركة ال C & C في الاجهزة المحلية. أداة خاصة تستخدم لمراقبة حركة تراسل/ إستقبال أوامر السيطرة و التحكم (C&C) أسميناها (PCTool). تتم معالجة البيانات التي تم جمعها من PCTool للعثور على المزيد من الميزات الإحصائية لحركة تراسل/إستقبال أوامر السيطرة و التحكم لل Botnet. ومن ثم يتم تدريب SVM باستخدام تلك الميزات لتصنيف البرمجيات الخبيثة عن الحميدة. كان لدينا معدل 94.11% نجاح و معدل إيجابية كاذبة يصل إلى 0.14%.

CHAPTER 1

INTRODUCTION

Botnet is a group of infected computers (bots) running software which connect to attacker (botmaster) through C&C channel to coordinate fraud and attack activities. Botnet is the largest security threat on internet due to the tremendous number of attacks which can be held through it such as distributed denial-of-service (DDOS), phishing, spamming, malware distributing, information stealing, etc.

Authors in [2] informed that about 70 percent of distributed spams worldwide come from botnets. Furthermore, a prediction says that 25 percent of computers connected to internet around the world are bots (around 150 million PCs) [3]. Bot-master sends commands to bots through C&C overlay network. As a result, many earlier botnets detections proposals especially network-based botnet detection techniques tried to exploit this ongoing C&C behavior [4].

Since, most infected machines around the world are personal computers (PCs) used by home users and are not part of local networks [5], we need a methodology that can easily detect bots on single PC. It is assumed on theory that bot program installed on end computer will have different outgoing and incoming traffic behavior than benign program [4]. There are many proposals used this assumption in their works.

Our approach is host-based mitigation technique independent of any structure/protocol with high detection rate reaches 94.11% and low false positive about 0.14%. Our

experimental approach is to find common anomaly on bots traffic behavior. Experiments are based on observing and comparing traffic behavior of known bots with traffic behavior of other benign programs. Packet Capturing Tool (PCT) was built for this purpose. The statistical analysis will held after capturing network packets of each running process on each machine. Then, statistics are used as main features to train the Support Vector Machine (**SVM**) classifier. Radial Basis Function (RBF) is used as kernel function of SVM. This way, our mitigation technique is based on detection methodology which is behavioral anomaly and statistical analysis.

1.1 BOTNET LIFE CYCLE:

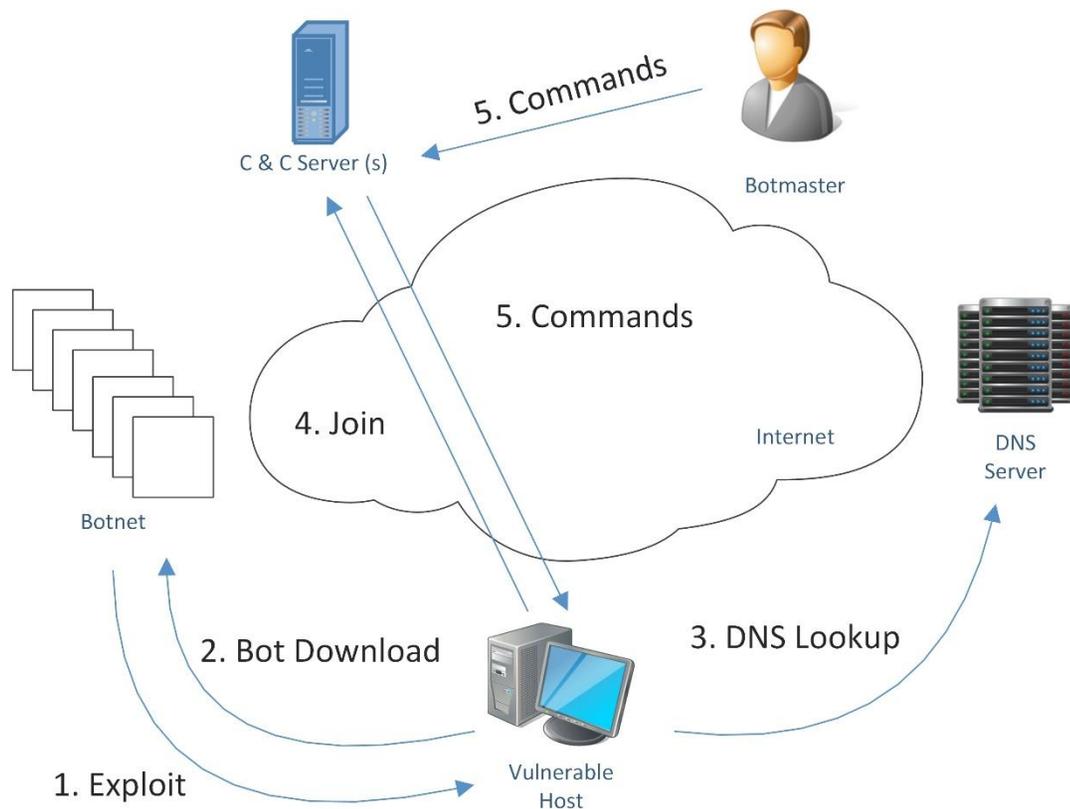


Figure 1: Botnet Life Cycle

Life cycle of botnet begins with infection stage as illustrated in figure 1 where vulnerable host is being exploited. When machine gets infected it called bot (zombie). After infection, bot binary is downloaded from remote server and automatically installed (step 2). Then, bot starts DNS lookups (step 3) in order to find C&C server(s) and authenticate itself as part of certain botnet. After joining to botnet (step 4), bot program can be

updated along with C&C servers list. Once infected machines (bots) are online and connected to C&C servers, they will be ready to perform botmaster's commands (step 5) as soon as they are issued.

1.2 BOTNET ARCHITECTURE:

Most botmasters are using command and control (C&C) channel to communicate with their bots. Actually, command and control (C&C) communication is the main attribute that distinguish botnets among other malicious software's[6]. C&C is a way that allows bot controller to command bot in his botnet to perform one of the previously mentioned activities of botnet (distributed denial-of-service (DDoS), phishing, spam, malware distribution, information theft, etc.). Botnets use various protocols for its command and control(C&C) communication mostly , IRC protocol , as well as , HTTP , or P2P protocols [7].

1.2.1 CENTRALIZED BOTNET ARCHITECTURE:

In this botnet architecture, C&C channels connect bots to certain C&C servers as illustrated in fig (2). Bots connection to C&C servers is done through various methods [7], namely :

- Hardcoded IP Addresses of C&C Servers:

The executable file which infects computers has in its binary, lists of IP addresses of C&C servers where bots should connect. This can be detected easily through reverse engineering [7] .

- Dynamic DNS:

According to botmaster's commands bots migrates from one C&C server to another using the public DNS service[7].

- Distributed DNS:

This method is same as the Dynamic DNS but they use their own distributed DNS service (away from law enforcement)[7].

Several protocols used in this architecture such as IRC, HTTP, and POP3.

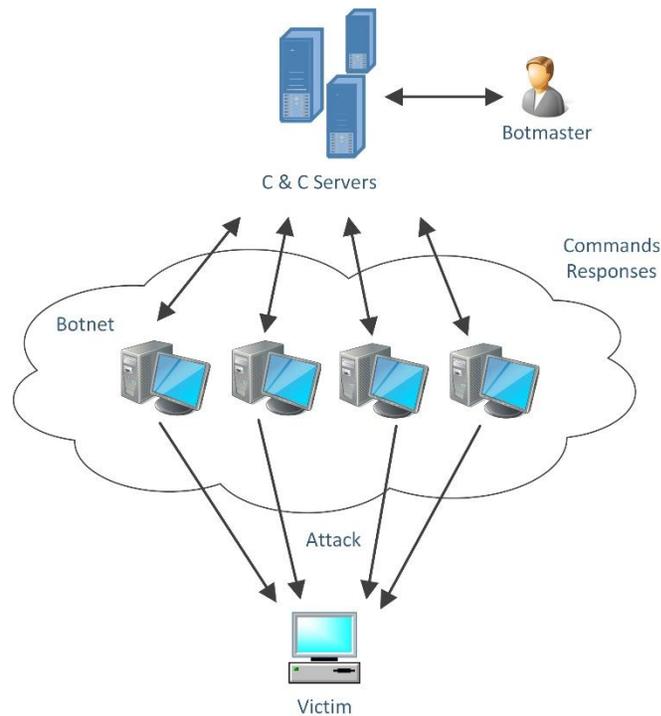


Figure 2 : Centralized Botnet Architecture

1.2.1.1 IRC BOTNETS:

IRC is very famous public exchange point. It enables virtually instant communication, which provides common, simple, low latency, wide availability and anonymity command and control protocol for bot communication. IRC network is composed of one or more IRC servers. According to botnet design, each bot connects to public IRC network or hidden IRC server. Bot receives commands from controller (bot-master) and can be instructed to attack. Simplicity and multicast delivery mechanism of IRC protocol fascinate attackers to use this protocol to send instructions and commands to bots. Wang

et al. [8] and Gizzard et al. [9] observed that most easily detected botnets use IRC for their C&C communication. They pointed out the weaknesses of IRC botnet because of its centralized server architecture. Moreover, IRC traffic is usually unencrypted and once the centralized IRC C&C channel is detected by defender the whole botnet could be disabled by shutting down central server.

1.2.1.2 HTTP BOTNET:

In this architecture, HTTP protocol is used by botnet to make its C&C communications. HTTP post and poll mechanism facilitates the work of botnets[10]. Bots use HTTP poll for several times to get bot-master commands[11]. In meantime, botmaster uses HTTP post to distribute commands then bots do polling from web server. In this way, it will be hard to detect due to difficulty to identify anomaly traffic from normal traffic, as well as, HTTP botnet traffic can easily pass through firewall policies[12]. But when HTTP server revealed, it will be easy to shut down the whole botnet[11].

1.2.1.3 POP3 BOTNET:

In this architecture, POP3 protocol is used for C&C communications. In POP3 botnet, bot is getting commands through retrieving E-Mail message from pre-defined mail server. Commands are within E-Mail attachment and responses to commands are following the same channel. This way, it will be harder to detect connection than in IRC server [13].

1.2.2 DECENTRALIZED BOTNET ARCHITECTURE:

In this botnet architecture, each bot can publish command as shown in figure 3. In this way, botnet we still be functional even after identification of bot-master and bring it down, since, any bot in the decentralized botnet still can issue commands[7]. Peer-to-Peer botnets is good example of decentralized botnets.

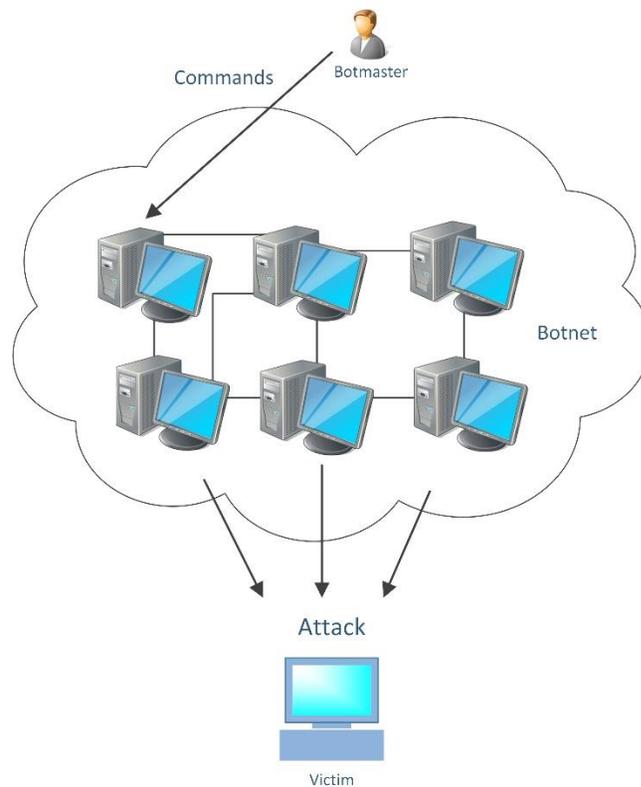


Figure 3 : Decentralized Botnet Architecture.

1.2.2.1 PEER-TO-PEER BOTNETS:

The publish/subscribe system is usually used in peer-to-peer (P2P) botnets for communications. Bot-master doesn't have ability to send commands to bots. Instead, he makes set of commands to P2P system and each bot subscribe to this set. When bot-master want to launch attack he publish command to the P2P system and subscribed bots to the set can receive it [11].

1.3 THESIS MOTIVATION:

Most infected machines around the world are personal computers (PCs) used by home users and are not part of local networks[5]. In addition, network based botnet detection techniques suffer from following drawbacks [14][1].

- Identifying the C&C traffic is very difficult task as C&C traffic is considered to be light compared to the legitimate traffic.
- Minor modification on botnet, such as, C&C traffic encryption, allows it to evade detection.
- Bot attacks have negligible network visible effects [1].
- High rate false positive detection leads to blocking on wrong hosts.

According to [1] lots of leading botnet detection methods [15][16][6][17] can be evaded using simple evasive tactic such as encrypting C&C traffic. Zhang et al. [18] concluded that, an effective botnet detection methodology should relay on behavior anomaly and statistical analysis.

This yields interesting insights to have Host-based botnet mitigation technique independent of any structure/protocol, rely on behavior anomaly and statistical analysis with high detection rate, low false positive rate, and not easily evadable.

1.4 THESIS OBJECTIVE:

Main objectives of this research are as follows:

1. Conduct research on botnet detection methodology and develop required techniques and tools.
2. Implement host-based botnet mitigation technique with high detection rate, low false positives and not be easily evadable.
3. Proposed technique should independent of any structure/protocol, rely on behavior anomaly and statistical analysis.

1.5 THESIS ORGANIZATION:

This thesis is organized as in following chapters. Chapter 2 starts with literature review of existing botnet detection techniques. Categorization of these techniques is established according to many characteristics such as topology of techniques and targeted component of botnet. Chapter 3 provides general description about research problem and how we are going to solve it. Chapter 4 discusses development and testing of the tool built to capture traffic of certain process. We tested this tool with real experiments, modification done to improve performance of the tool. After testing the tool, we built new dataset of malicious and benign processes network traffics. This dataset is used in experiments done in next chapters. Results of real experiments to classify malicious and benign application showed

in Chapter 5. Finally, thesis is concluded in chapter 6 in which overall research phases is summarized with suggestions for future work.

CHAPTER 2

BOTNET DETECTION AND MITIGATION METHODOLOGIES

In this chapter, we explore previous work done in literature in context of defense against botnet. The topic of defense against botnet has been extensively researched in recent years, and variety of proposed solutions were produced.

Defense against botnet can be broadly categorized to botnet detection and botnet mitigation. Accordingly, we classified available literatures to one of these categories.

2.1 BOTNET DETECTION:

We broadly categorize existing botnet detection techniques in the literature as Host Based detection, and Network-based detection. In botnet detection, there is no one technique to detect the whole botnet at once [19]. Targeted component of botnet in each category fluctuate. In Host-based, detection of malicious process is the ultimate goal. However, infected machine (**Bot**), C&C channel, and Botmaster are the targeted components in network-based detection techniques. Literatures actively detecting botnet are not reviewed in this work.

2.1.1 Host-based Detection Techniques:

In this section, we are going to review host-based botnet detection. Host-based botnet detection methods strategically monitor and analyze computer system activities (such as system logs, API calling, registry records etc.) internally instead of observing network traffic. Main target of these methods is to find the malicious process (**Bot process**) running in system.

L. Liu et al. [20] assumed that each bot has three main features. Firstly, no user intervention to initialize bot. Secondly, C&C communication should be established by bot. Finally, sooner or later bot should perform an attack. They proposed Bot-Tracer to identify three features with help of virtual machine (VM) techniques. This methodology is weak against the Zero day attack, and bots that check existence of the VM.

S. Balram and M. Wilson [14] have proposed host-based botnet detection methodology. Users tend to use certain common collections of application to perform normal daily tasks for their job, education, or even entertainment. So, normal usage of application will generate traffics that belong to certain pattern. The authors [14] in used this fact to build a profile for normal traffic. The profile is built through testing outgoing traffic of a user to a destination. The normal profile is initialized with a common destinations list such as www.google.com. Then, they add to list the tested flows to every destination. A flow means a group of network packets that have the same Destination Port, Destination IP, Source Port, Source IP, and Protocol. Flows to destinations are analyzed and destinations which exhibit normal behavior are added to the normal profile. Then, the profile is used to filter

host generated traffic into normal traffic and suspicious traffic. This traffic filtering stage will reduce the amount of traffic which needs to go to detailed Analysis by 70%. In the detailed analysis stage, suspicious traffics are classified to bot or normal. To classify the suspicious traffic as bot or normal, method involves analysis of traffic by looking for similar flows to a destination domain/IP at periodic intervals which indicate an existence of an active bot. Furthermore, periodic failed DNS queries to destination domain, many NetBIOS queries to a destination domain/IP, or appearance of many SYN scans in traffic indicates an inactive bot (a bot cannot connect to its C&C server). The authors claim of high botnet detection rate with low false positive using their real-time detection. However, the authors mentioned that their work does not consider IRC or P2P traffics which are widely used by bots as a C&C communication. Thus, this method is not a generic botnet detection method. Also, in their experiments, processing of traffic to build the normal profile is done on a timeslot basis. The time slot considered is only 30 minutes which causes the false positives.

E.Stinson and J. Mitchell [4] follow the hypothesis that each participating bot independently executes each command received over C&C network. A bot command takes some number of parameters (possibly zero) – each of a particular type – in some fixed order. Therefore, botnet constitutes remotely programmable platform with set of commands it supports forming its API. Many parameterized bot commands are implemented by invoking operating system services on host system. Typically, command's parameters provide information used in system call invocation. Thus, execution of many parameterized commands causes system call invocations on arguments obtained from those parameters. They characterize remote control behavior of bots via identifying when selected system call

arguments contain data received over network, this occurs when bot executes command received from its botmaster [4]. This approach performs explicit information flow tracking on network data. To distinguish remotely-initiated from locally initiated system calls, the method identifies data which is dependent upon local user input and sanitizes such data. This method has high false positives, and it degrades the performance of host.

In 2008, E.Stinson and J. Mitchell [1] proposed BotSwat, which is a tool to monitor operating system. Initially, BotSwat scan and monitor Win32 library execution status and runtime system calls. Despite botnet C&C architecture, protocols, and botnet structure, Botswat tries to find out bots with its general properties.

Masud et al. [21] used a flow based technique where log files of hosts correlated. This considered to be an effective host-based methodology. As bots normally respond more quickly than humans, mining and correlating multiple log files can be easily realized. It is proposed that these techniques can be efficiently performed for both IRC and non-IRC bots, by correlating several host-based log files for some C&C traffic detection.

In MABDS (Multi-agent bot detection system)[22], five agent where used, user agent, administrative agent, knowledge database (KnB), system analysis, and honeypot agents. The multi-agent technology uses a hybrid technique which combine HIDS (Host based IDS) and event log analyzer. Slow interchanging of new signatures with KnB, is the main issue for this technique.

Another HIDS was introduced by Ying et al. [23], but they used different techniques. HIDS combines back-propagation (BP) neural network and log analyzer. Detection was mainly based on misuse. Combination of the two techniques outperform current -at that time- other detection systems. But, major issue of this technique was scalability.

A host-based mechanism was proposed in [24]. This security mechanism can detect onset of malicious ware infection through the use of the right rules to apply the right dependency. Dependency in this context means, corresponding system event of user behavior. One issue of this tool is security, where OS routine can be intercepted by kernel mode routine.

HIIDS (Hybrid intelligent IDS) [25], applied neural network to mine the definition of malicious attacks. Collected information of certain attack behavior goes to decision support system. This combination of neural network and decision support system provide efficient and accurate detecting technique. But, does not provide autonomous learning in decision support system.

Proposals in [26][27][28], exploited static features of malicious code by extracting it by disassembly techniques. But this method is not effective anymore, due to emergence of packers, deformation, and polymorphic techniques.

API hooking technique and dynamic behavior analysis were exploited in [29][30][31][32] to overcome effect of the packers, deformation, and polymorphic techniques. Willems et al. [31] used API calling sequence for behavior analysis to detect malicious code.

The following are some intrusion detection methods which inspired our work away from botnet. Somayaji et al. [33], [34] proposed a kernel Linux extension (pH) which monitor any executing process and responding to abnormal behavior such as unexpected system-calls sequence. Once system has enough training on a sufficient number of normal program executions. The model will recognize most normal behavior as “normal” and most attacks as “abnormal”.

Microsoft Windows Registry is an essential part of the Windows Operating System , and it is significantly used by programs, which make it an excellent source of audit data [35]. Stolfo et al. [35] introduced a host-based Intrusion Detection System(IDS) to observe software usage of Windows Registry. They presented a Registry Anomaly Detection (RAD) system. RAD uses a machine learning model to learn normal model of accesses to registry in real-time and detects actions of malicious software access to registry at run-time.

2.1.2 Network-Based Detection Techniques:

In this section, we are going to review Network based Botnet Detection techniques. Unlike Host-based detection, Network-based Detection strategically monitor and analyze data while passing through medium. In network based detection techniques various parameters are being used to perceive malicious traffic through monitoring network traffic. Requests and responses of bots traffic tend to have same patterns despite architecture employed. In the network-based detection techniques components targeted are Bot (machine), C&C channel, and Botmaster. Accordingly, we categorize network based detection literature.

2.1.2.1 Bot Detection:

Bot detection is done without any regards to bot peers/family. This is very important to users and network administrators. As their main goal is protecting their computers and networks from getting exploited by botnet, without any concern about bot family. Bot detection designates existence of vulnerabilities in host or network to be exploited by botnet. Then, remedial strategies can take place to recover from infection and precaution to avoid the possibility of botnet infection in future.

Akiyama et al. [36] proposed the use of relationship, response, and synchronization as a three metric that bots within same botnet have regularities in. This metric is used for botnet detection. Relationship between bot-master and bots is one-to-many relationship. When bot-master publishes command each bot respond automatically with no mistakes in a way that can be different from human responses. Actions of bots after receiving bot-master commands are taken at the same time. This synchronization that just accrued can be used as botnet detection metric.

J. Binkley and S. Singh [5] proposed algorithm to detect anomaly traffic of IRC-based botnet. This algorithm can detect bots by combining TCP anomaly, and IRC messages statistics and tokenization, as well as, can find C&C server. One weakness of this algorithm that it is defeated by simple encoding of IRC commands. C. Mazzariello [37] introduced model of IRC user behavior in channel to differentiate between ordinary and botnet-related

activity. They used decision trees in experiments. Despite that the experiments were succeed, it was not clear if the success is due to the method or the dataset used.

Rishi [6] is an IRC based detection and it is pure syntactic(signature based). Snort [38] is a famous syntactic tool as well . Syntactic detection can easily be evaded by simple encryption and it cannot detect newly emerged botnets.

However, semantic is superior approach which involves correlation or behavior based techniques. In correlation, bot families can be exposed. The idea of correlation techniques based on that botnet is an arranged infrastructure, and thus clustering hosts with similar activates is applicable. Researchers used correlating email spams to detect similar botnets [39].

Correlation divided to Horizontal and vertical. Horizontal correlation concerned about similarities of communication and machines behavior. This way it can detect bots. BotMiner[40] lists bots in network by performing cross cluster correlation of Similar communication and patterns of malicious activities clusters. BotSniffer [17] detect bots by finding similarities of bot families C&C traffic in local network. The approach was basically built on observation that responses of bots in same botnet which are running same bot program will be same due to the pre-programmed C&C interaction to bot-master commands. This approach comes with very low false positive rate.

In the literature many proposed methods using similarities of traffic to detect bots [15] [41] [42]. Strayer et al. [15] examine the flow characteristics such as bandwidth, packet timing,

and duration searching for the existence of botnet C&C activity in order to detect botnets. Then, they separate the likely to be botnet traffic from the unlikely to be botnet traffic. The likely to be botnet traffic is grouped and correlated to find patterns of communications which suggest the activity of a botnet. A weakness of these methods that it need more than one bot from the same botnet in the network monitored to be able to detect the bot.

In vertical correlation, activities of one host are correlated and compared with bot behavior model. BotHunter [43] couple suspicious traffic with intrusion detection activity to announce successful bot detection. Bots can escape this method since it is capable of evade correlating timings of events to perform local attack.

In behavior analysis where botnet analyzed by observing the machine/traffic behavior. Behavior analysis in network-based detection technique is by network traffic tracking and accordingly classifying it into applications traffic, then analyze to find bot-like behavior. Classification of network application traffic is not an easy job and it can defeated by application using dynamic/random port numbers. Using payload to classify network application traffic can be defeated by encryption. Lu et al [44] found that 40 % of traffic cannot be classified to application traffic.

Literatures [45] [16] [6] proposed an IRC-based botnet detection by separating IRC and None-IRC traffic. Karasaridis et al. [16] Presented an approach of detecting and categorizing botnet. Their approach does not depend on information of certain application layer. It is performed on transport layer data using passive analysis. Therefore, it will be invisible from botnet operator. This algorithm come with very low false positive of about

2% and can detect encrypted botnet communications. P2P bot traffic can be separated from P2P normal traffic based on 1) P2P churn, 2) Flow characteristics, and 3) Human and machine behavior[42]. Bots communication in p2p botnet exploited in [46] for detection using likelihood ratio. Despite this technique is efficient, it experience high false positive rate.

Others literature detect botnet through analyzing , monitoring, and correlating DNSBL[47] , DDNS [48] , and search engine queries [49]. Yu F et al [50] proposed a detection technique based on search engine of query logs. Unfortunately, Diverse Search requests are remain undetectable.

In order to avoid detection of C&C server, the bot-masters usually duplicate and move the C&C server frequently. Therefore, in order for bots to locate the current C&C server, they use Dynamic DNS (DDNS) queries. David Dagon [51] introduced a way to detect C&C servers by identifying domain names with huge concentrated rate of DDNS queries. This approach comes with high false positive, because, a famous domain name with high DDNS query rates and short Time-To-Live DNS could be falsely detected as a C&C server. In the other hand, A. Schonewille and D.van Helmond [52] introduced an approach which is very effective, since it detect numerous suspicious domain names . This approach is based on observing the frequent anomalous DDNS responses which indicates that the DDNS query was sent to an unavailable domain name. This DDNS replies often correspond to a taken down botnet C&C servers. Therefore, PCs which sending such DDNS queries are probably bots. The false positive of this approach is very low according to [53].

2.1.2.2 C&C Channel Detection:

C&C Channel detection is very important to understand botnet behavior. Analysis C&C can be very beneficial to expose bots and C&C servers.

C&C detection can be Syntactic or Semantic. Syntactic C&C detection involves finding signature of C&C traffic and model it. Doing this manually is time-consuming and less reliable. Automated syntactic C&C detection proposed in the literatures [54] [55][56]. Semantic C&C detection involves heuristic to for associating specific behavior to C&C traffic. Semantic classified to Statistical, Correlation, and behavioral methods. Statistical methods involved in detecting C&C communication. Machine learning used to classify the C&C traffic. Various features used to train the classifier to detection the C&C traffic. In [57] machine learning used separate IRC and non-IRC traffic , then used again to detect C&C traffic within the IRC traffic. In correlation, the idea is similar patterns of network traffic could be a clue of existence of C&C traffic. Strayer et al. [41] used this method. They find bot-like traffic, then cluster traffic according to similarity in behavior, and characteristics to detect the C&C traffic. Gu et al. [40] combined this method with activity correlation, then they applied cross- cluster correlation to identify the botnet. In behavior detection of C&C traffic methods, the detection of C&C traffic done through observing the abnormal traffic or by comparing traffic with C&C traffic established behavioral model. Wurzinger et al.[55] Proposed an automatic system uses botnet running in a controlled environment to generate C&C models.

A classification and identification of P2P network traffic presented in [58] [59] , both techniques does not consider details of application layer, [59] is a transport layer identification which has unidirectional traces.

In 2009 Barsamian et al [60] proposed a framework where behavior of network traffic of an Ethernet network characterized. Their approach can detect changes in the botnet behavior and existing signatures. Based on a K-means approximation they provide a reliable method to detect periodic and synchronous behavior, but this technique can easily be evaded by using architecture for the C&C than the IRC. Liu et al [61], introduced a P2P botnet detection tool based on network stream analysis. Their method is grounded on three algorithms: (1) P2P node detection algorithm, (2) P2P node clustering algorithm, and (3) similarity detection algorithm. The lengthy process to identify botnets through network stream analysis discourages this technique to be implemented in real-time environments. Zhang et al. [62] proposed a detection technique for P2P botnets, using the statistical patterns of the network traffic. Based on the flow-clustering, they estimated the active time of bots. A statistical fingerprints of the P2P bots used in the detection algorithm. Communication patterns of P2P bots are random which consider main weakness of this method.

Iliofotou et al [63] used Traffic Dispersion Graphs to monitor networks and measure its hosts social interaction. But this restricted to only support access layer. François *et al.* [64] observe the behavior of communication and analyze the patterns. Jiang *et al.* [65] monitor DNS failure through a graph analysis approach. Weaknesses of these approaches that they

are non-scalable and the DNS failure traces are unpredictable. Ha *et al.* [66] made a test bed for P2P botnet detection and mitigation, they only focused on P2P structure botnet.

2.1.2.3 Bot-Master Detection:

There are four main hurdles that limit bot-master traceback researches. These four hurdles are a) C&C traffic between bot and bot-master considered to be low. b) Bot-master launder his connections by placing several logical security systems, used as authentication servers or what so-called “stepping stones”. c) Usage of encryption along with stepping stones. d) Low-volume of C&C traffic from the bot-master [67].

Communication between bots and bot-master of IRC-based botnet is bidirectional Ramsbrock *et al.*[67] exploited this fact and proposed an approach of watermarking responses from bots to bot-maser, and thus, they can eventually trace and locate bot-master. This approach overcome all aforementioned hurdles. This is an active detection technique which mean more overhead. Recently, bot-master use newer techniques to escape it.

Z. Chi and Z. Zhao [68] said when attack command received by bots . Bots start attacking victim simultaneously. They proposed an approach of identifying bot-master during attacking the victim. Tracking back process is starting from victim reversing the path through routers. During identifying bot-master, malicious traffic blocked by routers. This algorithm has low false negative rate and low computation penalty. However, it still evadable by encryption and it is only for IRC-based botnets.

Bot-master usage of stepping stones technique to hide his trails recursively exploited by few proposals. However, such methods could be easily evaded by delaying packet or by adding meaningless packets (chaff). Zhang et al. [69] used packet timing and size to detect encrypted stepping-stone connection. Inter-packet timing correlation used by Wang et al. [70] to trace encrypted/unencrypted stepping-stone connection. Literatures [71][72] used timing to detect stepping-stone connection.

2.2 BOTNET MITIGATION:

Mitigating thread of botnet is the ultimate purpose of defending against it. Botnet mitigation can be before infection (precautionary) or after infection (Corrective).

2.2.1 Precautionary:

Precautionary means, precaution for the possibility of botnet infection. This might be technical or non-technical. Technical by assuring hosts and network cleanliness. Cleanliness can be assured by deploying OS updating, strict policies, and multi-level security system. Non-technical such as attacker dissuasion (through suppression financial motivation), user awareness (education), and legal accountability.

2.2.2 Corrective:

Corrective means, botnet infection recovery. Corrective can be defensive (Infection removal), or offensive (botnet destruction).

Defensive corrective for hosts can either by disinfecting host using off-the-shelf software, or OS reinstallation which is much guaranteed. In network performing defensive corrective is either by Bot-Block or C&C-Block.

Offensive corrective performed by either indirect attack or direct attack on botnet.

- Indirect attack involves reducing the usability of botnet by providing fake information (Passwords, bank credentials) to harm the credibility of bot-master (no third party to pay).
- Direct attack involves attacking one component of a botnet (Bot, C&C channel, C&C server). Botnet suppression and Botnet take over are two methods for direct attack. Botnet suppression involves bots removal and C&C channel interference. Wang et al. [73] proposed a technique to target bots and remove them. Two main techniques used in channel interference: index poisoning technique [74], where command faked and injected by defender in the P2P network , and Sybil attack technique[75], where fake route message distributed through fake Sybille in the botnet . Botnet Take Over, no detailed study to show how to take over P2P botnet[18].

CHAPTER 3

PROBLEM DESCRIPTION

In this chapter we state the problem to be addressed within this thesis and we discuss the methodology that will be followed to achieve proposed work.

3.1. PROBLEM STATEMENT

As explored in literature, most of existing botnet detection technique cannot be easily deployed commercially [5]. Detecting if host is bot or not, is not an easy task [5]. Most of proposed detection techniques are network based. While, most infected machines around the world are personal computers (PCs) used by home users and are not part of local networks[5]. In addition, network based botnet detection techniques suffer from the following drawbacks [14][1].

- Identifying the C&C traffic is very difficult task as C&C traffic is considered to be light compared to the legitimate traffic.
- Minor modification on botnet, such as, C&C traffic encryption, allows it to evade detection.
- Bot attacks have negligible network visible effects [1].
- High rate false positive detection leads to blocking on wrong hosts.

According to [1] lots of leading botnet detection methods [15][16][6][17] can be evaded using simple evasive tactic such as encrypting C&C traffic. Zhang et al. [18] concluded that, an effective botnet detection methodology should relay on behavior anomaly and statistical analysis.

This yields interesting insights to have Host-based botnet mitigation technique independent of any structure/protocol, relay on behavior anomaly and statistical analysis with high detection rate, low false positive rate, and not easily evadable.

3.2. PROPOSED APPROACH AND RESEARCH METHODOLOGY

The focus of this thesis is to develop Host-based botnet mitigation technique. We built our assumption on the theory that bot program installed on an end computer will have different traffic behavior than benign program [4].

An experimental approach is used to find patterns on bots traffic. Experiments are based on observing and comparing traffic behavior of known bots, with traffic behavior of other benign programs. Firstly, we built a tool (PCTool) to capture each outgoing or incoming packets to computer and associate captured packet to process. This tool is to observe behavior of bot traffics in host based manner. Secondly, bot processes and benign processes will be executed on hosts. Captured traffic by PCTool from each computer for bot/benign processes will be analyzed. We rely on statistical method to extract statistics out of captured packet to determine behavior of malicious or benign processes. Statistics such as, Number of Packets Sent, Number of Packets Received, Number of IPs, Number of Local ports, and

Number of Remote ports through which we can distinguish behavior of bots and benign processes. Then, we use these statistics as main features (or factors) of classification model by building SVM classifier. Finally, classification result will be presented and discussed to determine worthiness of our approach.

.

CHAPTER 4

TOOL DEVELOPMENT AND TESTING

An experimental approach is used to find patterns of bots traffic. Our experiments are based on observing and comparing traffic behavior of known bots, with traffic behavior of other benign programs. Firstly, we built Packet Capturing Tool (PCTool) to capture each outgoing or incoming packets and associate captured packet to process. This tool is to observe behavior of bot traffics in host based manner. Secondly, we built an isolated network in which each host in this network is infected (is a bot) and has PCTool.

4.1. PCTOOL VS.1.0 DEVELOPMENT:

We developed PCTool v.1.0 on Microsoft.net environment using C#. In order to capture each outgoing or incoming packets, we used SharpPcap Driver. SharpPcap is .NET assembly (library) for interfacing with Libpcap or Winpcap from .NET application. For Packet-To-Process association, we used “netstat -a -o -n” to retrieve used ports by running processes in system. We wrote class especially to associate retrieved Process IDs (PIDs) and Port Numbers from Netstat command to Process Names. Figure 4 illustrate how PCTool vs.1.0 works.

PCTool V .1.0

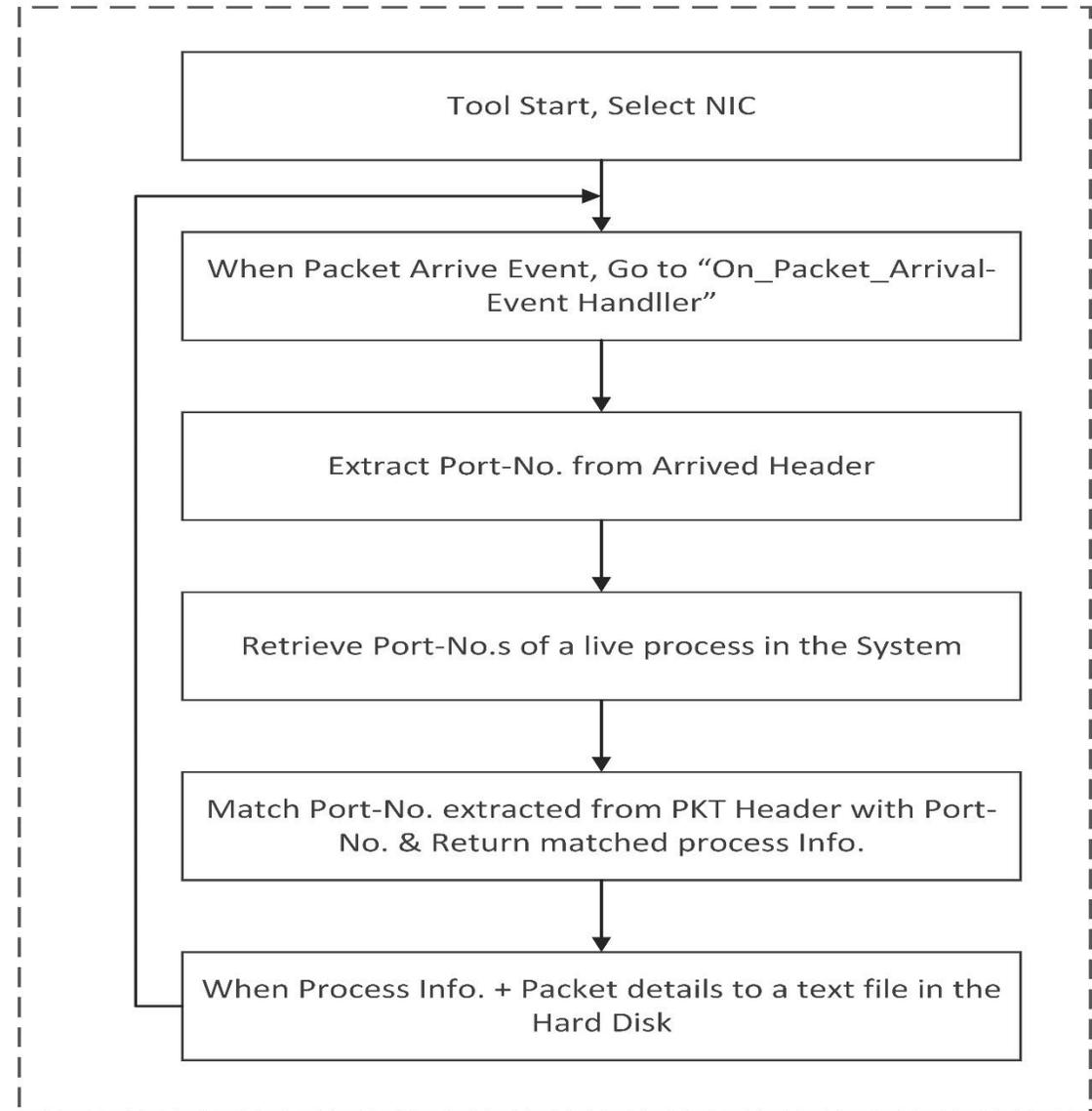


Figure 4: Illustration of PCTool V 1.0

4.1.1. Common Experiments Setup

Experiment were conducted to test the ability of PCTool v 1.0 and PCTool v 2.0 to associate traffics to malicious processes during monitoring time. We run bots in more than one PC just to avoid the abnormal behavior and to ensure that no misconfiguration in single PC. Thus, number of PCs used in experiments is not affecting results but give us insurance that nothing went wrong during experiments.

We built an isolated network from outsider world, in which each host in this network is infected (is a bot). Three botnets (DDoSeR 3.0 Mod, Zeus botnet, and YassinOX botnet) where used. These botnet are the only we could get, configure, and install.

DDoSeR 3.0 Mod (Botnet):

This is a botnet tool which produces a .exe file. When .exe file executed on any other machine it contact server and wait command from master. The master commands, for example, to flood certain machine from all of its clients with certain request message.

Zeus botnet:

XAMPP were used as web server in experiment. Botmaster can control bots through accessing this web server and browse bots in his botnet. Zeus produces an .exe file and when this .exe file executed on any other machine it contact server and wait command from botmaster.

YassinOX botnet:

Apache Server where used as web server in experiments. Botmaster can control bots through accessing this web server and browse bots in his botnet. YassinOX produces an .exe file and when this .exe file runs on any other machine it becomes zombie. This tool provides option of how bots distributed and how it receive commands.

4.1.2. Testing PCTool v 1.0

We conducted three experiments to test this version of PCTool. We separately deployed three real botnets in isolated environment. These experiments, are to see ability of PCTool vs.1.0 in capturing and associating traffic of both malicious and benign processes. We run bots in more than one PC just to avoid abnormal behavior or misconfiguration in single PC. Thus, number of PCs used in experiments is not affecting results but give us insurance that nothing went wrong during experiments.

We used additional tool we designed to reveal processes in these experiments:

- **Process Reveal Tool:**

We designed this tool to show all processes currently running on a given machine and store this information into text file.

Purpose of using it is to compare list of processes in given machine before and after it got infected with bot.

4.1.2.1. First Experiment Setup:

We installed Packet Capturing Tool and Process Reveal Tool on all machines (7 PCs). DDoSeR 3.0 Mod was installed on PC-1 (IP 192.168.2.1) where .EXE (bot) file built. EXE file executed on remaining machines (PC-2 ... PC-5, PC-7, and PC-8).

IP addresses of PCs as following:

PC-1 / 192.168.2.1

PC-2 / 192.168.2.2

PC-3 / 192.168.2.3

PC-4 / 192.168.2.4

PC-5 / 192.168.2.5

PC-6 / 192.168.2.6

PC-7 / 192.168.2.7

PC-8 / 192.168.2.8

Last digit of IP is same as in PC name. Figure 5 shows behavior of bot in experiment 1. In addition, it give us clear view about the ability of PCTool v1.0 to associate traffic packets to initiated or received process which in this experiment a process of DDOSER .3.0 mod. The x-axis of figures representing number of packets associated to process of DDOSER, and the y-axis is representing monitoring time. We found all figures from all PCs are alike, so we are adding only a figure below.

From figures below, we can say that PCTool v1.0 is being able to associate traffic during all the monitoring time or it failed to do. Conclusion and discussion of this experiment in section 4.1.3.

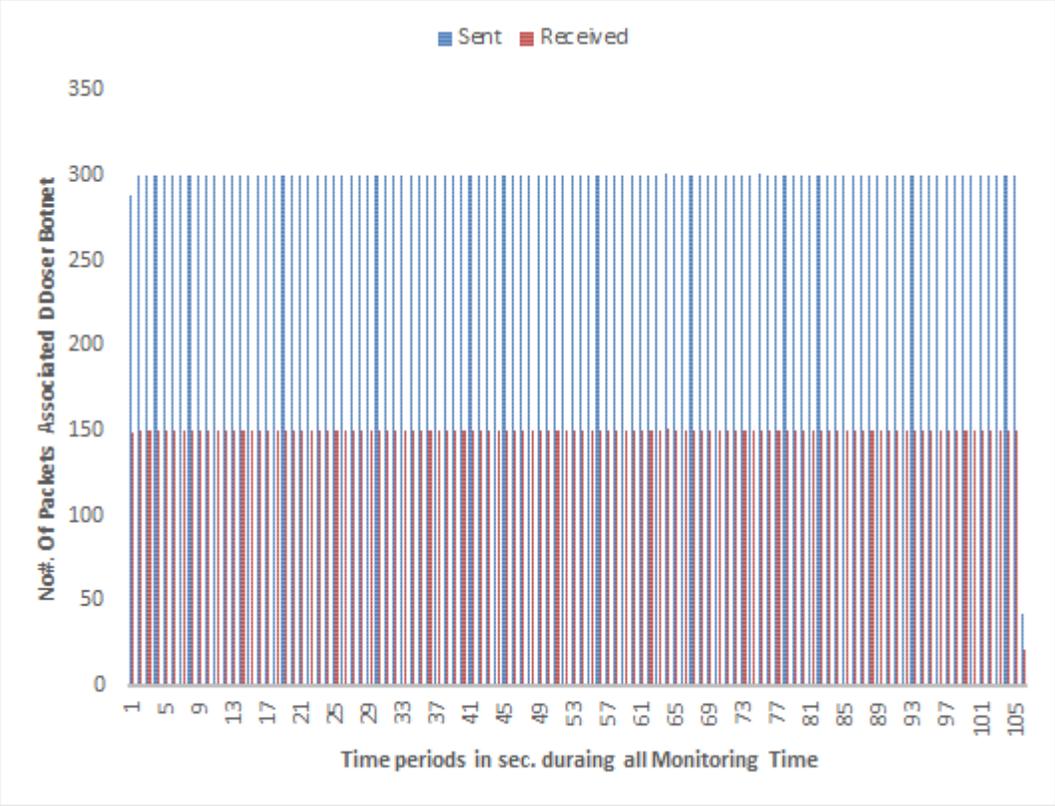


Figure 5: Packets sent and received by DDoser Process

4.1.2.2. Second Experiment Setup:

We installed Packet Capturing Tool and Process Reveal Tool on all machine (6 PCs). XAMPP and Zeus Bot-master was installed and Configured on PC-3 (172.16.0.3) where .EXE file built. EXE file executed on remaining machines (PC-2, PC-4, PC-5, PC-6, and PC-8).

IP addresses of PCs as following:

PC-2 / 172.16.0.2

PC-3 / 172.16.0.3

PC-4 / 172.16.0.4

PC-5 / 172.16.0.5

PC-6 / 172.16.0.6

PC-7 / 172.16.0.7

PC-8 / 172.16.0.8

Last digit of IP is same as in PC name. Figure 6 shows behavior of bot in experiment 2. In addition, it give us clear view about the ability of PCTool v1.0 to associate traffic packets to initiated or received process which in this experiment process of Zeus Botnet. The x-axis of figures representing number of packets associated to process, and the y-axis is representing monitoring time. We found all figures from all PCs are alike, so we are adding only a figure below.

From figure below, we can say that PCTool v1.0 is being able to associate traffic during all the monitoring time or it fail to do. Conclusion and discussion of this experiment in section 4.1.3

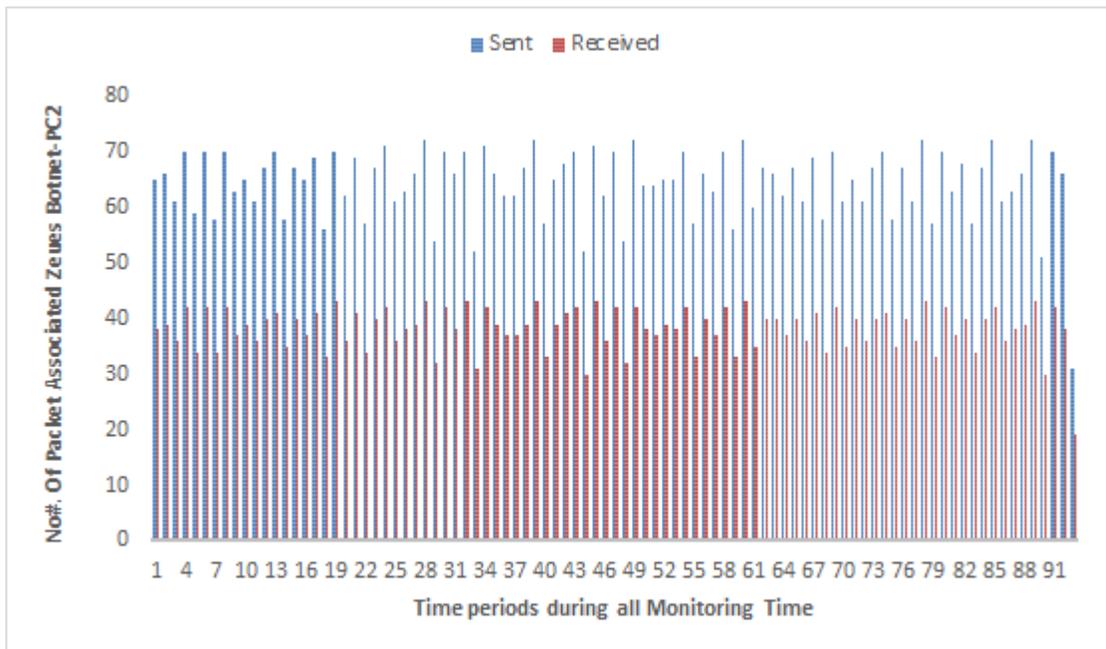


Figure 6: Packets sent and received by Zues Process

4.1.2.3. Third Experiment Setup:

We installed Packet Capturing Tool and Process Reveal Tool on all machine (7 PCs). Apache Server and YassinOX Bot-master was installed and Configured on PC-4 (172.16.0.4) where .EXE file built. EXE file executed on remaining machines (PC-1, PC-2, PC-3, PC-5, PC-6, PC-7, and PC-8).

IP addresses of PCs as following:

PC-1 / 172.16.0.1

PC-2 / 172.16.0.2

PC-3 / 172.16.0.3

PC-4 / 172.16.0.4

PC-5 / 172.16.0.5

PC-6 / 172.16.0.6

PC-7 / 172.16.0.7

PC-8 / 172.16.0.8

Last digit of IPs are same as in PC name. Figure 13 shows behavior of bot in experiment 3. Notice, we just draw graphs for one PC (PC-1). PCTool vs.1.0 could not associate packets to bot process. This issue detailed in discussion (section 4.1.3) of PCTool vs.1.0. Conclusion and discussion of this experiment in section 4.1.3.

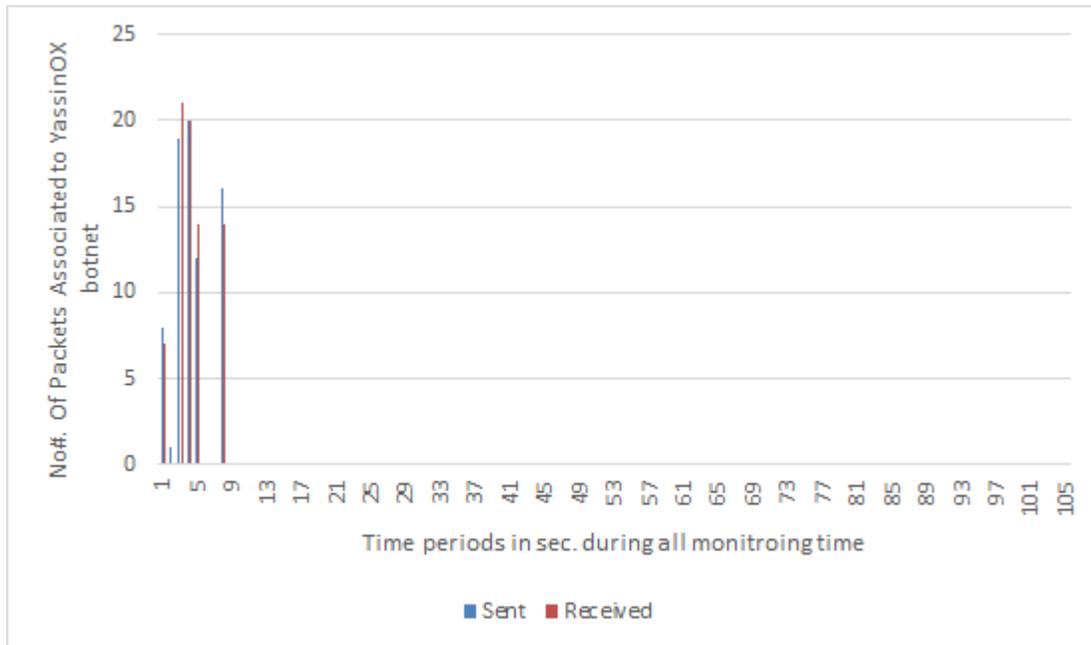


Figure 7: Packets sent and Received by YassinOx Process

4.1.3. Discussion:

As figure 4 shows how PCTool vs.1.0 works, and after we did deeper analysis, tool failed to associate captured packet to a live process in certain situations such as:

1. If malicious process keeps changing its port number frequently. Where, PCTool vs.1.0 extract port numbers from header of packet arrived, and match it with retrieved port numbers of alive processes in system, but, retrieved port numbers might be expired.
2. If inter-arrival speed of packets is higher than processing speed. Where, PCTool vs.1.0 takes long time to write processed data (Process ID, Process Name, Port No., and Packet details) to text file in hard disk.

In following, we will explain and discuss why outcome of the three experiments was as in figures 5-13.

4.1.3.1. First Experiment:

Despite drawbacks of PCTool vs.1.0, we did not notice or record any failure in this experiment run. We believe this is due to following:

- Botnet used has very low packet inter-arrival rate. It only sends two packets and receives one packet every two seconds. Thus, procedures of packet-to-process associating, and writing process-packet details into text file in hard disk are higher rate than packet inter-arrival rate.
- Bot process is not changing its port number at any time during run of the experiment.

This can explain why figures 5-9 show good performance where, all packets associated to their processes.

4.1.3.2. Second experiment:

In this experiment we slightly noticed some of drawbacks of PCTool vs.1.0. Though, it did not affects results and remained acceptable. This could be Due to following reasons:

- Zeus botnet has higher packets inter-arrival rate than botnet used in first experiment. Yet, PCTool vs.1.0 can associate 75% of packets. See figures 10-12.
- Zeus bot process also does not change its port number at any time during run of the experiment.

This can explain why the figures 10-12 show an acceptable detection of behavior of Zeus bot since about 75% of packets are associated to their processes.

4.1.3.3. Third experiment:

In this experiment, we noticed a poor performance of PCTool vs.1.0. We believe this is due to following:

- YassinOX botnet has higher packets inter-arrival rate than botnets used in first and second experiments.
- YassinOX bot process keeps changing port numbers it uses in very frequent manner.

4.1.3.4. Discussion summary:

Table 1 summarizes all discussion points of the three experiment used PCTool v.1.0 in section 4.1.2.

Experiment No.	PCTool v.1.0 performance	Justification
First	Good	<ul style="list-style-type: none">– Small PKT inter-arrival rate.– Port No. Not changing.
Second	Acceptable	<ul style="list-style-type: none">– Higher PKT inter-arrival rate.– Port No. Not changing
Third	Poor	<ul style="list-style-type: none">– Very high PKT inter-arrival rate.– Port No. frequently changing

Table 1: A summary of all dissection points for the three experiment used PCTool

v1.0

4.2. PCTOOL VS.2.0 DEVELOPMENT:

In this version of PCTool, we tried to overcome issues of PCToolvs.1.0 examined, and discussed in section 4.1.2. In order to do that, we changed the design of the tool. As shown in figure 14, we exploited multi-threading concept. Three threads were used, they integrate each other, and each of which has certain tasks.

4.2.1. Thread (1):

This thread solves the issue of writing processed data directly to text file in hard disk. Instead, it will write data into memory structure. In this case, “a queue”. The queue is for temporary storage of process’s data, where it cleaned frequently by thread (2).

Thread (1) Tasks:

- Captures every outgoing\incoming network packet.
- Associates captured packet with its process using updated list of alive processes provided by thread (3).
- Add processed data to the queue.

4.2.2. Thread (2):

This thread starts at the same time thread (1) starts. Thread (2) is an infinite loop that reads and empty –to avoid queue overflow- the queue filled by thread (1), and then, writes to hard disk where text file located. We assume that computation speed is way faster than packet inter-arrival rate of packets. This means, no queue overflow.

4.2.3. Thread (3):

This thread solves delay issue of getting port numbers of running processes. The only task of thread (3) is updating list of alive processes in PCToolvs.2.0 frequently. The list updated every 10 millisecond. We also tried 100 millisecond to update the process list. We found no difference in performance. So we kept the smallest which is 10 milliseconds.

Experiment in section 4.1.2.3 conducted once again using PCToolvs.2.0. We gathered information from each computer and analyzed bot traffic behavior. In following we will provide details about experiment and then discuss our findings.

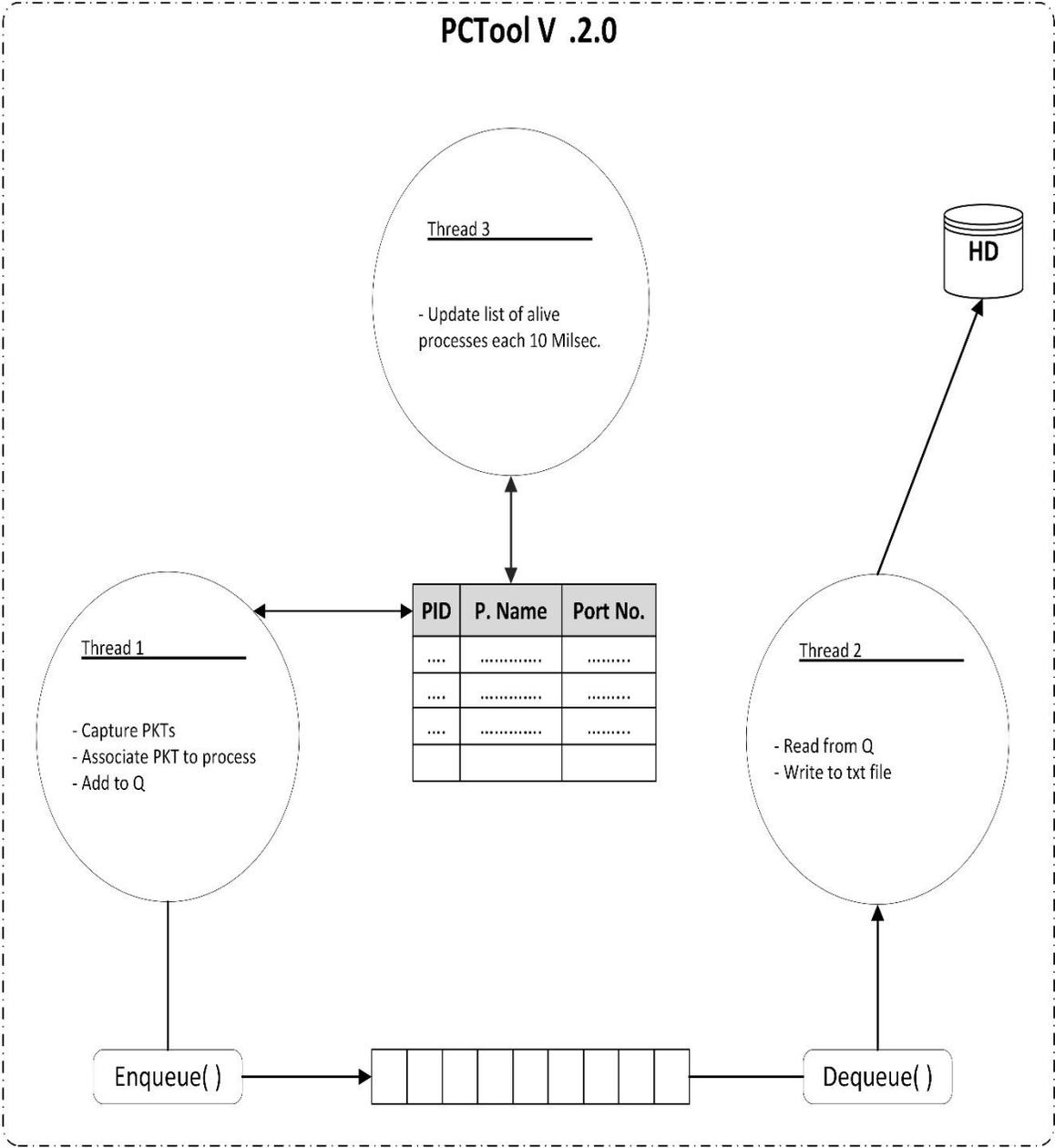


Figure 8: Illustration of PCTool V.2.0

4.2.4. Experiment Setup:

This experiment is to test the ability of PCTool v 2.0 to associate traffics to YassinOX process during monitoring time. We run bots in more than one PC just to avoid the abnormal behavior and to ensure that no misconfiguration in single PC.

We installed Packet Capturing Tool and Process Reveal Tool on all machines (4 PCs).

Apache Server and YassinOX Bot-master was installed and Configured on PC-4 (172.1.0.4) where .EXE file built.

EXE file executed on remaining machines (PC-5, PC-6, PC-7, and PC-8).

IP addresses of e PCs as following:

PC-5 / 172.16.0.5

.PC-8 / 172.16.0.8

Last digit of IP is e same as in PC name. Figure 9 show behavior of bot in this experiment. Notice, we can see big improvement in PCTool vs.2.0. Especially, where PCTool vs.1.0 could not associate packets to same bot (YassinOX) process, see section 4.1.2.3, but PCTool vs.2.0 showed good performance. We found all figures from all PCs are alike, so we are adding only a figure below.

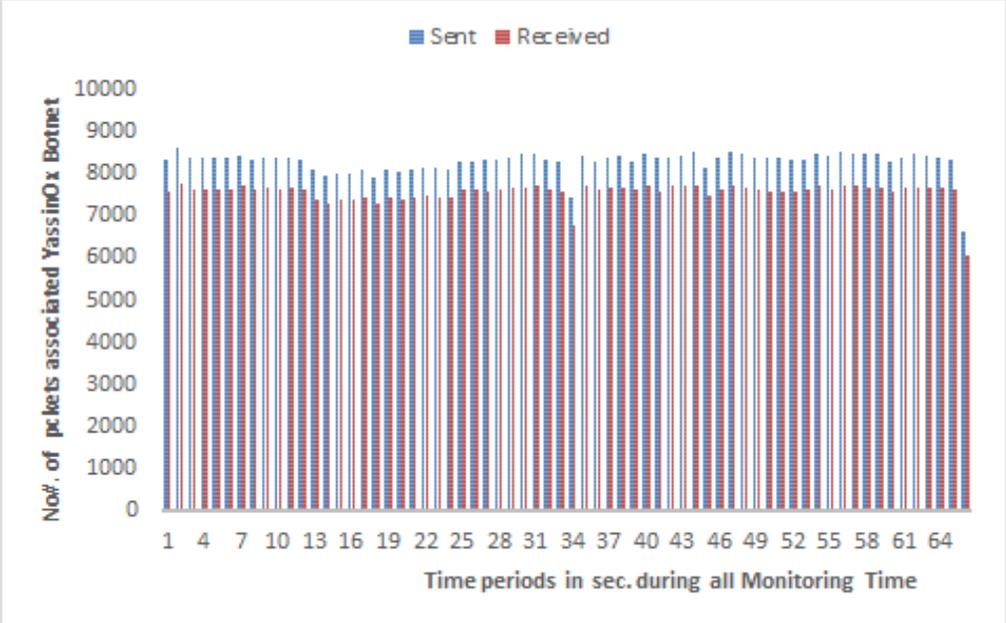


Figure 9: Packets sent and received by YassinOx Process

4.2.5. Discussion:

With PCTool vs.2.0, we believe that we overcame situations where PCTool vs.1.0 examined failure to associate captured packet to a live process, see section 4.2.4. From now on this PCTool will refer to PCTool vs.2.0.

4.2.6. Data set:

After testing our tool (PCTool), we conducted experiment with same setup as in above experiments to collect data set. Botnets used in previous experiments installed in machines. In addition, various benign application has been installed as well such as, Mozilla firefox, google chrome, skype, tribler (torrent), and Bitcommt(torrent).

Both malicious and benign application executed while machines monitored using PCTool. MATLAB code used to perform statistical analysis on collected data from PCTool. Output of the MATLAB code formed statistical features that used as classification factors for SVM classification model (see chapter 5).

CHAPTER 5

RESULT AND DISCUSSION

We chose experimental approach to find patterns of bots traffic. Experiments are based on observing and comparing traffic behavior of known bots, with traffic behavior of other benign programs. Firstly, we built Packet Capturing Tool (PCTool) to capture each outgoing or incoming packets to computer and associate captured packet to process. This tool is to observe behavior of bot traffics in host-based manner. Secondly, we built an isolated network, in which each host in this network is infected (is a bot) and has PCTool installed. Three botnets (DDoSeR 3.0 Mod, Zeus botnet, and YassinOX botnet) were used. Botnets detailed in section 4.1.1.

As study of Zhang et al. [18] concluded that an effective botnet detection methodology should rely on behavior anomaly, statistical analysis, and independent of any structure/protocol. In our work, machine learning was not used for features extraction. Instead, we rely on statistical method to extract statistics out of captured packets using our MATLAB code. Output of the MATLAB code formed statistical features that used as classification factors for SVM classification model to determine behavior of malicious or benign processes. Then, we use these statistics as main features (or factors) of classification model by building SVM classifier. We have six features: 1) period of time (ΔT), 2) Number of Packets Sent, 3) Number of Packets Received, 4) Number of IPs, 5) Number of Local ports, 6) Number of Remote ports. These features are collected in file for further processing.

We assumed that, depending on only these six features we can classify malicious and benign traffic. Table 2 shows sample of features, each line in table is resampling one period of time ($1 \Delta T$). Ranges of ΔT are set for each run of experiment.

PID	ΔT	No. of Pkts Sent	No. of Pkts Received	No. of IPs	No. L_Ports	No. R_Ports	M.Status
3152	100	55	39	4	2	6	0
1732	200	6	5	2	4	3	1

Table 2 : Show sample of features used for training

5.1. MALICIOUS - BENIGN PROCESS CLASSIFICATION:

Classification is a way to predict labels of class for a given data input. In our work we are doing binary classification, with two possible output classes' malicious process or benign process.

In machine learning code we use SVM as our classifier, and radial basis function (RBF) [76], as kernel function. K-folds cross validation [77] was used in our experiment, where dataset randomly broken into k partitions (or folds). One fold taken for testing, remaining folds for training. The model should run for K times. In each run, testing part is changing. For instance, in first run last part (k) is for testing rest parts for training, in second run part (k-1) is for testing and rest parts are for training... etc. see figure 17. In our work we considered ten-folds cross validation ($K = 10$) which best performing variation of cross-validation [78].

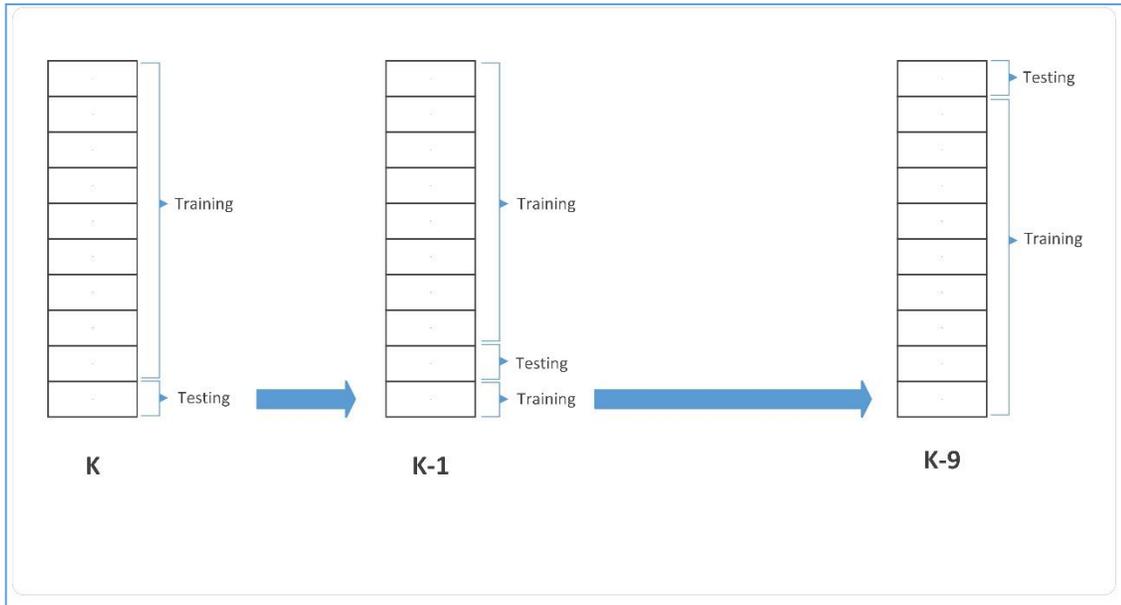


Figure 10: Illustration of how data divided to K parts (K=10), each part. One part for testing and other 9 parts are for training. We run the model for k times. In each run, the testing part is changed.

5.2. EVALUATION METRICS:

After the ten runs of our classification model are done. We need to show how the classifier can separate malicious process from normal process. There are many ways to show classifier performance. **Confusion Matrix** and **Area under the Curve (AUC)** are well-known classification performance measuring metrics [79]. In order to find AUC, we need to plot **receiver operating characteristic (ROC)** curve.

In sections 5.2.1, and 5.2.2 we will give brief explanation of **AUC** of ROC curve, and confusion matrix.

5.2.1. Receiver Operating Characteristic (ROC) curve:

ROC curve is a graphic plotting to illustrate binary classifier performance as its cut-point (or Discrimination threshold) varied. ROC curve presented graphically by plotting sensitivity (TPR) against 1-specificity (FPR) of classifier at various cut-point settings.

ROC curve is a whole curve. It provides a subtle difference in details about behavior of classifier, but it's hard to quickly compare many ROC curves to each other. Machine would need quantifiable score instead of plot that requires visual inspection. AUC is one way to summarize ROC curve into a single number so that it can be compared easily and automatically.

5.2.2. Area under the Curve (AUC):

Area under the ROC Curve is a measurement of accuracy of binary classifier. A perfect classifier represented with area of 1 (100%), while worthless classifier represented with 0.5 (50%) of area or less [80] [81]. A guide for categorizing accuracy of classifier is illustrated in table 3.

Accuracy	Fail	Poor	Fair	Good	Excellent
Category	50% - 60%	60% - 70%	70% - 80%	80% - 90%	90% - 100%

Table 3: A guide for classifying accuracy

5.2.3. Confusion matrix:

Confusion matrix contains information about actual and predicted classifications done by a classification system. Performance of such systems is commonly evaluated using data in the matrix. Table 4 shows confusion matrix for two class classifier.

	Predicted (P)	Predicted (N)
Actual (P)	True Positive (TP)	False Negative (FN)
Actual (N)	False Positive (FP)	True Negative (TN)

Table 4: Contingency table show the arrangement of TP, FP, FN, and TN which are the output of the confusion matrix

In predictive analytics, contingency table (or confusion matrix) reports number of true positives, false positives, false negatives, true positives, and true negatives to allow precise analysis. When using K-folds cross-validation TP, FP, FN, and TN are computed respectively by the following equations:

Metric Name	Equations	Location in confusion matrix
True Positive (TP)	$TP = \frac{1}{k} \sum_{i=1}^k TP_i$	(1, 1)
False Positive (FP)	$FP = \frac{1}{k} \sum_{i=1}^k FP_i$	(2, 1)
False Negative (FN)	$FN = \frac{1}{k} \sum_{i=1}^k FN_i$	(1, 2)
True Negative (TN)	$TN = \frac{1}{k} \sum_{i=1}^k TN_i$	(2, 2)

Table 5: how to calculate TP, FP, FN, and TN

Where k is number of folds used in training and testing model. Then, True Positives Rate (TPR), False Positives Rate (FPR), False Negatives Rate (FNR), and True negatives Rate (TNR) are calculated as in table 6 below:

Metric Name	Equations
True positive rate (or sensitivity)	$\text{TPR} = \frac{\text{TP}}{(\text{TP} + \text{FN})}$
False positive rate	$\text{FPR} = \frac{\text{FP}}{(\text{FP} + \text{TN})}$
True negative rate (or specificity)	$\text{TNR} = \frac{\text{TN}}{(\text{FP} + \text{TN})}$
False Negative rate	$\text{FNR} = \frac{\text{FN}}{(\text{FP} + \text{TN})} \text{ or } \text{FNR} = 1 - \text{TPR}$

Table 6: How to calculate TPR, FPR, TNR, and FNR

5.3. RESULTS:

In this section, we present our experimentation results of the proposed malicious - benign process classification technique. We tested the classification model for various ranges of ΔT . Our target is to get excellent AUC with the smallest ΔT and FPR less than 1 %. Keeping FPR very low means very less benign application detected as bot. FNR representing bot process detected as benign which something tolerable and does not affect users.

We started with, nine ΔT ranges starting from 100 Milliseconds to 4500 milliseconds. The 100 Milliseconds is only the smallest period of time we will check the accuracy of our classifier in. Periods will start from 100, 200, 300... 4500. The aim is to find smallest period that achieves the lowest FBR. Each ΔT range consist of 5 steps where each step is 100 milliseconds. In following, we show results and figures for various ΔT ranges.

5.3.1. $\Delta T = 100-500$ Milliseconds:

In first range where ΔT vary from 100 – 500 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 18 - 27 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 7 shows AUCs of figures 18-27.

Figure Number	K values (Fold #)	AUCs	Mean AUC
18	1	16.1008 %	60.31%
19	2	86.57148 %	
20	3	24.7939 %	
21	4	41.55969 %	
22	5	24.00294 %	
23	6	94.95488 %	
24	7	96.77862 %	
25	8	94.7456 %	
26	9	93.33067 %	
27	10	30.29309 %	

Table 7: Summary of AUCs for Ten-Folds where $\Delta T = 100 - 500$

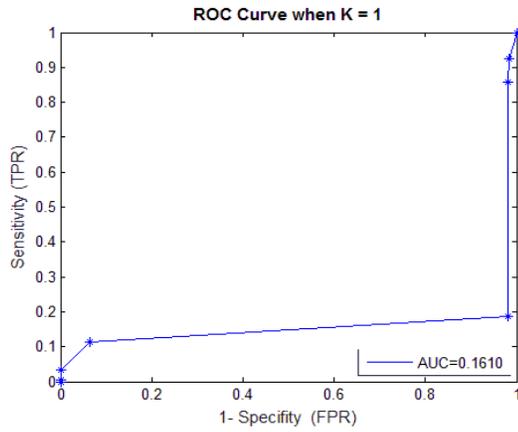
AUC of all Ks in ROC curves is summarized with mean AUC (60.31 %). According to accuracy guide of AUC (see table 3) the classification model almost fail in range of $\Delta T = 100 - 500$ milliseconds.

Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. Accordingly, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate

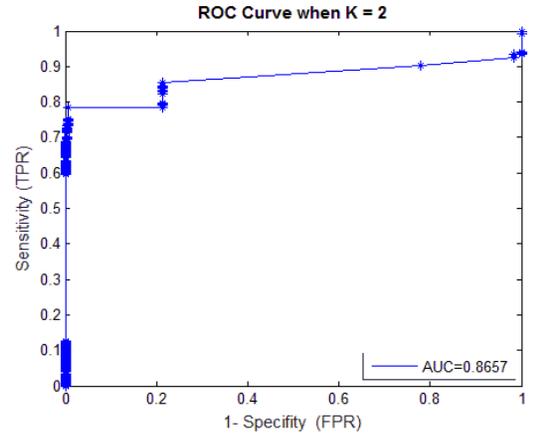
(TNR) as shown in table 8, where FBR is 99.8% and that considered failure of the classification model. Both metric showed same results.

ΔT	TPR	FPR	FNR	TNR
100-500	99.8%	99.8%	0.18%	0.19%

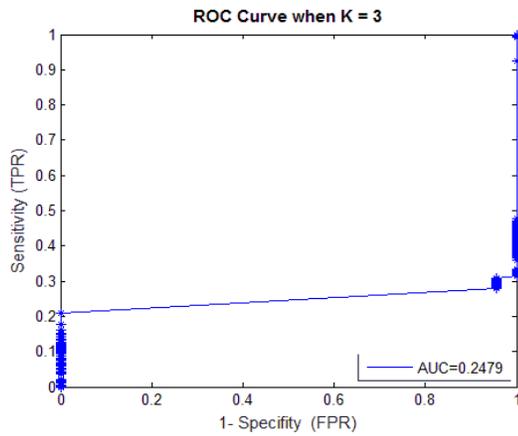
Table 8: TPR, FPR, FNR, and TNR where $\Delta T = 100-500$



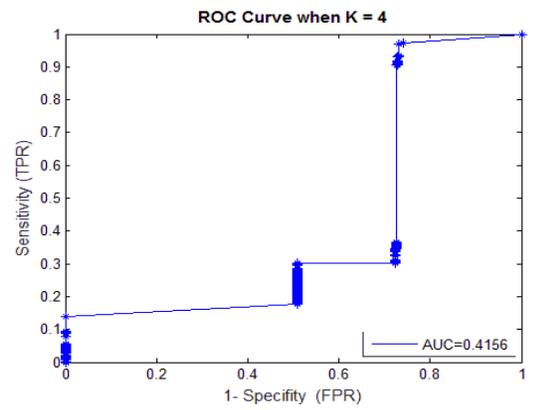
**Figure 11: ROC Curve when $\Delta T = 100$ -
500 Millisecond , and K=1**



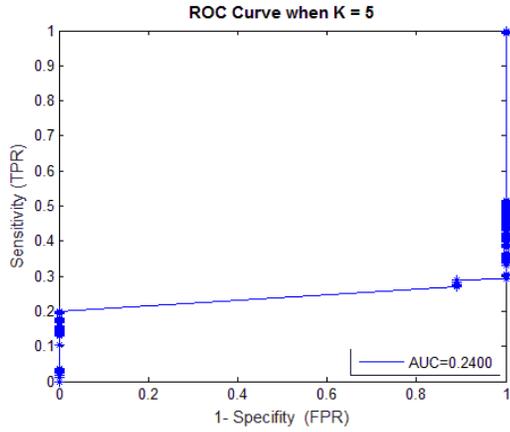
**Figure 12: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=2**



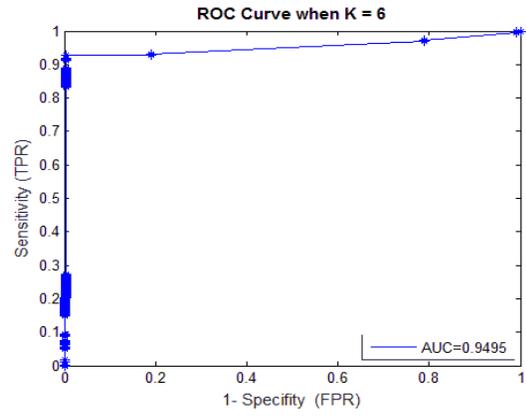
**Figure 13: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=3**



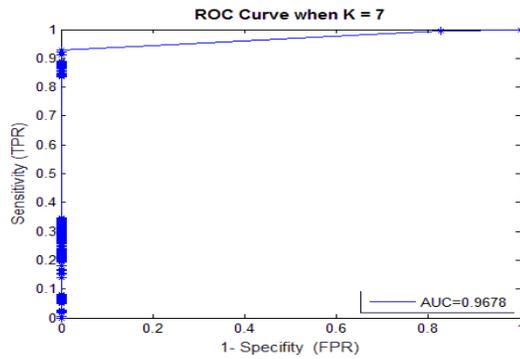
**Figure 14: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=4**



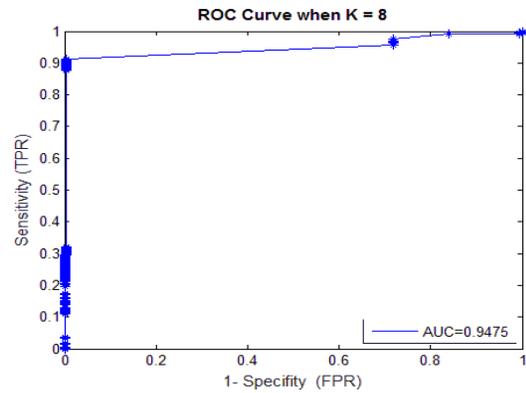
**Figure 15: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=5**



**Figure 16: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=6**



**Figure 17: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=7**



**Figure 18: ROC Curve when $\Delta T = 100$ -
500 Millisecond, and K=8**

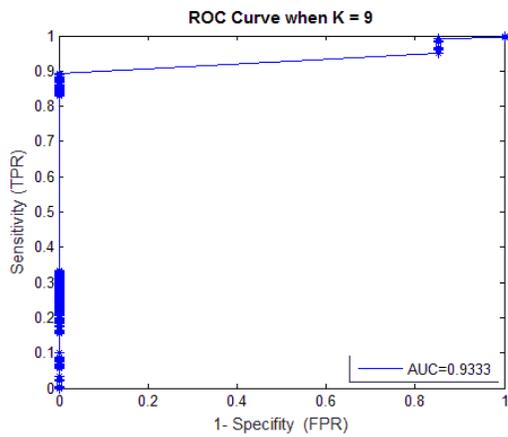


Figure 19: ROC Curve when $\Delta T = 100-500$ Millisecond , and $K=9$

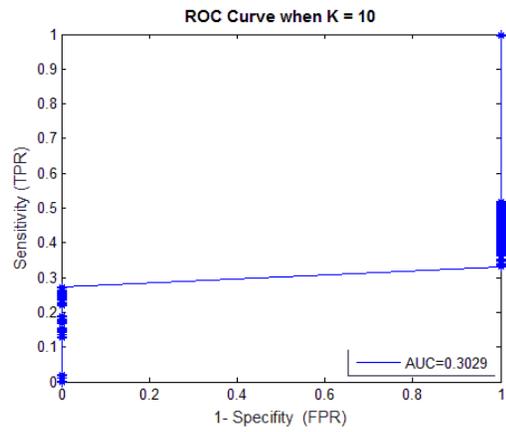


Figure 20: ROC Curve when $\Delta T = 100-500$ Millisecond, and $K=10$

5.3.2. $\Delta T = 600-1000$ Milliseconds:

In the second range where ΔT vary from 600 – 1000 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 28 - 37 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 9 shows AUCs of figures 28-37.

Figure Number	K values (Fold #)	AUCs	Mean AUC
28	1	71.76044%	51.92%.
29	2	30.42461%	
30	3	69.75216%	
31	4	34.62774%	
32	5	73.92658%	
33	6	35.60788%	
34	7	72.78775%	
35	8	36.35425%	
36	9	61.78402%	
37	10	32.21035%	

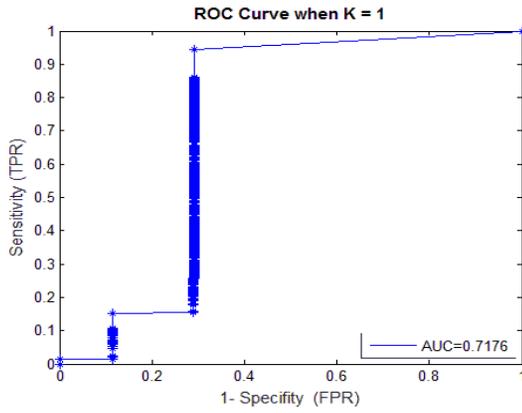
Table 9: Summary of AUCs for Ten-Folds where $\Delta T = 600 - 1000$

AUC of all Ks in ROC curves is summarized with mean AUC (**51.92%**). According to accuracy guide of AUC (see table 3) the classification model fail in the range of $\Delta T = 600 - 1000$ milliseconds.

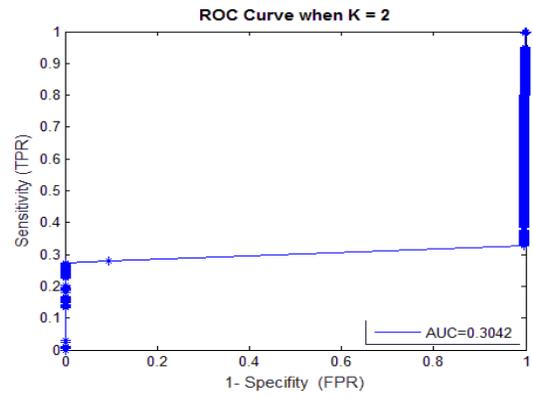
Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 10, where FBR is 66.1% and that considered failure of the classification model. Both metric showed same results.

ΔT	TPR	FPR	FNR	TNR
600-1000	97%	66.1%	3.02%	33.8%

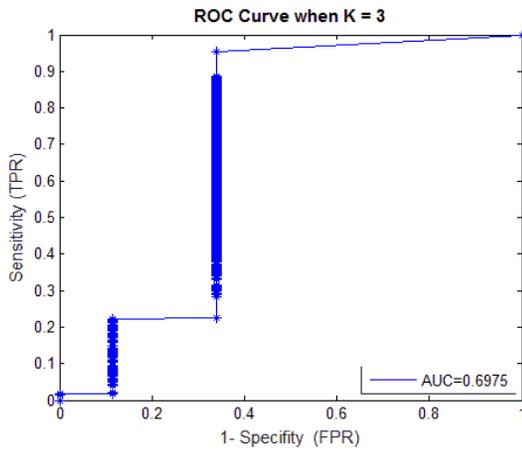
Table 10: TPR, FPR, FNR, and TNR where $\Delta T = 600-1000$



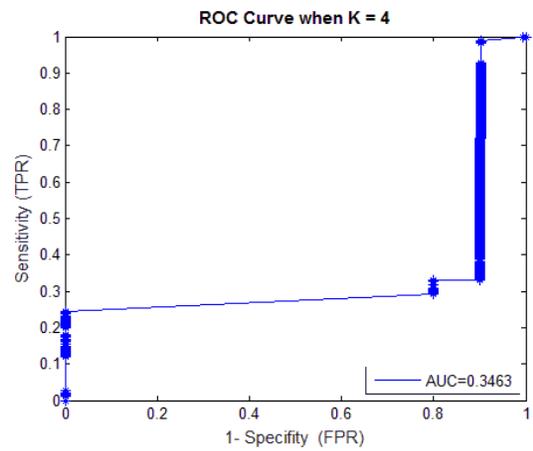
**Figure 21: ROC Curve when $\Delta T = 600$ -
1000 milliseconds , and $K=1$**



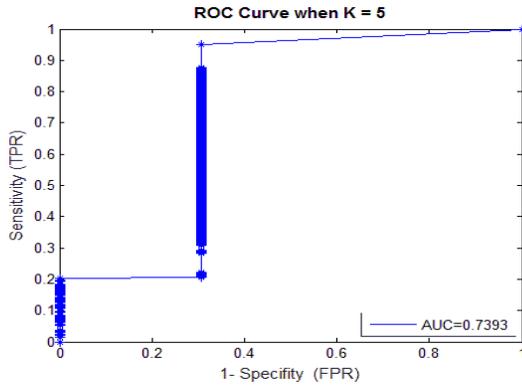
**Figure 22: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=2$**



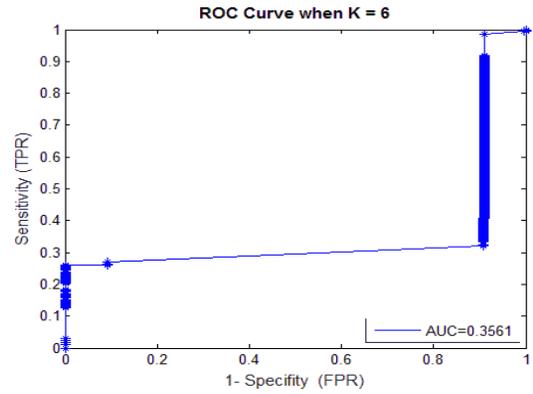
**Figure 23: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=3$**



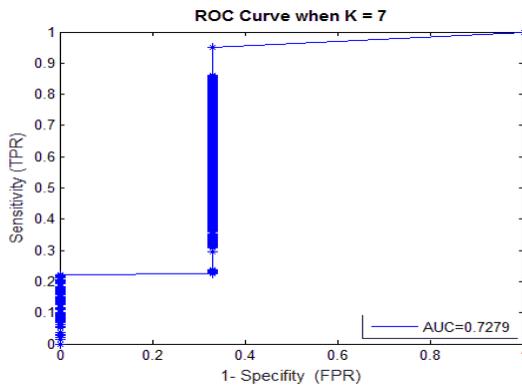
**Figure 24: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=4$**



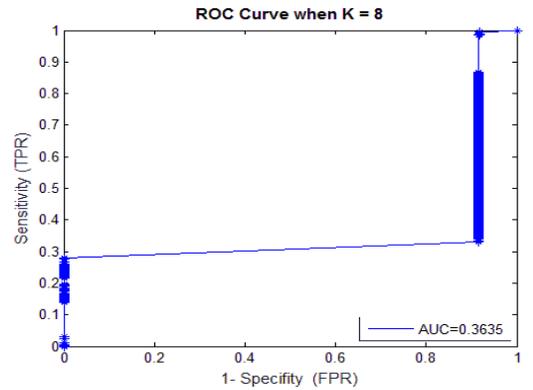
**Figure 25: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=5$**



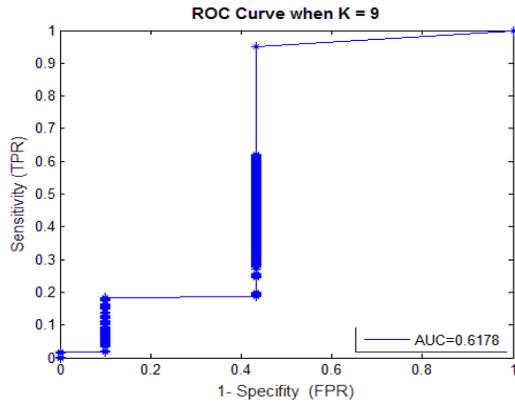
**Figure 26: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=6$**



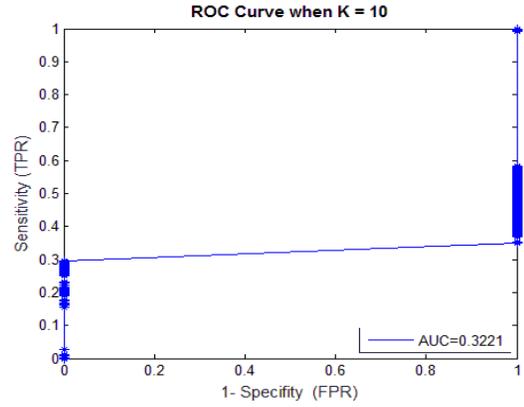
**Figure 27: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=7$**



**Figure 28: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=8$**



**Figure 29: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=9$**



**Figure 30: ROC Curve when $\Delta T = 600$ -
1000 milliseconds, and $K=10$**

5.3.3. $\Delta T = 1100-1500$ Milliseconds:

In the third range where ΔT vary from 1100 – 1500 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 38 - 47 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 11 shows AUCs of figures 38-47.

Figure Number	K values (Fold #)	AUCs	Mean AUC
38	1	63.41207%	68.2%
39	2	41.07005%	
40	3	65.10728%	
41	4	28.79476%	
42	5	63.47819%	
43	6	97.07812%	
44	7	65.25282%	
45	8	96.61236%	
46	9	64.46695%	
47	10	96.69529%	

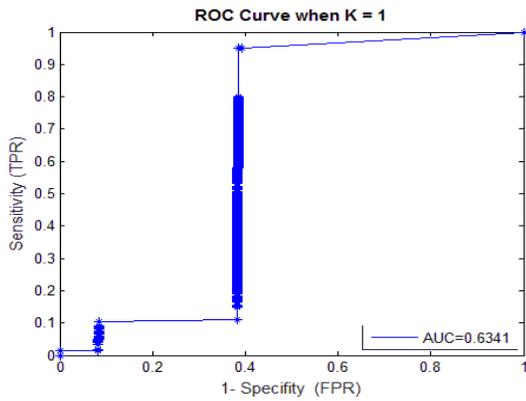
Table 11: Summary of AUCs for Ten-Folds where $\Delta T = 1100 - 1500$

AUC of all Ks in ROC curves is summarized with mean AUC (**68.2%**). According to accuracy guide of AUC (see table 3) the classification model performed poorly in the range of $\Delta T = 1100 - 1500$ milliseconds.

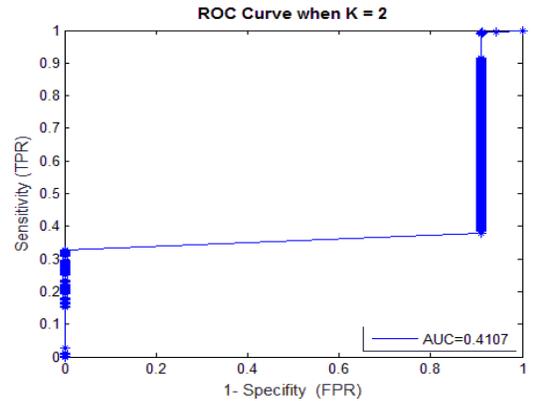
Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 12, where FBR is 38.2% and that considered a poor performance of the classification model. Both metric are almost on the same line.

ΔT	TPR	FPR	FNR	TNR
1100-1500	95.73%	38.2%	4.2%	61.8%

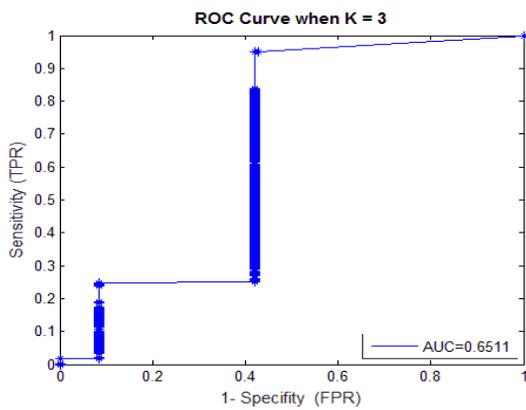
Table 12: TPR, FPR, FNR, and TNR where $\Delta T = 1100-1500$



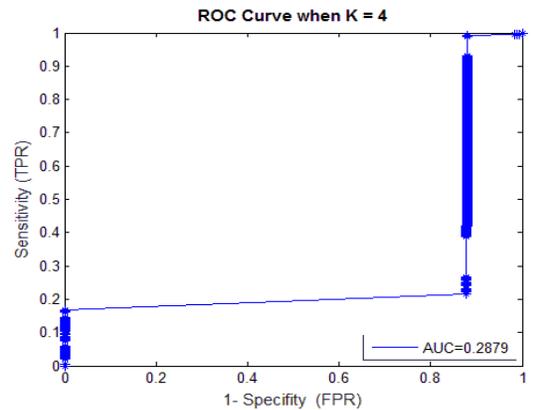
**Figure 31: ROC Curve when $\Delta T = 1100$ -
1500 milliseconds , and K=1**



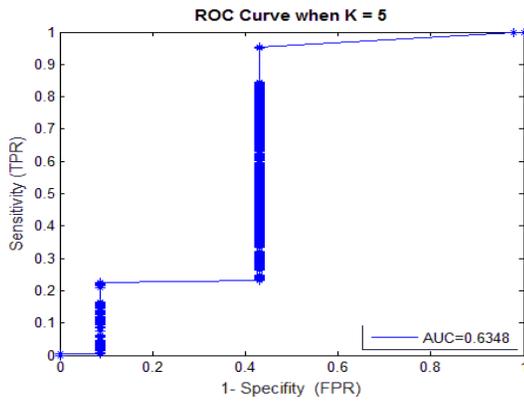
**Figure 32: ROC Curve when $\Delta T =$
1100-1500 milliseconds, and K=2**



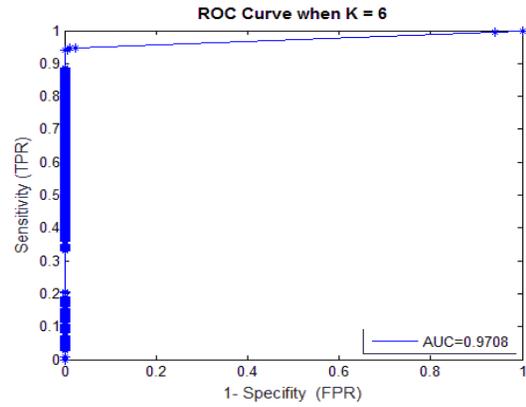
**Figure 33: ROC Curve when $\Delta T = 1100$ -
1500 milliseconds, and K=3**



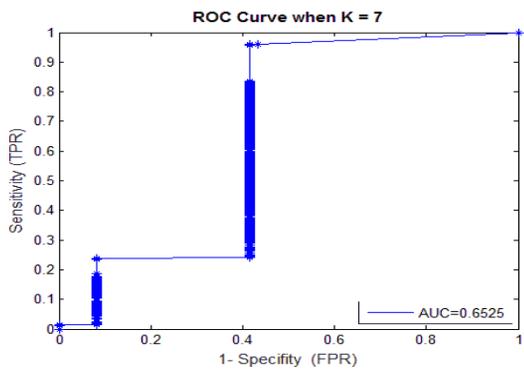
**Figure 34: ROC Curve when $\Delta T =$
1100-1500 milliseconds, and K=4**



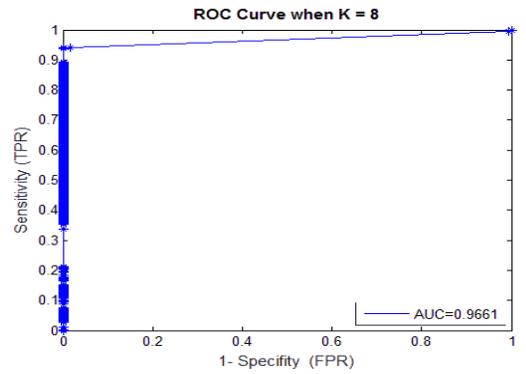
**Figure 35: ROC Curve when $\Delta T = 1100$ -
1500 milliseconds, and $K=5$**



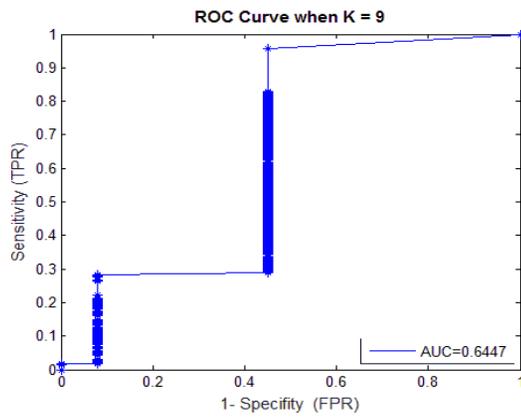
**Figure 36: ROC Curve when $\Delta T =$
1100-1500 milliseconds, and $K=6$**



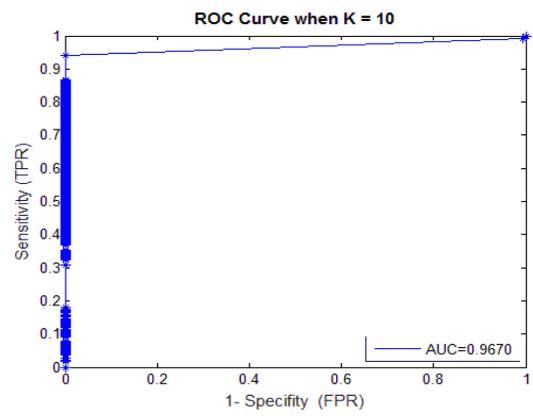
**Figure 37: ROC Curve when $\Delta T = 1100$ -
1500 milliseconds, and $K=7$**



**Figure 38: ROC Curve when $\Delta T =$
1100-1500 milliseconds, and $K=8$**



**Figure 39: ROC Curve when $\Delta T = 1100$ -
1500 milliseconds, and K=9**



**Figure 40: ROC Curve when $\Delta T =$
1100-1500 milliseconds, and K=10**

5.3.4. $\Delta T = 1600-2000$ Milliseconds:

In the fourth range where ΔT vary from 1600 – 2000 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 48 - 57 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 13 shows AUCs of figures 48 - 57.

Figure Number	K values (Fold #)	AUCs	Mean AUC
48	1	59.88226%	89.2%
49	2	96.86411%	
50	3	65.60388%	
51	4	96.9108%	
52	5	93.29135%	
53	6	97.1421%	
54	7	93.53047%	
55	8	97.72783%	
56	9	93.21223%	
57	10	97.6072%	

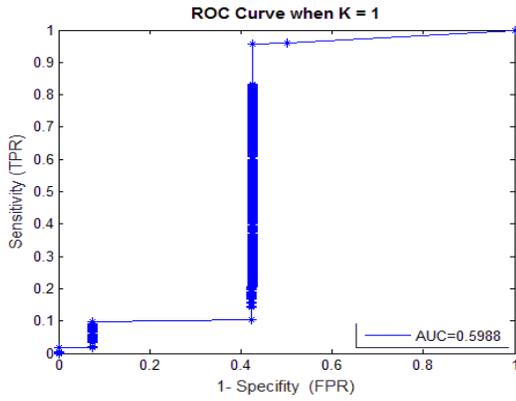
Table 13: Summary of AUCs for Ten-Folds where $\Delta T = 1600 - 2000$

AUC of all Ks in ROC curves is summarized with mean AUC (**89.2%**). According to accuracy guide of AUC (see table 3) the classification model showed good (almost excellent) performance in the range of $\Delta T = 1600 - 2000$ milliseconds.

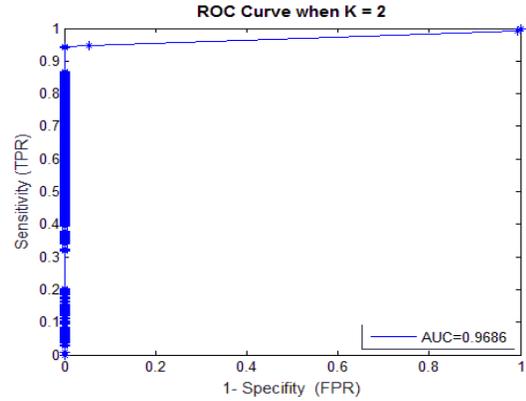
Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 14, where FBR is 10.14 % and that considered a good improvement of the classification model. Both metric are almost on the same line.

ΔT	TPR	FPR	FNR	TNR
1600-2000	95.05%	10.14%	4.94%	89.86%

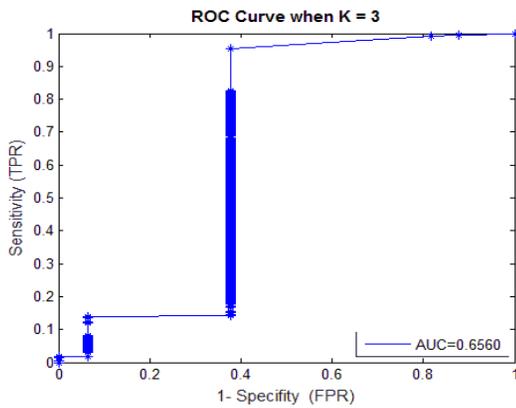
Table 14: TPR, FPR, FNR, and TNR where $\Delta T = 1600-2000$



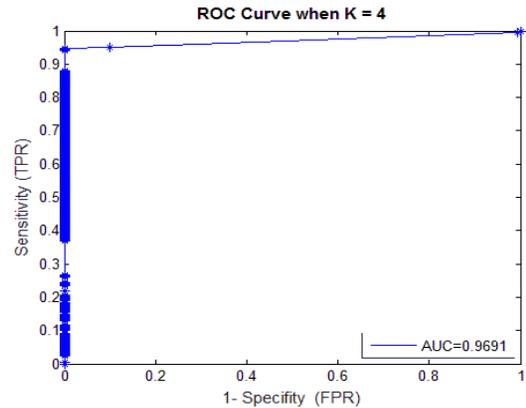
**Figure 41: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=1$**



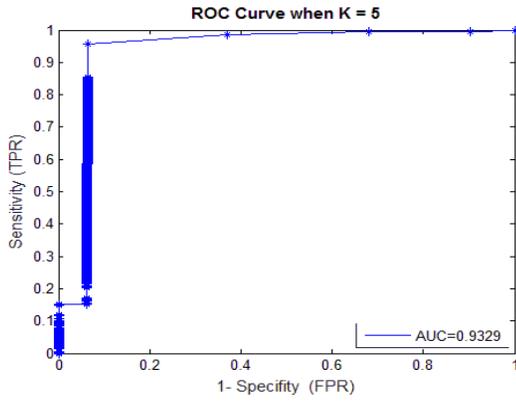
**Figure 42: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=2$**



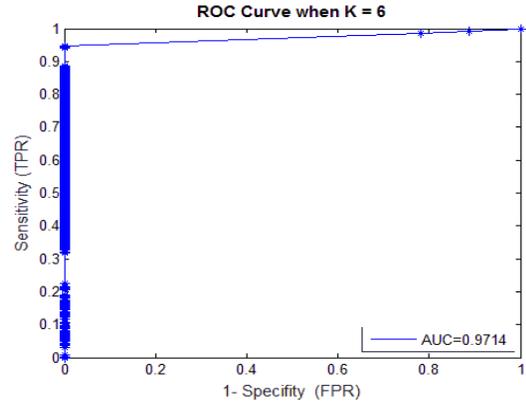
**Figure 43: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=3$**



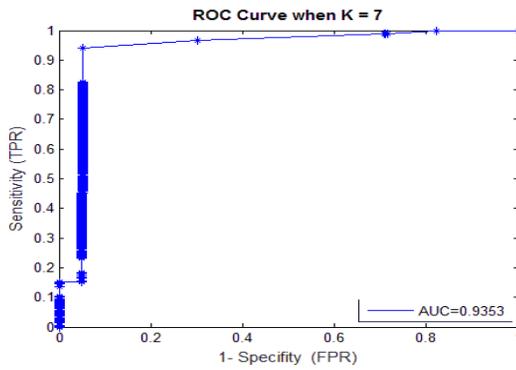
**Figure 44: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=4$**



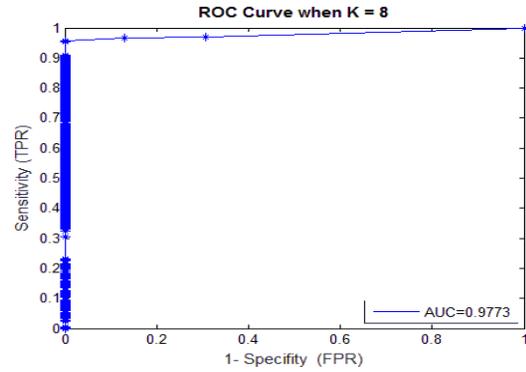
**Figure 45: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=5$**



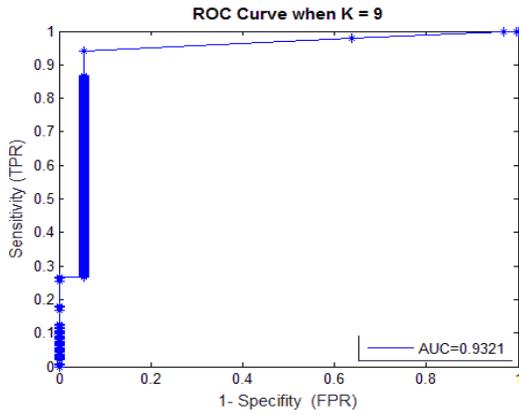
**Figure 46: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=6$**



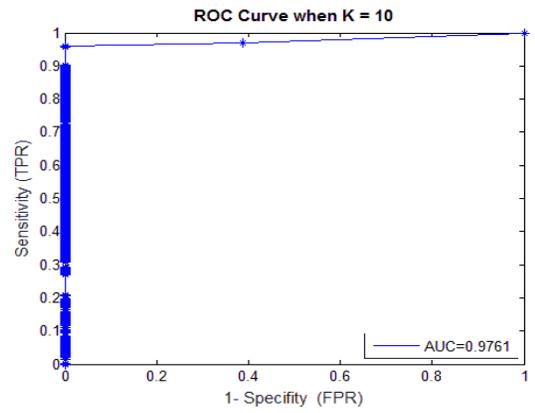
**Figure 47: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=7$**



**Figure 48: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and $K=8$**



**Figure 49: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and K=9**



**Figure 50: ROC Curve when $\Delta T =$
1600-2000 milliseconds, and K=10**

5.3.5. $\Delta T = 2100-2500$ Milliseconds:

In the fifth range where ΔT vary from 2100 – 2500 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 58 - 67 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 15 shows AUCs of figures 58 - 67.

Figure Number	K values (Fold #)	AUCs	Mean AUC
58	1	62.8526%	87.35%
59	2	97.655%	
60	3	92.46053%	
61	4	97.34113%	
62	5	93.37351%	
63	6	98.26763%	
64	7	92.8983%	
65	8	71.27156%	
66	9	96.70643%	
67	10	70.71158%	

Table 15: Summary of AUCs for Ten-Folds where $\Delta T = 2100 - 2500$

AUC of all Ks in ROC curves is summarized with mean AUC (**87.35%**). According to accuracy guide of AUC (see table 3) the classification model showed good performance in the range of $\Delta T = 2100 - 2500$ milliseconds with no improvement from previous range.

Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 16, where FBR is 12.51% and that considered a good but with no improvement of the classification model from previous range. Both metric are almost on the same line

ΔT	TPR	FPR	FNR	TNR
2100-2500	95.17%	12.51%	4.82%	87.48%

Table 16: TPR, FPR, FNR, and TNR where $\Delta T = 2100-2500$

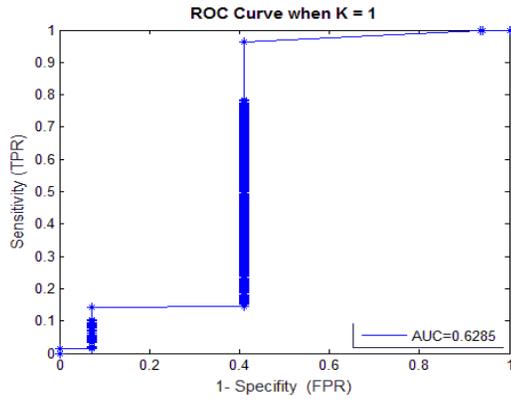


Figure 51: ROC Curve when $\Delta T =$ 2100-2500 milliseconds , and $K=1$

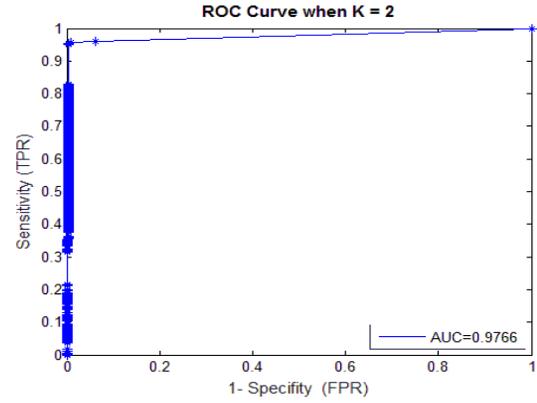


Figure 52: ROC Curve when $\Delta T =$ 2100-2500 milliseconds, and $K=2$

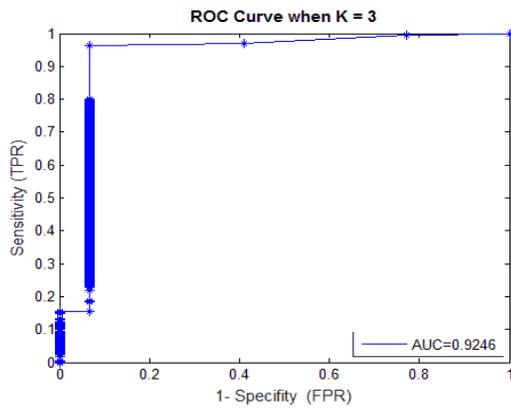


Figure 53: ROC Curve when $\Delta T =$ 2100-2500 milliseconds, and $K=3$

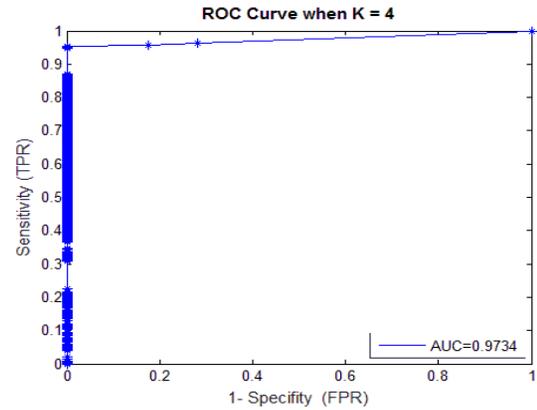
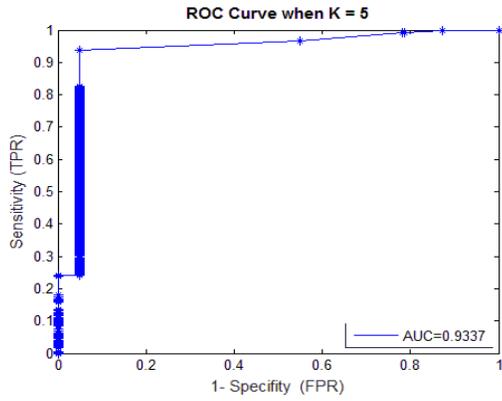
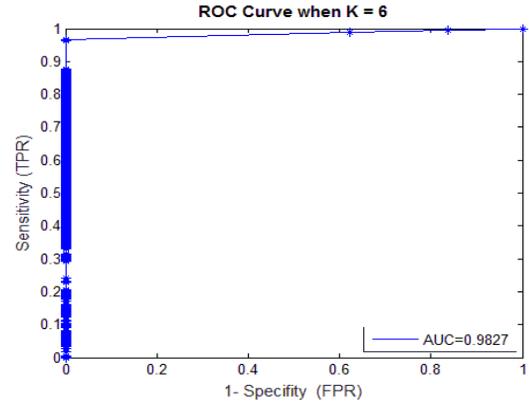


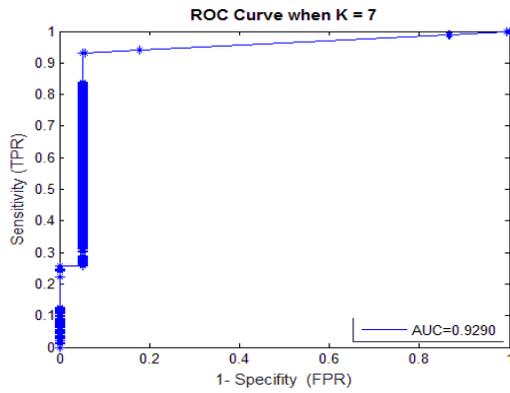
Figure 54: ROC Curve when $\Delta T =$ 2100-2500 milliseconds , and $K=4$



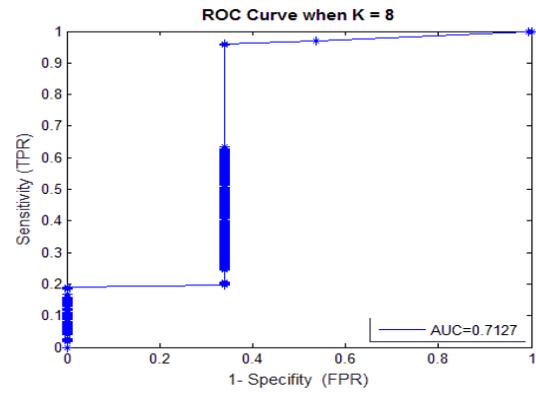
**Figure 55: ROC Curve when $\Delta T =$
2100-2500 milliseconds, and $K=5$**



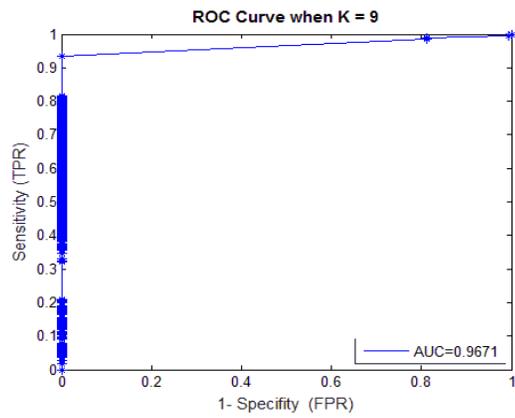
**Figure 56: ROC Curve when $\Delta T =$ 2100-
2500 milliseconds, and $K=6$**



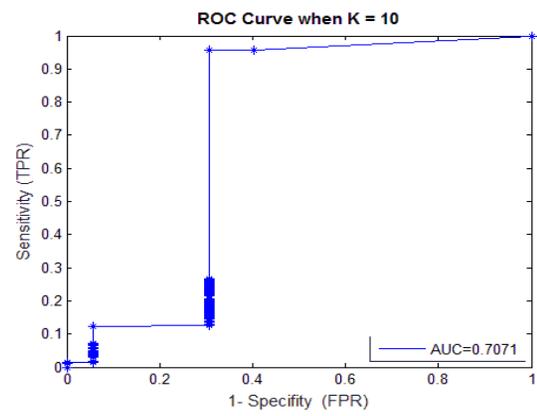
**Figure 57: ROC Curve when $\Delta T =$
2100-2500 milliseconds, and $K=7$**



**Figure 58: ROC Curve when $\Delta T =$ 2100-
2500 milliseconds, and $K=8$**



**Figure 59: ROC Curve when $\Delta T =$
2100-2500 milliseconds, and K=9**



**Figure 60: ROC Curve when $\Delta T =$ 2100-
2500 milliseconds, and K=10**

5.3.6. $\Delta T = 2600-3000$ Milliseconds:

In the sixth range where ΔT vary from 2600 – 3000 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 68 - 77 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 17 shows AUCs of figures 68 - 77.

Figure Number	K values (Fold #)	AUCs	Mean AUC
68	1	67.76058%	91.33%
69	2	97.61333%	
70	3	93.00419%	
71	4	97.69941%	
72	5	96.39581%	
73	6	72.78564%	
74	7	97.22593%	
75	8	96.72335%	
76	9	97.33914%	
77	10	96.83374%	

Table 17: Summary of AUCs for Ten-Folds where $\Delta T = 2600 - 3000$

AUC of all Ks in ROC curves is summarized with mean AUC (**91.33%**). According to accuracy guide of AUC (see table 3) the classification model showed excellent performance in the range of $\Delta T = 2600 - 3000$ milliseconds.

Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 18, where FBR is 6.91 % and that considered a excellent performance of the classification model but not yet reaching our target. Both metric are almost on the same line

ΔT	TPR	FPR	FNR	TNR
2600-3000	94.95%	6.91%	5.04%	93.09%

Table 18: TPR, FPR, FNR, and TNR where $\Delta T = 2600-3000$

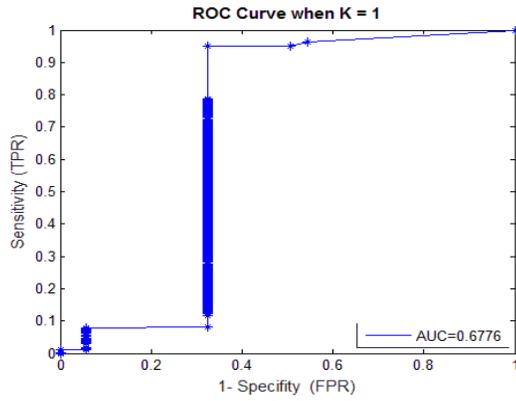


Figure 61: ROC Curve when $\Delta T =$ 2600-3000 milliseconds, and K=1

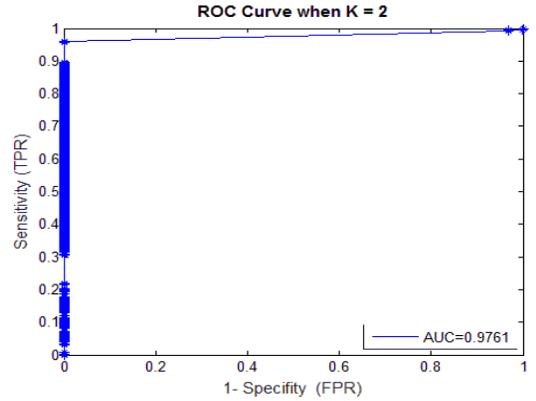


Figure 62: ROC Curve when $\Delta T =$ 2600-3000 milliseconds, and K=2

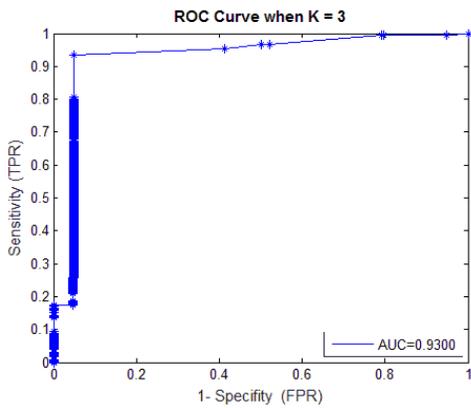


Figure 63: ROC Curve when $\Delta T =$ 2600-3000 milliseconds, and K=3

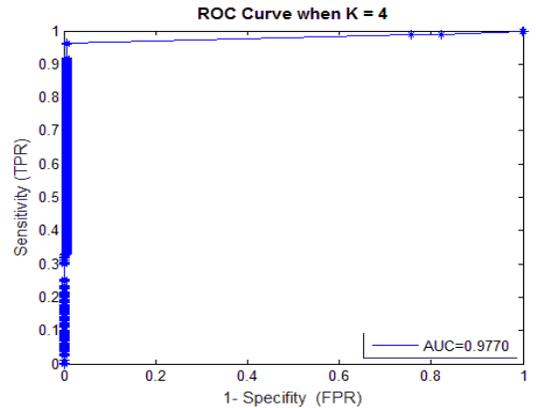
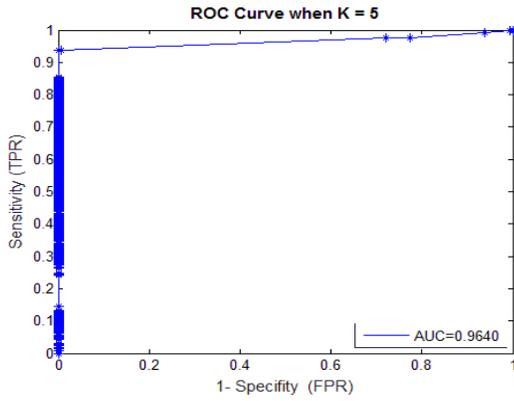
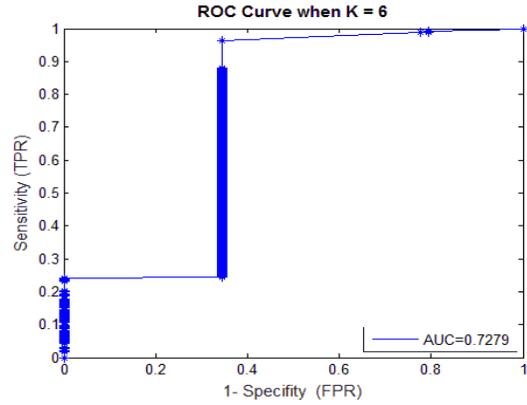


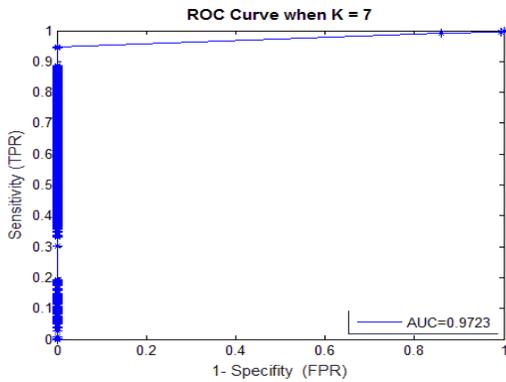
Figure 64: ROC Curve when $\Delta T =$ 2600-3000 milliseconds, and K=4



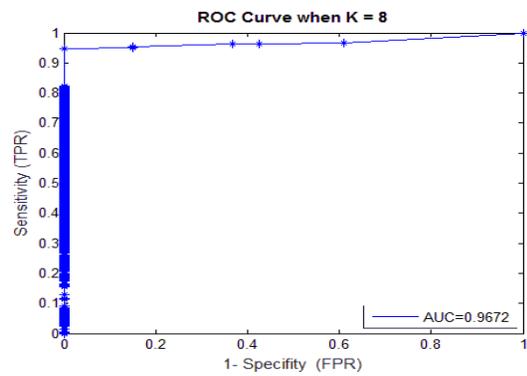
**Figure 65: ROC Curve when $\Delta T =$
2600-3000 milliseconds, and $K=5$**



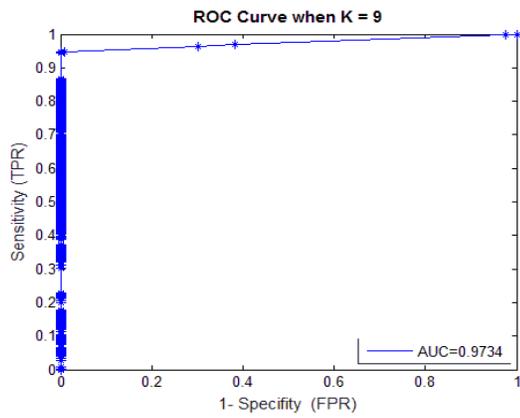
**Figure 66: ROC Curve when $\Delta T =$ 2600-
3000 milliseconds, and $K=6$**



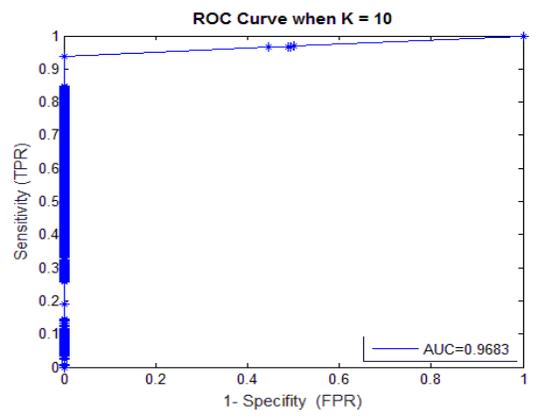
**Figure 67: ROC Curve when $\Delta T =$
2600-3000 milliseconds, and $K=7$**



**Figure 68: ROC Curve when $\Delta T =$ 2600-
3000 milliseconds, and $K=8$**



**Figure 69: ROC Curve when $\Delta T =$
2600-3000 milliseconds, and K=9**



**Figure 70: ROC Curve when $\Delta T =$ 2600-
3000 milliseconds, and K=10**

5.3.7. $\Delta T = 3100\text{-}3500$ Milliseconds:

In the seventh range where ΔT vary from 3100 – 3500 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 78 - 87 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 19 shows AUCs of figures 78 - 87.

Figure Number	K values (Fold #)	AUCs	Mean AUC
78	1	74.0044%	92.73%
79	2	98.39487%	
80	3	98.01491%	
81	4	97.93869%	
82	5	95.79271%	
83	6	74.61635%	
84	7	97.84251%	
85	8	96.7024%	
86	9	97.84928%	
87	10	96.18241%	

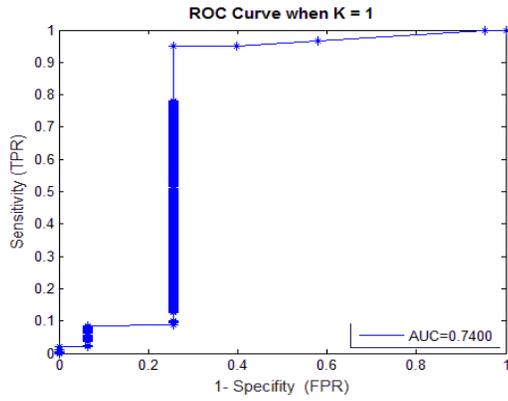
Table 19: Summary of AUCs for Ten-Folds where $\Delta T = 3100 - 3500$

AUC of all Ks in ROC curves is summarized with mean AUC (**92.73%**). According to accuracy guide of AUC (see table 3) the classification model showed excellent performance in the range of $\Delta T = 3100 - 3500$ milliseconds.

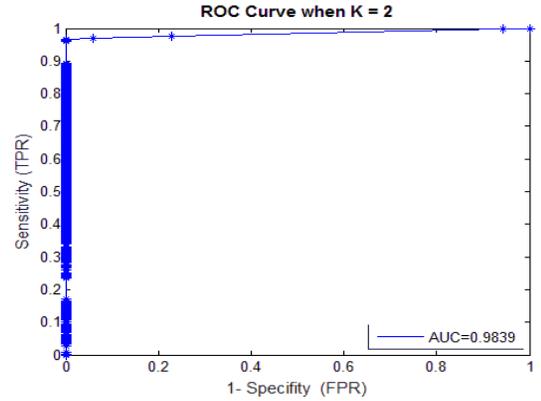
Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 20, where FBR is 6.91 % and that considered an excellent performance of the classification model with no improvement from previous range and yet not reaching our target. Both metric are almost on the same line

ΔT	TPR	FPR	FNR	TNR
3100-3500	94.81%	6.91%	5.18%	94.53%

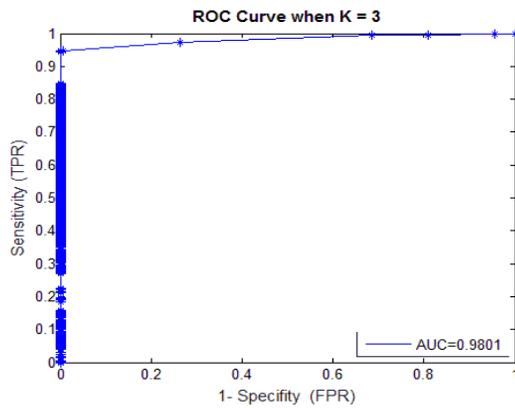
Table 20: TPR, FPR, FNR, and TNR where $\Delta T = 3100-3500$



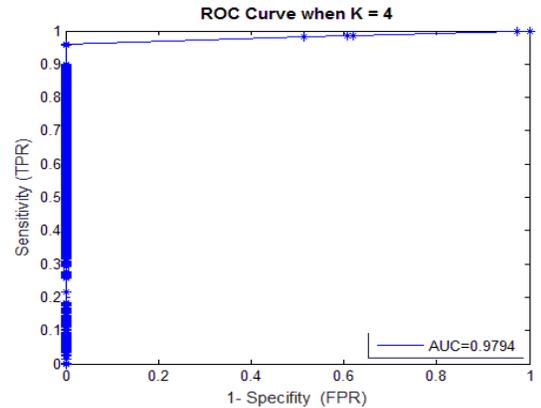
**Figure 71: ROC Curve when $\Delta T =$
3100-3500 milliseconds , and K=1**



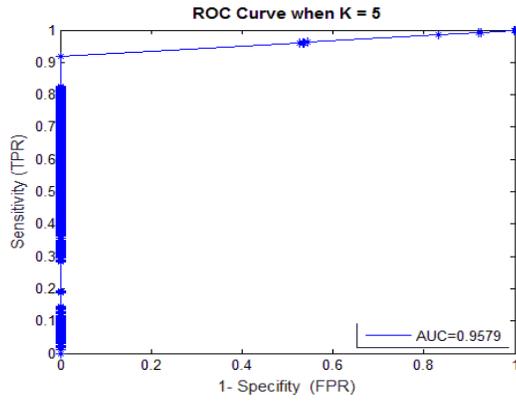
**Figure 72: ROC Curve when $\Delta T =$ 3100-
3500 milliseconds, and K=2**



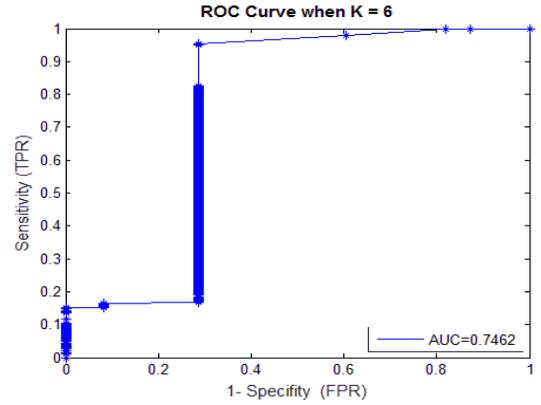
**Figure 73: ROC Curve when $\Delta T =$
3100-3500 milliseconds, and K=3**



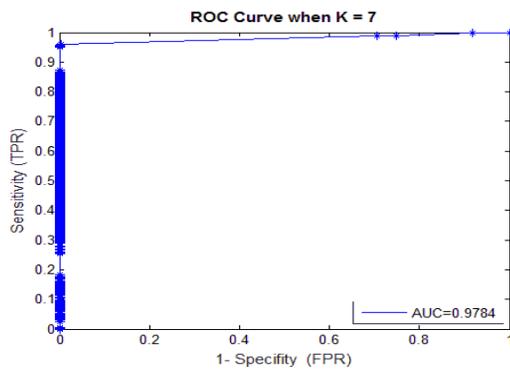
**Figure 74: ROC Curve when $\Delta T =$ 3100-
3500 milliseconds, and K=4**



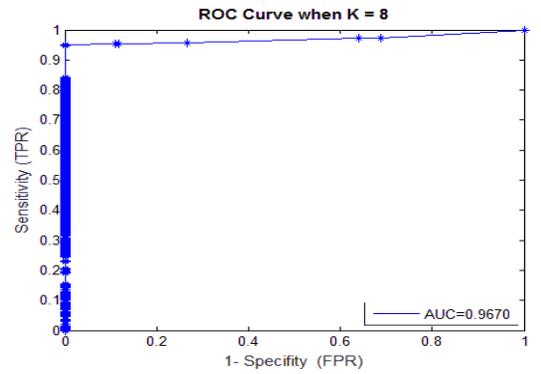
**Figure 75: ROC Curve when $\Delta T =$
3100-3500 milliseconds, and $K=5$**



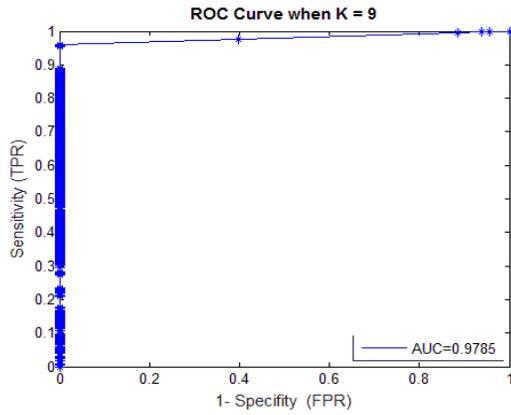
**Figure 76: ROC Curve when $\Delta T =$ 3100-
3500 milliseconds, and $K=6$**



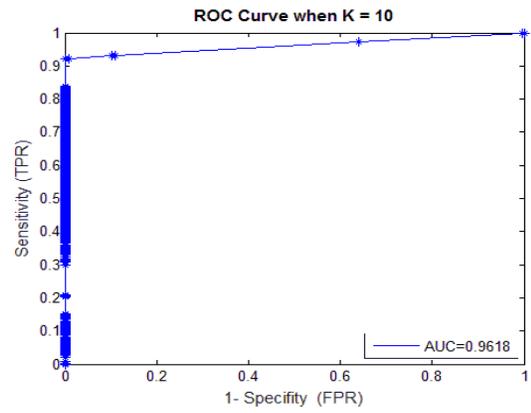
**Figure 77: ROC Curve when $\Delta T =$
3100-3500 milliseconds, and $K=7$**



**Figure 78: ROC Curve when $\Delta T =$ 3100-
3500 milliseconds, and $K=8$**



**Figure 79: ROC Curve when $\Delta T =$
3100-3500 milliseconds, and K=9**



**Figure 80: ROC Curve when $\Delta T =$ 3100-
3500 milliseconds, and K=10**

5.3.8. $\Delta T = 3600-4000$ Milliseconds:

In the eighth range where ΔT vary from 3600 – 4000 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 88 - 97 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 21 shows AUCs of figures 88 - 97.

Figure Number	K values (Fold #)	AUCs	Mean AUC
88	1	90.17688%	96.63%
89	2	97.68266%	
90	3	96.47288%	
91	4	98.79641%	
92	5	96.65926%	
93	6	98.4203%	
94	7	97.71725%	
95	8	96.39093%	
96	9	97.05732%	
97	10	97.00978%	

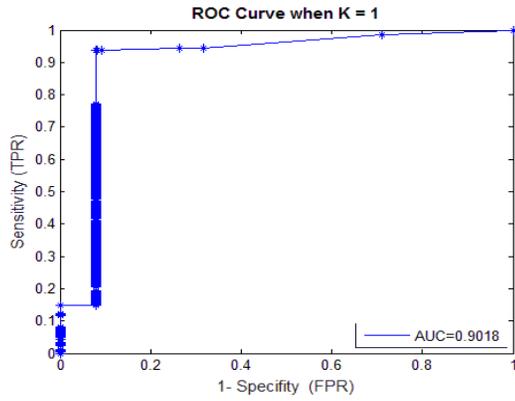
Table 21: Summary of AUCs for Ten-Folds where $\Delta T = 3600 - 4000$

AUC of all Ks in ROC curves is summarized with mean AUC (**96.63%**). According to accuracy guide of AUC (see table 3) the classification model showed excellent performance in the range of $\Delta T = 3600 - 4000$ milliseconds.

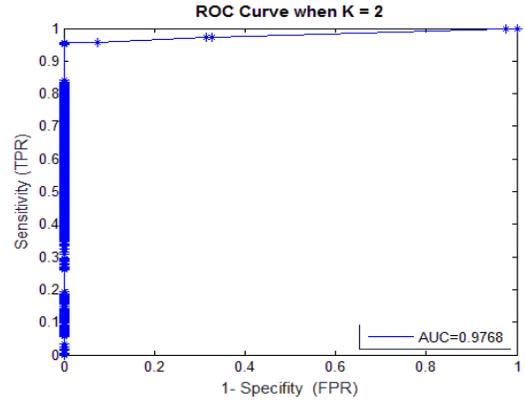
Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 22, where FBR is 1.067 % and that considered an excellent performance of the classification model with improvement from previous range but yet not reaching our target. Both metric are almost on the same line

ΔT	TPR	FPR	FNR	TNR
3600-4000	94.48%	1.067%	5.528%	98.93%

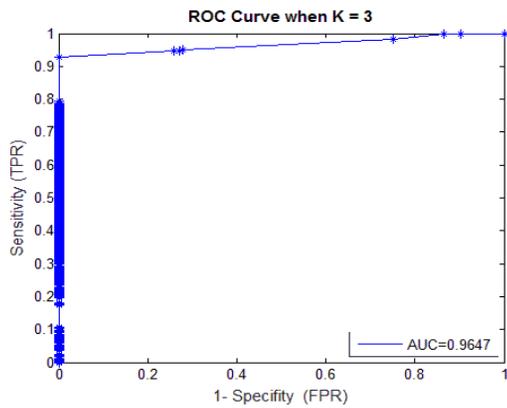
Table 22: TPR, FPR, FNR, and TNR where $\Delta T = 3600-4000$



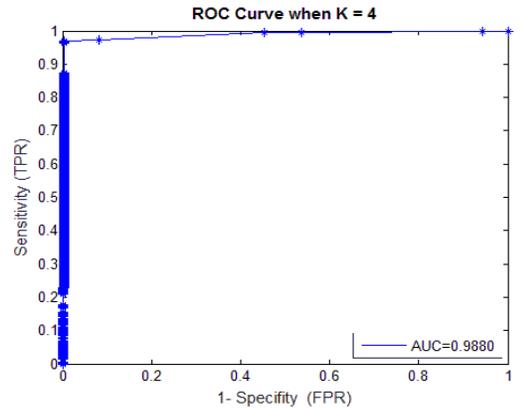
**Figure 81: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=1$**



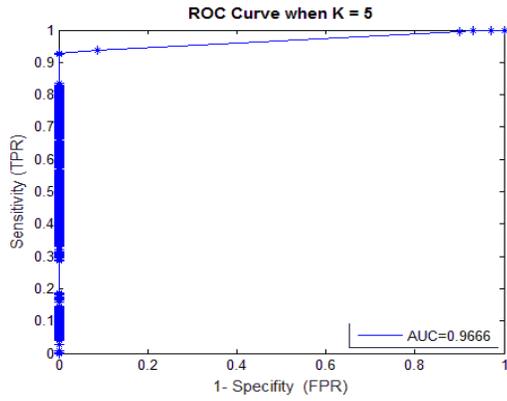
**Figure 82: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=2$**



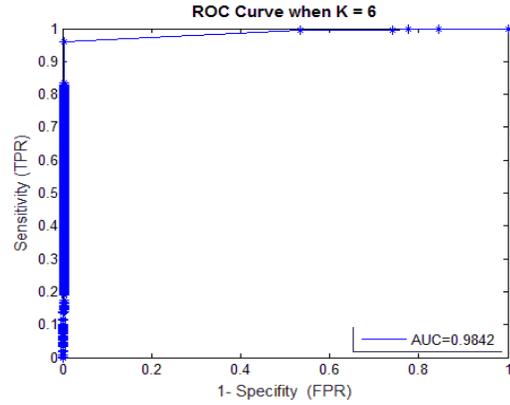
**Figure 83: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=3$**



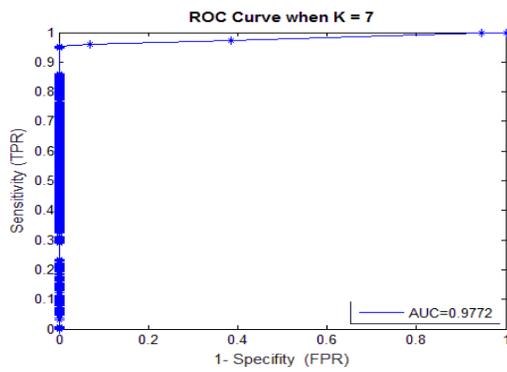
**Figure 84: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=4$**



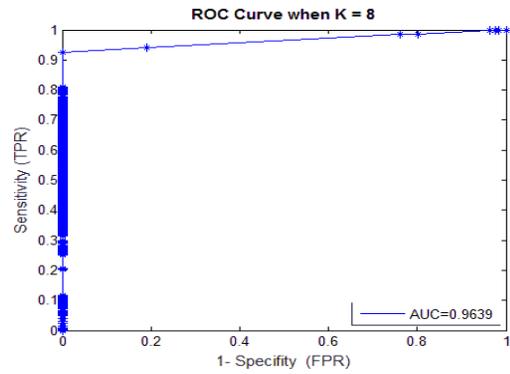
**Figure 85: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=5$**



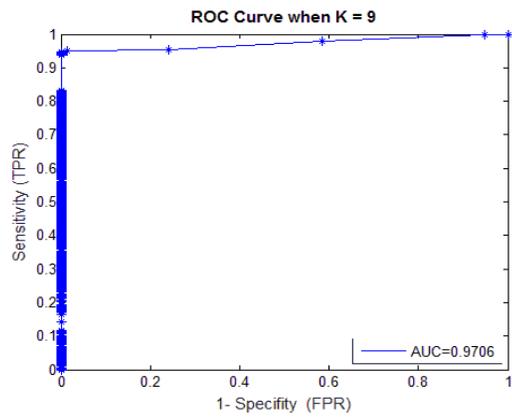
**Figure 86: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=6$**



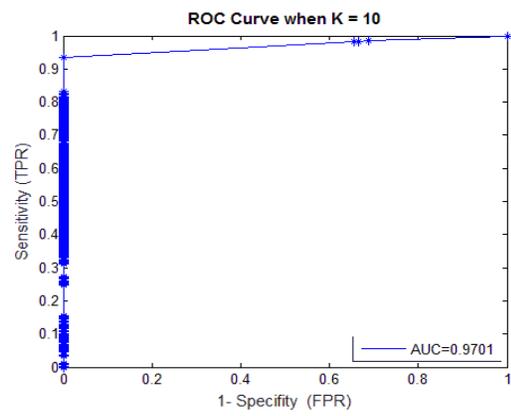
**Figure 87: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=7$**



**Figure 88: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and $K=8$**



**Figure 89: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and K=9**



**Figure 90: ROC Curve when $\Delta T =$
3600-4000 milliseconds, and K=10**

5.3.9. $\Delta T = 4100\text{-}4500$ Milliseconds:

In the ninth range where ΔT vary from 4100 – 4500 milliseconds. Ten-fold cross validation was used in the classification model. Plotting of ROC curves are showing in figures 98 - 107 as testing fold varying from **one** to **ten**. As we explained in section 5.2.1, AUC is the best way to summarize and compare ROC curves. Table 23 shows AUCs of figures 98 - 107.

Figure Number	K values (Fold #)	AUCs	Mean AUC
98	1	96.36282%	97.01 %
99	2	97.37852%	
100	3	95.96666%	
101	4	98.47551%	
102	5	96.63339%	
103	6	97.67813%	
104	7	98.00322%	
105	8	95.69785%	
106	9	96.64973%	
107	10	97.34479%	

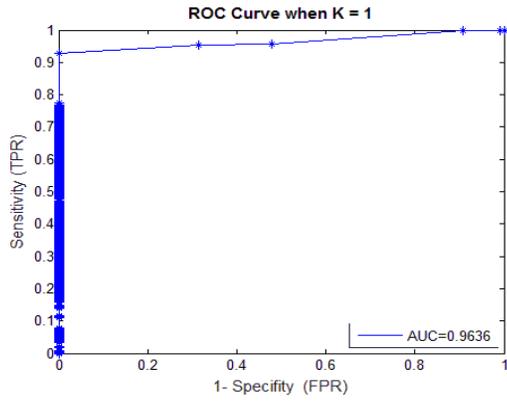
Table 23: Summary of AUCs for Ten-Folds where $\Delta T = 4100 - 4500$

AUC of all Ks in ROC curves is summarized with mean AUC (**97.01 %**). According to accuracy guide of AUC (see table 3) the classification model showed excellent performance in the range of $\Delta T = 4100 - 4500$ milliseconds.

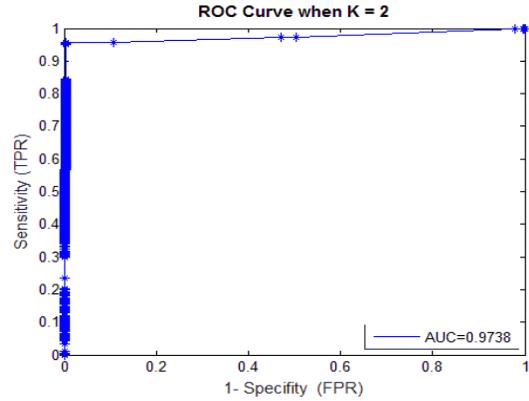
Confusion matrix used as another evaluation criteria. Equations in table 5 used to compute TP, FP, FN, and TN. According, we can use equations in table 6 to find: True Positive Rate (TPR), False Positive Rate (FPR), False Negative Rate (FNR), and True Negative Rate (TNR) as shown in table 24 where FBR is 0.14 % and that considered an excellent performance of the classification model with improvement from previous range and that fulfill our target. Both metric are almost on the same line

ΔT	TPR	FPR	FNR	TNR
4100-4500	94.11%	0.14%	5.89%	99.85%

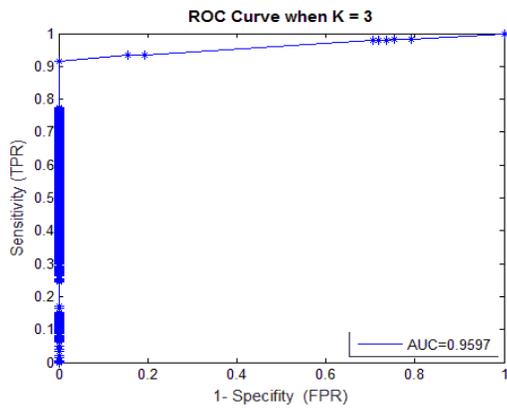
Table 24: TPR, FPR, FNR, and TNR where $\Delta T = 4100-4500$



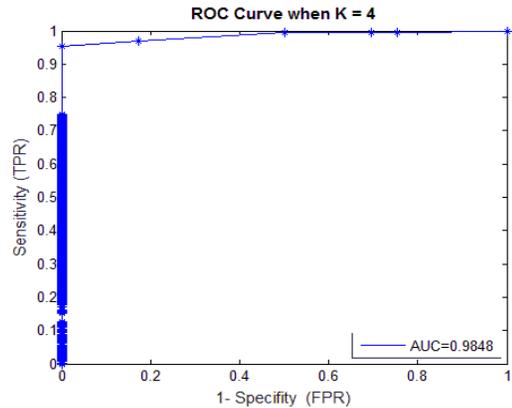
**Figure 91: ROC Curve when $\Delta T =$
4100-4500 milliseconds , and $K=1$**



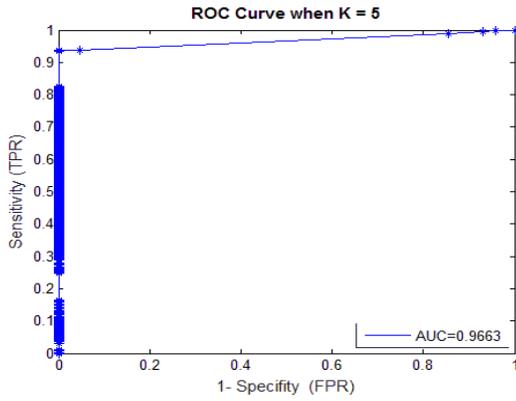
**Figure 92: ROC Curve when $\Delta T =$ 4100-
4500 milliseconds, and $K=2$**



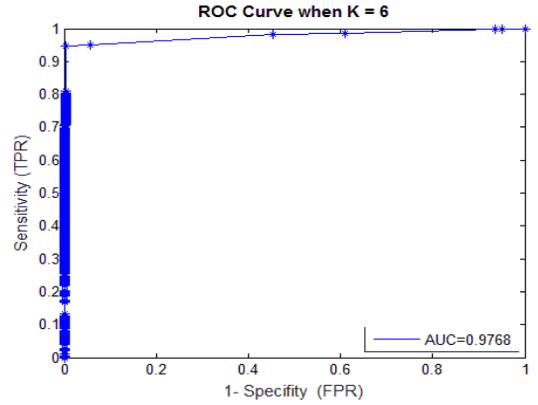
**Figure 93: ROC Curve when $\Delta T =$
4100-4500 milliseconds, and $K=3$**



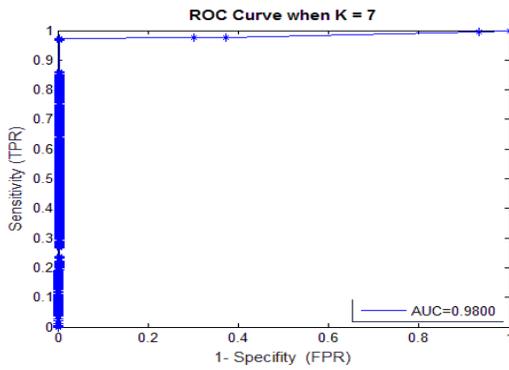
**Figure 94: ROC Curve when $\Delta T =$ 4100-
4500 milliseconds, and $K=4$**



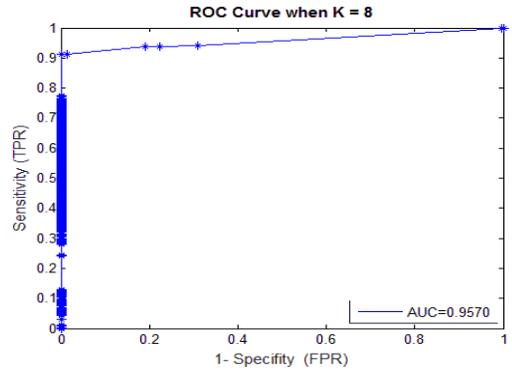
**Figure 95: ROC Curve when $\Delta T =$
4100-4500 milliseconds, and $K=5$**



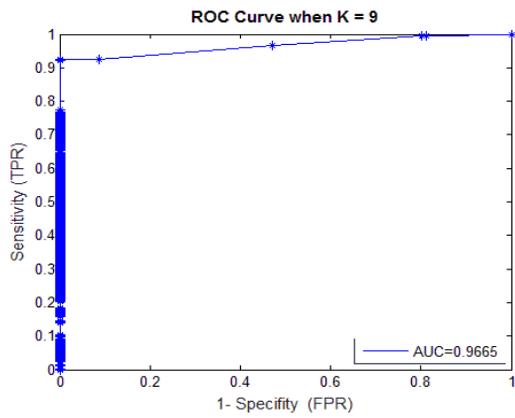
**Figure 96: ROC Curve when $\Delta T =$ 4100-
4500 milliseconds, and $K=6$**



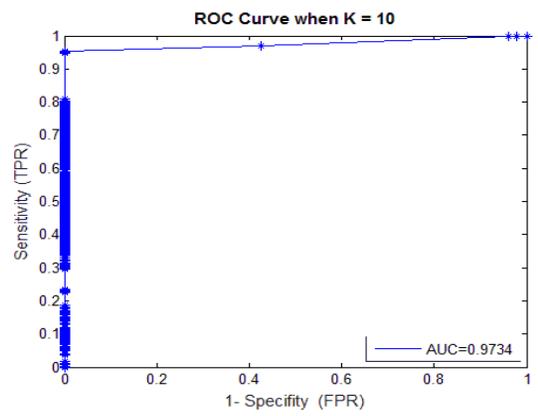
**Figure 97: ROC Curve when $\Delta T =$
4100-4500 milliseconds, and $K=7$**



**Figure 98: ROC Curve when $\Delta T =$ 4100-
4500 milliseconds, and $K=8$**



**Figure 99: ROC Curve when $\Delta T =$
4100-4500 milliseconds, and K=9**



**Figure 100: ROC Curve when $\Delta T =$
4100-4500 milliseconds, and K=10**

5.4. SUMMARY AND DISCUSSION:

In this section we will summarize and discuss experiments results showed in section 5.3. From results, we found that as ΔT increases, accuracy and performance of our classification model increase. Table 25 shows, the produced results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT vary from 100 milliseconds to 4500 milliseconds.

For ΔT in the ranges “100 -500”, and “600 – 1000” classification model failed for AUC of ROC curve and confusion matrix evaluation criteria. AUC showed failure percentage in both ranges. False Positive Rates was very high (nearly 100%).

For ΔT in the range “1100 -1500”, classification model performed poorly for both criteria. AUC showed poor performance and FPR showed a little bit improvement, but still considered to be high (nearly 40%).

For ΔT in the ranges “1600 -2000”, and “2100 – 2500” classification model showed good performance for both criteria. AUC in both ranges fall in the good category of AUC guide (see table 3). Comparing to above ranges FPR showed proportional improvement (nearly 11.32%).

For ΔT in the ranges “2600 -3000”, and “3100 – 3500” classification model showed excellent performance for both criteria. AUC in both ranges fall in the excellent category of AUC guide (see table 3). Comparing to above ranges FPR showed proportional improvement (6.91%).

For ΔT in the ranges “3600 -4000”, and “4100 – 4500” classification model showed very excellent performance for both criteria. AUC in both ranges fall in the excellent category of AUC guide (see table 3). For both ranges, FPR showed tremendous improvement (1.067 % and 0.14 %).

Range of ΔT	Confusion Matrix				ROC Curve
	TPR	FPR	FNR	TNR	AUC
100 - 500	99.8%	99.8%	0.18%	0.19%	60.31%
600 - 1000	97%	66.1%	3.02%	33.8%	51.92%
1100 - 1500	95.73%	38.2%	4.2%	61.8%	68.2%
1600 - 2000	95.05%	10.14%	4.94%	89.86%	89.2%
2100 - 2500	95.17%	12.51%	4.82%	87.48%	87.35%
2600 - 3000	94.95%	6.91%	5.04%	93.09%	91.33%
3100 - 3500	94.81%	6.91%	5.18%	94.53%	92.73%
3600 - 4000	94.48%	1.067%	5.528%	98.93%	96.63%
4100 - 4500	94.11%	0.14%	5.89%	99.85%	97.01%

Table 25: Summary of results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT vary from 100 milliseconds to 4500 milliseconds

As shown above, when ΔT increases behavior of malicious and benign processes can be distinguished by the classification model with very low false positive rate where the difference in activities becomes more obvious. In contrast, for small ΔT , behavior of malicious and benign processes cannot be distinguished and classification model showed poor performance and random classification. This confirm the theory we followed in this research that bot program installed on end computer will have different outgoing and incoming traffic behavior than benign program.

However, in order to pick up the exact smallest ΔT that achieves the lowest FBR (below 1%) we cannot depend on range of ΔT s where dataset will differ from range to another. Thus, results range will not be accurate. So, we tested random ΔT s (1000, 2000, and 3000 milliseconds) to just give us big picture from where to start finding smallest ΔT . As shown in table 26 smallest ΔT shall be around the 3000 milliseconds.

Confusion Matrix					ROC Curve	Con. Int
ΔT	TPR	FPR	TNR	FNR	AUC	-
1000	94.69 %	29.79 %	70.20 %	5.30 %	80.43 %	+/- 20.04
2000	95.13 %	3.56 %	96.43 %	4.86 %	95.53 %	+/- 5.07
3000	94.87 %	0.13 %	99.86 %	5.13 %	97.13 %	+/- 0.62
4000	94.30 %	0.14 %	99.85 %	5.69 %	96.54 %	+/- 0.88

Table 26: Results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT randomly selected (1000, 2000, 3000, and 4000)

We started from $\Delta T=2500$ millisecond till $\Delta T= 3300$ milliseconds as shown in table 27. According to the table, smallest ΔT that achieves the lowest FBR (below 1%) is 2800 milliseconds where FBR is 0.13% and AUC is 97.12%.

Confusion Matrix					ROC Curve	Con. Int
ΔT	TPR	FPR	TNR	FNR	AUC	-
2500	94.98%	3.34%	96.65%	5.01%	95.13%	+/- 5.40
2600	94.92%	3.33%	96.66%	5.07%	95.12%	+/- 5.38
2700	94.79%	3.30%	96.69%	5.20%	95.14%	+/- 5.41
2800	94.93%	0.13%	99.86%	5.06%	97.12%	+/- 0.59
2900	94.84%	0.13%	99.86%	5.15%	97.44%	+/- 0.8
3000	94.87 %	0.13%	99.86 %	5.13 %	97.13 %	+/- 0.62
3100	94.86%	0.13%	99.86%	5.13%	96.95%	+/- 0.7
3200	94.80%	0.13%	99.86%	5.19%	96.86%	+/- 0.658
3300	94.73%	0.13%	99.86%	5.26%	96.84%	+/- 0.613

Table 27: Results using AUC of ROC curve and confusion matrix evaluation criteria where ΔT vary from 2500 milliseconds to 3300 milliseconds

CHAPTER 6

CONCLUSION

6.1. SUMMARY

In this thesis, many of existing botnet detection techniques are reviewed. Most of these techniques are network-based and few of which were host-based botnet detection techniques. Most of reviewed host-based detection techniques rely on observing bot system calls, event logs and/or registry modifications.

In contrast, in this thesis, we introduced Host-based botnet detection technique relying on observing network traffic of bot in local machine. We developed special tool to capture outgoing and incoming network traffic and associate it with process in local machine. We named it PCTool. MATLAB code used to collect statistics of captured traffic. Statistics were used as main features to train RBF-SVM.

Confusion Matrix and **Area under the Curve (AUC)** were used as the classification performance measuring metrics. RBF-SVM classification model showed very excellent performance for both criteria. According to accuracy guide of AUC, the classification model showed excellent performance where $AUC = 97.01\%$. For the other metric, detection rate reaches 94.11% and false positive about 0.14%.

In experiments, ΔT vary from 100 milliseconds to 4500 milliseconds. We found that as ΔT increases behavior of malicious and benign processes can be distinguished by the classification model with very low false positive rate. In contrast, as ΔT decreased, behavior of malicious and benign processes cannot be distinguished and classification model showed poor performance and random classification.

6.2. FUTURE WORK

Many issues need more investigation in addition to what we done in this thesis as following:

- We used limited number of botnets to build our data set. Large dataset for various botnets needed to be built using PCTool, Such project will help researchers.
- Improve mitigation technique through whitelisting.

REFERENCES

- [1] E. Stinson and J. Mitchell, "Towards Systematic Evaluation of the Evadability of Bot/Botnet Detection Methods.," *WOOT*, 2008.
- [2] "Most spam generated by botnets, says expert | ZDNet." [Online]. Available: <http://www.zdnet.com/most-spam-generated-by-botnets-says-expert-3039167561/>. [Accessed: 18-Nov-2014].
- [3] "Net pioneer predicts overwhelming botnet surge - CNET News." [Online]. Available: http://news.cnet.com/Net-pioneer-predicts-overwhelming-botnet-surge/2100-7348_3-6154221.html. [Accessed: 18-Nov-2014].
- [4] E. Stinson, E. Stinson, J. C. Mitchell, and J. C. Mitchell, "Characterizing Bots' Remote Control Behavior," *DIMVA '07 Proc. 4th Int. Conf. Detect. Intrusions Malware, Vulnerability Assessment. Springer Berlin Heidelb.*, pp. 89–108, 2007.
- [5] J. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," *Proc. USENIX Steps to Reducing ...*, pp. 43–48, 2006.
- [6] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation," *Proc. first Conf. First Work. ...*, 2007.
- [7] M. Mahmoud, M. Nir, and A. Matrawy, "A Survey on Botnet Architectures, Detection and Defences," *ijns.femto.com.tw*, vol. 0, no. 0, pp. 1–18, 2013.

- [8] S. Sparks and C. C. Zou, “An Advanced Hybrid Peer-to-Peer Botnet,” *IEEE Trans. Dependable Secur. Comput.*, vol. 7, no. 2, pp. 113–127, Apr. 2010.
- [9] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, “Peer-to-peer botnets: overview and case study,” p. 1, Apr. 2007.
- [10] K. Chiang and L. Lloyd, “A case study of the rustock rootkit and spam bot,” p. 10, Apr. 2007.
- [11] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, “Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm.,” *LEET*, 2008.
- [12] J. Nazario, “Blackenergy ddos bot analysis,” *Arbor*, 2007.
- [13] J. Chandler, “Liability for botnet attacks.,” *Can. J. Law Technol.*, vol. 5, no. 1, pp. 13 – 25, 2006.
- [14] S. Balram and M. Wilscy, “User Traffic Profile for Traffic Reduction and Effective Bot C&C Detection.,” *IJ Netw. Secur.*, vol. 16, no. 1, pp. 46–52, 2014.
- [15] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley, “Detecting Botnets with Tight Command and Control,” in *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, 2006, pp. 195–202.
- [16] A. Karasaridis, B. Rexroad, and D. Hoeflin, “Wide-scale botnet detection and characterization,” ... *first Conf. First ...*, 2007.

- [17] G. Gu and W. Lee, "BotSniffer : Detecting Botnet Command and Control Channels in Network Traffic BotSniffer : Detecting Botnet Command and Control Channels," 2008.
- [18] W. Zhang, Y. J. Wang, and X. L. Wang, "A Survey of Defense against P2P Botnets," *2014 IEEE 12th Int. Conf. Dependable, Auton. Secur. Comput.*, pp. 97–102, 2014.
- [19] S. Khattak, N. R. Ramay, K. R. Khan, A. a. Syed, and S. A. Khayam, "A Taxonomy of botnet behavior, detection, and defense," *IEEE Commun. Surv. Tutorials*, vol. 16, no. 2, pp. 898–924, 2014.
- [20] L. Liu, S. Chen, G. Yan, and Z. Zhang, "Bottracer: Execution-based bot-like malware detection," *Inf. Secur.*, 2008.
- [21] M. M. Masud, T. Al-khaleeb, L. Khan, B. Thuraisinghatn, and K. W. Hamlen, "Flow-based identification of botnet traffic by mining multiple log files," *2008 1st Int. Conf. Distrib. Fram. Appl. DFmA 2008*, pp. 200–206, 2008.
- [22] M. Szymczyk, "Detecting Botnets in Computer Networks Using Multi-agent Technology," *2009 Fourth Int. Conf. Dependability Comput. Syst.*, pp. 192–201, 2009.
- [23] Y. Lin, Y. Zhang, and Y. Ou, "The Design and Implementation of Host-Based Intrusion Detection System," *2010 Third Int. Symp. Intell. Inf. Technol. Secur. Informatics*, pp. 595–598, 2010.

- [24] K. Xu, D. Yao, Q. Ma, and A. Crowell, "Detecting infection onset with behavior-based policies," *Proc. - 2011 5th Int. Conf. Netw. Syst. Secur. NSS 2011*, pp. 57–64, 2011.
- [25] S. Murugan and K. Kuppusamy, "System and methodology for unknown malware attack," *Int. Conf. Sustain. Energy Intell. Syst. (SEISCON 2011)*, no. 1, pp. 803–804, 2011.
- [26] M. Christodorescu, "Static Analysis of Executables to Detect Malicious Patterns *."
- [27] M. Alazab, S. Venkataraman, and P. Watters, "Towards Understanding Malware Behaviour by the Extraction of API Calls," *2010 Second Cybercrime Trust. Comput. Work.*, no. November 2009, pp. 52–59, 2010.
- [28] Y. Ye, D. Wang, T. Li, D. Ye, and Q. Jiang, "An intelligent PE-malware detection system based on association mining," *J. Comput. Virol.*, vol. 4, no. 4, pp. 323–334, 2008.
- [29] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic Analysis of Malicious Code," *J. Comput. Virol.*, vol. 2, no. 1, pp. 67–77, 2006.
- [30] F. Sun, T. Li, and H. Li, "Knowledge Engineering and Management," vol. 2012, no. Iske, p. 702, 2012.
- [31] G. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Secur. Priv.*, vol. 5, no. 2, pp. 32–39, 2007.

- [32] M. Egele, T. Scholte, E. Kirda, and S. Barbara, “A survey on automated dynamic malware analysis techniques and tools,” *ACM Comput. Surv.*, vol. V, pp. 1–49, 2011.
- [33] A. Somayaji and S. Forrest, “9th USENIX Security Symposium,” 2000.
- [34] A. Somayaji, “Operating system stability and security through process homeostasis,” no. July, 2002.
- [35] S. Stolfo, F. Apap, and E. Eskin, “A comparative evaluation of two algorithms for windows registry anomaly detection,” *J. Comput. ...*, 2005.
- [36] M. Akiyama, T. Kawamoto, M. Shimamura, T. Yokoyama, Y. Kadobayashi, and S. Yamaguchi, “A Proposal of Metrics for Botnet Detection Based on Its Cooperative Behavior,” in *2007 International Symposium on Applications and the Internet Workshops*, 2007, pp. 82–82.
- [37] C. Mazzariello, “IRC Traffic Analysis for Botnet Detection,” *2008 Fourth Int. Conf. Inf. Assur. Secur.*, pp. 318–323, Sep. 2008.
- [38] M. Roesch, “Snort: Lightweight Intrusion Detection for Networks.,” *LISA '99 13th Syst. Adm. Conf.*, pp. 229–238, 1999.
- [39] L. Zhuang, J. Dunagan, D. R. Simon, H. J. Wang, and J. Tygar, “Characterizing botnets from email spam records,” *Proc. 1st Usenix Work. Large-Scale Exploit. Emergent Threat.*, pp. 1–9, 2008.
- [40] “BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-

- Independent Botnet Detection.” [Online]. Available: https://www.usenix.org/legacy/event/sec08/tech/full_papers/gu/gu_html/index.html. [Accessed: 20-Dec-2014].
- [41] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, “Botnet Detection Based on Network Behavior,” *Botnet Detect.*, vol. 36, no. August, pp. 1–24, 2008.
- [42] T. F. Yen and M. K. Reiter, “Traffic aggregation for malware detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5137 LNCS, pp. 207–227, 2008.
- [43] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “BotHunter: detecting malware infection through IDS-driven dialog correlation,” *USENIX Secur. '07 Proc. 16th USENIX Secur. Symp.*, p. 12, 2007.
- [44] W. Lu, M. Tavallaee, and A. a. Ghorbani, “Automatic discovery of botnet communities on large-scale communication networks,” *Proc. 4th Int. Symp. Information, Comput. Commun. Secur. - ASIACCS '09*, p. 1, 2009.
- [45] “An Algorithm for Anomaly-based Botnet Detection.” [Online]. Available: https://www.usenix.org/legacy/event/sruti06/tech/full_papers_old/binkley/binkley_html/srutibot.html. [Accessed: 24-Dec-2014].
- [46] S.-K. Noh, J.-H. Oh, J.-S. Lee, B.-N. Noh, and H.-C. Jeong, “Detecting P2P botnets using a multi-phased flow model,” in *Digital Society, 2009. ICDS'09. Third International Conference on*, 2009, pp. 247–253.

- [47] A. Ramachandran, N. Feamster, and D. Dagon, “Revealing botnet membership using DNSBL counter-intelligence,” *Proc. 2nd USENIX Steps to ...*, 2006.
- [48] R. Villamarín-Salomón and J. C. Brustoloni, “Identifying botnets using anomaly detection techniques applied to DNS traffic,” *2008 5th IEEE Consum. Commun. Netw. Conf. CCNC 2008*, no. 1, pp. 476–481, 2008.
- [49] J. Zhang, Y. Xie, F. Yu, D. Soukal, and W. Lee, “Intention and Origination: An Inside Look at Large-Scale Bot Queries.,” *Ndss*, 2013.
- [50] F. Yu, Y. Xie, and Q. Ke, “SBotMiner,” *Proc. third ACM Int. Conf. Web search data Min. - WSDM '10*, p. 421, 2010.
- [51] D. Dagon, “Botnet Detection and Response The Network is the Infection,” *OARC Work.*, 2005.
- [52] A. Schonewille and D. van Helmond, “The domain name service as an IDS,” *Res. Proj. Master Syst. ...*, pp. 1–24, 2006.
- [53] R. Villamarín-salomón and J. C. Brustoloni, “Identifying botnets using anomaly detection techniques applied to DNS traffic,” *... Conf. 2008. CCNC ...*, no. 1, pp. 476–481, 2008.
- [54] R. Perdisci, W. Lee, and N. Feamster, “Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces,” *USENIX Symp. Networked Syst. Des. Implement.*, 2010.

- [55] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda, “Automatically generating models for botnet detection,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5789 LNCS, pp. 232–249, 2009.
- [56] C. Rossow and C. J. Dietrich, “P R O V E X : Detecting Botnets with Encrypted Command and Control Channels,” pp. 21–40, 2013.
- [57] C. Livadas, B. Walsh, D. Lapsley, and T. Strayer, “Using Machine Learning Techniques to Identify Botnet Traffic,” no. 1.
- [58] F. Constantinou and P. Mavrommatis, “Identifying known and unknown peer-to-peer traffic,” *Proc. - Fifth IEEE Int. Symp. Netw. Comput. Appl. NCA 2006*, vol. 2006, pp. 93–100, 2006.
- [59] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, “Transport layer identification of P2P traffic,” *IMC '04 Proc. 4th ACM SIGCOMM Conf. Internet Meas.*, pp. 121–134, 2004.
- [60] J. Votano, M. Parham, and L. Hall, *Network characterization for botnet detection using statistical-behavioral methods*. 2004.
- [61] D. Liu, Y. Li, Y. Hu, and Z. Liang, “A P2P-botnet detection model and algorithms based on network streams analysis,” *2010 Int. Conf. Futur. Inf. Technol. Manag. Eng. FITME 2010*, vol. 1, pp. 55–58, 2010.

- [62] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo, “Detecting stealthy P2P botnets using statistical traffic fingerprints,” *Proc. Int. Conf. Dependable Syst. Networks*, pp. 121–132, 2011.
- [63] M. Iliofotou, M. Mitzenmacher, and G. Varghese, “Network Monitoring using Traffic Dispersion Graphs (TDGs),” *IMC '07 Proc. 7th ACM SIGCOMM Conf. Internet Meas.*, no. c, pp. 315–320, 2007.
- [64] J. François, S. Wang, T. Engel, R. State, and T. Engel, “BotTrack: Tracking Botnets Using NetFlow and PageRank,” *Netw. 2011*, vol. 6640, pp. 1–14, 2011.
- [65] N. Jiang, J. Cao, Y. Jin, L. E. Li, and Z. L. Zhang, “Identifying suspicious activities through DNS failure graph analysis,” *Proc. - Int. Conf. Netw. Protoc. ICNP*, pp. 144–153, 2010.
- [66] D. T. Ha, G. Yan, S. Eidenbenz, and H. Q. Ngo, “On the effectiveness of structural detection and defense against P2P-based botnets,” *Proc. Int. Conf. Dependable Syst. Networks*, pp. 297–306, 2009.
- [67] D. Ramsbrock, X. Wang, and X. Jiang, “A first step towards live botmaster traceback,” *Recent Adv. Intrusion Detect.*, pp. 59–77, 2008.
- [68] Z. Chi and Z. Zhao, “Detecting and Blocking Malicious Traffic Caused by IRC Protocol Based Botnets,” *2007 IFIP Int. Conf. Netw. Parallel Comput. Work. (NPC 2007)*, pp. 485–489, Sep. 2007.

- [69] Y. Zhang and V. Paxson, "Detecting stepping stones," *Proc. 9th USENIX Secur.*, no. August, pp. 1–11, 2000.
- [70] X. Wang, D. Reeves, and S. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," *Comput. Secur. 2002*, pp. 1–20, 2002.
- [71] A. Blum, D. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," *Recent Adv. Intrusion Detect. Springer Berlin Heidelb.*, pp. 258–277, 2004.
- [72] K. Yoda and H. Etoh, "Finding a connection chain for tracing intruders," *Comput. Secur. 2000*, pp. 191–205, 2000.
- [73] P. Wang, S. Sparks, and C. Cou, "An advanced hybrid peerto- peer botnet," *1st Work. Hot Top. Underst. Botnets*, p. 2, 2008.
- [74] P. Wang, L. Wu, B. Aslam, and C. C. Zou, "A systematic study on peer-to-peer botnets," in *Computer Communications and Networks, 2009. ICCCN 2009. Proceedings of 18th Internatonal Conference on*, 2009, pp. 1–8.
- [75] C. R. Davis, J. M. Fernandez, S. Neville, and J. McHugh, "Sybil attacks as a mitigation strategy against the storm botnet," in *Malicious and Unwanted Software, 2008. MALWARE 2008. 3rd International Conference on*, 2008, pp. 32–40.
- [76] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal Least Squares Learning

Algorithm,” vol. 2, no. 2, pp. 302–309, 1991.

- [77] R. Kohavi, “A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection,” 1995.
- [78] U. M. Braga-neto and E. R. Dougherty, “Is cross-validation valid for small-sample microarray classification ?,” vol. 20, no. 3, pp. 374–380, 2004.
- [79] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Inf. Process. Manag.*, vol. 45, no. 4, pp. 427–437, 2009.
- [80] M. Vuk, “ROC Curve , Lift Chart and Calibration Plot,” *Metod. Zv.*, vol. 3, no. 1, pp. 89–108, 2006.
- [81] D. J. Hand, “Measuring classifier performance: A coherent alternative to the area under the ROC curve,” *Mach. Learn.*, vol. 77, no. 1, pp. 103–123, 2009.

VITAE

- Name: Ahmed Mansoor Ayedh Al-Ameri
- Nationality: Yemen
- Date of Birth: May 19, 1986
- Email: al3miry@gmail.com
- Permanent Address: Hadramout, Yemen
- Educational Qualification:

M.S. IN COMPUTER NETWORKS

May, 2016

King Fahd University of Petroleum and Minerals

Dhahran, Saudi Arabia

B.S. IN COMPUTER INFORMATION SYSTEM

July, 2009

Al-Ahgaff University

Mukalla, Yemen