

**FRAMEWORK TO DETECT AND MITIGATE UNTRACED
POLYMORPHIC SHELLCODE FOR INTRUSION DETECTION AND
PREVENTION SYSTEMS**

A Thesis

**SUBMITTED TO THE
TILAK MAHARASHTRA VIDYAPEETH, PUNE**

**FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY**

In Computer Science

Under the Board of Modern Sciences & Professional Skills



BY

**Mrs. NAVNEET KAUR POPLI
(25315008509)**

UNDER THE GUIDANCE OF

Dr. ANUP GIRDHAR

DEPARTMENT OF COMPUTER SCIENCE

2019

CERTIFICATE BY THE SCHOLAR

This is to certify that the thesis titled, “Framework to detect and mitigate untraced polymorphic shellcode for Intrusion Detection and Prevention Systems” submitted by Mrs. Navneet Kaur Popli, under the supervision of Dr. Anup Girdhar, P.R.N. Number 25315008509 for the award of Ph.D. degree of Computer science from ‘Tilak Maharashtra Vidyapeeth’ University embodies my original work and has not formed the basis for the award of any degree, diploma, fellowship, titles in this or any other university or any other similar institutions of higher learning.

Date:

Signature

Mrs. Navneet Kaur Popli,

Research Scholar, TMV.

Enrollment Number 25315008509

CERTIFICATE BY THE SUPERVISOR

This is to certify that the work incorporated in the thesis “Framework to detect and mitigate untraced polymorphic shellcode for Intrusion Detection and Prevention Systems” submitted by Mrs. Navneet Kaur Popli is an original piece of research work carried out by the candidate under my supervision. Literary presentation is satisfactory and the thesis is in a form suitable for publication. Work evinces the capacity of the candidate for critical examination and independent judgment. Such materials as has been obtained from other sources have been duly acknowledged in the thesis.

Date:

Signature

Dr. Anup Girdhar, Ph.D

Research Guide, TMV

DECLARATION

I hereby declare that this Ph.D. thesis entitled “Framework to detect and mitigate untraced polymorphic shellcode for Intrusion Detection and Prevention Systems” was carried out by me for the degree of Doctor of Philosophy in Computer Science under the guidance and supervision of Dr. Anup Girdhar, Tilak Maharashtra Vidyapeeth, Pune, India. The interpretations put forth are based on my own experimentation and understanding. The books, articles and websites, which I have made use of are acknowledged at the respective place in the text. All content is plagiarism free. All the information gained is from genuine sources and the results are obtained following the detailed procedure. For the present thesis, which I am submitting to the University, no degree or diploma or distinction has been conferred on me before, either in this or in any other University.

Date:

Signature

Mrs. Navneet Kaur Popli,

Research Scholar, TMV.

Enrollment Number 25315008509

ACKNOWLEDGEMENTS

I wish to sincerely thank all those who have contributed in one way or another to this study. Words can only inadequately express my deep gratitude to my guide, **Dr. Anup Girdhar**, for his meticulous care, kindness and generosity. His fruitful comments and insightful suggestions have been a crucial formative influence on the present study. He has supported me in every possible way since the beginning of my research. His critical and careful reading of my writing has saved me from a lot of errors. Without his guidance and encouragement, my research would have never come out in the present form. I have seen in him an unpretentious and devoted scholar. Furthermore, it has been a memorable and enjoyable experience for me to work with him. Sincere thanks to all my family members for cooperating with me and helping me at every juncture. A special thanks to my mother-in-law **Mrs. Bhupinder Kaur Popli** for helping me in this treacherous journey. I wish to express my sincere gratitude to **Dr. Sunanda Yadav**, the Director of the place of research, for her expert guidance and invaluable suggestions. Grateful acknowledgments are also due to my student **Mr. Varnit Goyal** for helping me in the research work. Special thanks are due to the **Mr. G.L. Manchanda, Mr. Abhilash Varghese**, Assistant Directors, ISACA and the entire fraternity at ISACA (Information Systems Audit and Control Association)-Delhi Chapter, for giving me an opportunity to attend their security and auditing conferences, both as a speaker and as a participant. A lot of information was obtained regarding the security threats faced by the industry today and the various steps taken by the security personnel to mitigate these threats and risks.

Mrs. Navneet Kaur Popli,

Research Scholar, TMV.

ABSTRACT

The Internet world is ridden with all kinds of malware and hackers are constantly trying out new innovative techniques to evade detection mechanisms. When shellcodes became the choice of attack, antiviruses were quick to come up with signature detection mechanisms to catch them. Hackers went one step ahead and created the so called 'Polymorphic shellcodes'. These shellcodes have different signatures for the same shellcode. Thus it becomes impossible for a signature detection system to detect them. In our study, we are going to create a framework for the detection and mitigation of untraced polymorphic shellcode.

In May 2017, when WannaCry ransomware attacked, 150 countries with 2,00,000 computers all over the world were affected and some \$50,000 was already paid to the attackers by various companies and individuals. The base of this ransomware was shellcode. In early 2017 when Donald Trump won the US elections, power shells were injected in mails which were sent to Hillary Clinton's aide John Podesta. Once the shellcode entered the machine, it infected all the systems in the network and exposed all emails of Hillary, which caused a great reputational loss to her campaign and she paid the price by losing the elections. In 2016, the game PokemonGo was found to be carrying shellcodes which caused clicking frauds and millions of dollars were earned by the attackers by gaining root access to the phones and clicking all kinds of ads and sometimes stealing network bandwidth and sensitive data from the compromised machine. HummingBad, another kind of shellcode malware, affected ten million phones worldwide and got admin access to the phone and used that for generating fraudulent advertising revenue up to \$300,000 per month -- through the forced downloading of apps and clicking of ads.

Shellcodes are tiny programs which look like legitimate code and are inserted in code-caves in a program code. Once executed, they provide a shell to the hacker with root privileges. Eg.'Win32/ShellCode.gen!V' is a shellcode exploit for the Windows system. The most common shellcode instruction is to execute a shell such as /bin/sh, or cmd.exe. The only possible reason for launching such commands is to take control or exploit a compromised machine. When the exploit code causes what would normally be a critical error in the targeted program, the program jumps to the shellcode and is tricked into executing the attacker's commands. There are many harmful effects of shellcode. A shellcode can connect itself with internet, can display promotional adverts and fake messages, can corrupt secured system programs and files and can affect internet speed and system performance. It can install more malwares and fake programs, can show unwanted pop-ups, can crash the system, can help view others desktop, can sniff data from the network, can dump password hashes or use the owned device to attack hosts deeper into the network.

Shellcodes have typical structures which can be detected through pattern - matching by the IDPS software. Thus hackers have come up with a new type of shellcodes called 'Polymorphic Shellcodes'. Some algorithms are executed on the shellcodes which make them 'look' very different, say by changing their commands, encryption, code transposition, dead code insertion, register reassignment etc. However, the polymorphic shellcode still performs the same functionality that the original shellcode did. Pattern matching fails here and increases worries of IDPS and Antivirus developers.

We basically have to design a framework which can attack a shellcode from four different angles. We have to do a static analysis of shellcode, then a dynamic (behavior) analysis, then study its network footprint and finally try to see if it evades the sandbox. Then we aggregate the result obtained from these four stages and give the final result. It is imperative that a check is put on malicious activities. The biggest threat, cyber world is facing today is that of 'Polymorphic Shellcodes'. These are shellcodes which are polymorphic in nature, meaning that they change their look but have the same behavior. This makes them difficult to detect by Signature based detection systems. Some polymorphic shellcodes are also capable of changing their behavior at runtime. Thus Anomaly based detection systems also fail here.

All the current solutions which are existing today have some lacunas in common:

1. Signature Matching-Most of them focus on malware detection by signature matching and pattern recognition. Malware authors are now smarter than ever before and signature detection is of no use due to techniques like polymorphism, metamorphism etc.
2. No behavioral analysis- Some of them do not take into consideration, the behavior of malware, like file behavior, network behavior and other dynamic behavior of the file to be analyzed.
3. Updating time-Another problem that current antiviruses face is that they take a lot of time to analyze the malware and then update the definition of antiviruses into user's device.
4. Danger to user's privacy- The antivirus companies are sometimes a danger to the user's privacy as they collect data from user on regular basis and use that to make money. The normal user acts only as a data feeder so that these antivirus companies can protect enterprises.
5. Centralized Antivirus Companies-All the antivirus organizations are centralized in nature. This implies that a lot of computation power is required and very few computers are available to provide it. If a distributed system could be designed where all systems in the network contribute to the work of malware detection, things will become faster and more efficient.

Note that the recent ransomware attacks by WannaCry and Petya can prove the above statements, not to mention that none of them were actually polymorphic or metamorphic.

Benefit of the Study

Our proposed model will take care of all kinds of polymorphic shellcodes. It consists of the snapshot technique and the revert-back model. It takes snapshots of the memory and processes and keeps them in the database to understand the working of the malware. Next time when the same malware shows up, we have the list of its behaviors and thus can easily detect it. After that we restore the memory and the processes back to their original state. Machine learning and artificial intelligence are further incorporated to make the work more efficient and detect any new malware also. Decentralized currency was another incredible innovation recently and it led us to the new system of bitcoins. Bitcoins are great but what makes them greater is the technology on which it works. The technology is known as blockchain and blockchain has so many other applications other than just decentralized currency. Here we use the blockchain concept to harness distributive power of all the systems in the network so that a large amount of computing power is gained with very little cost and everyone in the network is benefitted.

Our work is highly significant both in the present and in the future. Hackers have recently used shellcodes in WannaCry and Petya Malwares. The next step is to use polymorphic shellcode attack. We must be ready for them and our works makes us ready. Also we are proposing a framework of policies for Intrusion Detection and Prevention Systems against such malware to harden them so that they can tackle any kind of threat.

Conceptual Model Framework

Our conceptual model framework is a design created with the aim to identify and remove risks of polymorphic shellcodes. It uses advance machine learning algorithms for detection. It is going to attack a shellcode, in fact any type of malware from 4 sides-static, behavior, network and sandbox evasion:

- 1) Static analysis: Means that we will study the signature of the sample and match it with already given signatures in our existing databases. If the signature matches with a malware we calculate the static probability of the sample for maliciousness. The machine learning model will be trained and tested with huge, ever increasing data.

- 2) Behavior Analysis: There are advance malware like polymorphic and metamorphic malware which change their signatures quite often. In that case, we generally study the behavior of the sample to see if it matches with the behavior that normally malware families show. If it does, we again calculate the behavior probability of the sample for maliciousness.
- 3) Network Analysis: Now we will see the network behavior of the sample to match it with the alerts that network monitors generate. If we see significant number of alerts, we again calculate the network probability of the sample for maliciousness.
- 4) Sandbox Evasion Detection: Normally, an intrusion detection system will run a doubtful program in a controlled environment which is normally a sandbox. This environment will have all the resources required got the sample to run and all permissions are also given to the file. However, it is not allowed to run in an actual system, it is only run in a virtual system. This is a smart way to know the true characteristics of the sample and judge whether the file is malicious or benign before it enters the actual system environment. However, there are advance malware which are created to stay dormant whenever they detect that they are running in a virtual environment. There are many techniques which are applied for the detection of sandbox. Thus they try to evade a sandbox whenever they experience one. Therefore it is with surety one can declare a sample as a malware if it tries to evade a sandbox. This detection will give us a Boolean value of 0 or 1.
- 5) Final Detection: From all the above four steps, we are going to get four probabilities. An average of these four probabilities is calculated to get the final detection probability. A threshold probability is calculated for the model according to the amount and variance of the data it currently has. A sample is declared as malicious if its combined average probability is above the threshold probability of the model. It is benign if the probability is lower. However, if the probability is equal to the threshold value, the sample is termed as unknown and is sent to the model again for further analysis. The model is cyclic in nature and every new sample is used as data for training of the model to make it more effective. Therefore it will always be in an updated state. It will have information about all new malwares all the time.
- 6) Decentralized Environment: The whole model is going to work in a decentralized environment. Thus the model is very resource efficient. Also because the detection mechanism is completely decentralized,

therefore, every node is master node in the network and there is no central authority which can work for selfish gains or unrealistic profits.

- 7) Blockchain network: All the transactions for detection of malware are logged in Blockchain network and the data about the malwares which is stored in each node is stored in the form of blocks in the block chain. Thus this database is immutable, secure, reliable, distributed and always updated. Thus every node participating in the malware detection model gets an always updated data at a very cost effective rate with the latest of machine learning technology.
- 8) Balancing the model: the model has to be always in a balanced state so that is never biased with more of clean or more of malicious data. Whenever we find such a bias in the model according to our calculations, we either increase the clean samples or malicious samples for training of the model.

Objectives of the Framework:

- 1) To identify and the mitigate risk factors occurring due to the execution of polymorphic shellcode.
- 2) Injecting the binaries in the code caves of existing (PE) application by using binding, embedding and stubs methodology. This will create polymorphic shellcodes for our study.
- 3) Understanding polymorphic and metamorphic techniques of creating advanced malware.
- 4) Understanding the working of polymorphic shellcode including the attack methodology, hiding methodology replication methodology and metamorphosis.
- 5) Working with live ransomware samples to understand their behavior and attack methodology.
- 6) The static analysis, behavior analysis, network analysis and sandbox evasion detection of polymorphic shellcode on the host as well as on the network.
- 7) Proposing the framework for the detection and mitigation of untraced polymorphic shellcodes using advanced machine learning techniques.
- 8) Proposing how this framework would work best in a decentralized environment.
- 9) Proposing blockchain network usage for recording and logging all transactions to make them secure, updated and reliable.
- 10) Training the framework with already known samples.
- 11) Testing the framework with live samples from the wild.
- 12) Analysis of the proposed framework with its limitation and future scope.

- **Static analysis** -static analysis here refers to the act of extracting information based on file properties without running it. This is the quickest way to classify the file but not always accurate. We have extracted a total of 52 parameters using a python module called **PE Analyzer 2**.
- **Behavior analysis** -Behavior analysis refers to the act of extracting information at runtime. We have extracted api call graphs, files created and affected at runtime etc. We did this using a very powerful tool '**Volatile Framework**'.
- **Packet analysis** - In this module we are doing the analysis of network traffic of a particular file using **tcpdump** and **snort**.
- **Sandbox Evasion Analysis** – If the sample is trying to evade the sandbox, we consider that this is a sort of malware behavior.

In our approach, we first use supervised learning. We have a machine learning algorithm. It would be given some binary files as a training data set. Although most of our dataset will be contributed by the users of network, we have provided some initial dataset which comprises of malicious data from virus-share database , all .exe files from clean windows installation and .exe files of popular software from filehippo database as a clean source of data . After getting trained, the algorithm would be given an unknown sample. Through the knowledge it has attained, it would be able to recognize the malicious file. The algorithm is fine-tuned with better training samples.

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION

1.1 Introduction to the Study.....	1
1.2 Malware Detection Mechanism.....	2
1.2.1 Signature Based Systems.....	3
1.2.2 Anomaly Based Systems.....	3
1.2.3 Emulation/Behavior Based Systems.....	3
1.3 Types of Malicious Attacks	6
1.3.1 Viruses.....	6
1.3.2 Worms.....	6
1.3.3 Trojans.....	7
1.3.4 Rootkits.....	7
1.3.5 Botnets.....	7
1.3.6 Ransomware.....	7
1.3.7 Zombies.....	8
1.3.8 Rats.....	8
1.3.9 Spyware.....	8
1.3.10 Shellcodes.....	8
1.3.11 Polymorphic Shellcodes.....	9
1.3.12 Metamorphic Shellcodes.....	10
1.3.13 Advance Malware-Beyond Polymorphic and Metamorphic Shellcodes.....	11
1.4 Shellcode Infection Mechanisms.....	12
1.4.1 Buffer Overflow.....	13
1.4.2 Code Injection	13
1.5 Machine Learning used for Malware Detection.....	13
1.5.1 Introduction to Machine Learning.....	13
1.5.2 Reasons for Using Machine Learning for Malware Detection.....	14
1.5.3 Supervised v/s Unsupervised Machine Learning.....	14
1.6 Motivation for the Study.....	14
1.7 Scope and Contribution of the study.....	15
1.8 Research Methodology used in the Study.....	16
1.9 Structure of the Thesis.....	17
1.10 Keywords used.....	19

CHAPTER 2: LITERATURE REVIEW

2.1 Analysis of Literature Survey.....	21
2.2 Malware detection techniques.....	22

- 2.3 The various threats and Working of common antimalware solutions..... 22
- 2.4 Concept of CVE’s..... 23
- 2.5 Polymorphic and metamorphic malware..... 23
- 2.6 Machine learning for malware detection..... 24
- 2.7 Working of emulators like Cuckoo..... 24
- 2.8 Studying static and dynamic behavior of malware..... 25
- 2.9 Studying various ransomware..... 26
- 2.10 Studying distributive technologies like Hadoop..... 26
- 2.11 Understanding cloud architecture and working..... 26
- 2.12 Understanding Blockchains..... 27
- 2.13 Studying digital currencies like Bitcoins and Ethereum..... 27
- 2.14 Gap Analysis..... 28

CHAPTER 3: RESEARCH METHODOLOGY

- 3.1 Introduction: Research Approach..... 32
- 3.2 Method of Data Collection..... 34
- 3.3 Research Methodology used..... 36
- 3.4 Analysis Report of data collected..... 42
 - 3.4.1 Type of organization 43
 - 3.4.2 Location of organization..... 43
 - 3.4.3 Title level held in the organization 44
 - 3.4.4 Degree of security responsibility 45
 - 3.4.5 Types of malicious activities 46
 - 3.4.6 Origin of malicious activity 47
 - 3.4.7 Percentage of Overall budget allocated to cyber security 48
 - 3.4.8 Level till which devices get affected 49
 - 3.4.9 Level of concern for cyber security 50
 - 3.4.10 Technologies used for protection 51
 - 3.4.11 Approach followed for malware detection 52
 - 3.4.12 Resource allocation to cyber security 53
 - 3.4.13 Response time after malicious attack 54
 - 3.4.14 Factors hindering cyber security efforts in the organization 55
- 3.5 Key Understandings..... 56
- 3.6 Objectives of the Study..... 57
 - 3.6.1 The Theoretical Framework of the Present Study..... 57
 - 3.6.2 Conceptual Model Framework..... 57
 - 3.6.3 Objectives of the Framework..... 60
- 3.7 Experiments to assess Polymorphic shellcode threat..... 60
 - 3.7.1 Creating a shellcode by Smashing the Stack..... 60
 - 3.7.2 Injecting a polymorphic shellcode in PE file..... 68
 - 3.7.3 Getting privileges after infecting a file with polymorphic shellcode..... 72

3.8 Significance of Research.....	76
3.8.1 Need of the Study.....	76
3.8.2 Benefit of the Study.....	77
3.9 Designing PosDeF.....	78
3.9.1 Design Methodology.....	78
3.9.2 Objective.....	78
3.9.3 PosDeF Design.....	79
3.9.4 Proposed Working of PosDeF.....	80
3.9.4.1 Collection of Data for the Formation of the Training Set.....	83
3.9.4.2 Training with our initial dataset.....	83
3.9.4.3 Profiling the files.....	84
3.9.4.4 Convert the dataset into machine learning compatible format	84
3.9.4.5 Pre train the model.....	84
3.9.4.6 Train the dataset using various machine Learning Algorithms...	85
3.9.4.7 Iteratively scan all available files.....	86
3.9.4.8 Reward the user.....	86
3.9.4.9 Balancing the dataset.....	87
3.9.4.10 Endless loop.....	87
3.10 Algorithms Used for Building PosDeF.....	87
3.10.1. The Final Algorithm.....	87
3.10.1.1 Algorithm for Training of PosDeF.....	88
3.10.1.2 Algorithm for Threshold Calculation for the Final Framework during the training phase.....	89
3.10.1.3 Algorithm for Testing of PosDeF.....	91
3.10.2 Static Analysis.....	92
3.10.2.1 Algorithm for Static Training.....	92
3.10.2.2 Algorithm for finding out the Best Classification Algorithm for Static Testing.....	93
3.10.2.3 Algorithm for Static Testing.....	94
3.10.3 Behavior Analysis.....	94
3.10.3.1 Algorithm for Behavior Training.....	95
3.10.3.2 Algorithm for Finding Best Classification Algorithm for Behavior Testing.....	96
3.10.3.3 Algorithm for Behavior Testing.....	96
3.10.4 Snort Analysis.....	97
3.10.4.1 Algorithm for Snort Training.....	97
3.10.4.2 Algorithm for Finding out the Best Classification Algorithm for SnortTesting.....	98
3.10.4.3 Algorithm for Snort Testing.....	98
3.10.5 Algorithm for Sandbox Evasion.....	99

CHAPTER 4: ANALYSIS AND INTERPRETATION

4.1 Creation of Polymorphic and Metamorphic Shellcodes.....	102
4.1.1 Obfuscation of NOP sled.....	102

4.1.2 Obfuscation of the shellcode.....	105
4.2 Using the Polymorphic Shellcode for Attack.....	108
4.2.1 Launching a multi-staged attack.....	108
4.2.2 Sandbox evasion techniques.....	111
4.2.3 Polymorphic blending.....	111
4.2.4 Conversion to metamorphic code.....	112
4.3 Working with Live Ransomware Samples- WannaCry and Petya.....	112
4.3.1 Recent Impact of Ransomwares.....	112
4.3.1.1 WannaCry.....	113
4.3.1.2 Petya.....	113
4.3.2 Reasons of Attack of Ransomware.....	114
4.3.3 Behavior Analysis of WannaCry and Petya.....	115
4.3.4 Working of WannaCry.....	119
4.3.4.1 Components of WannaCry.....	122
4.3.4.1.1 DoublePulsar.....	122
4.3.4.1.2 TOR.....	123
4.3.4.1.3 Domain Check.....	123
4.3.4.1.4 Bitcoin Wallets.....	124
4.3.5 Precautions to be safe from ransomware.....	124
4.3.6 Remedy after ransomware attack.....	125
4.4 Building the Framework.....	125
4.4.1 Building the Static Part of the Framework.....	125
4.4.1.1 Feature Extraction.....	126
4.4.1.2 Feature Selection.....	128
4.4.1.3 Feature Classification.....	129
4.4.2 Building the Behavior Part of the Framework.....	132
4.4.2.1 File Analyzed in Cuckoo Sandbox.....	133
4.4.2.2 JSON Reports Generated.....	134
4.4.2.3 JSON Reports converted into MIST format.....	134
4.4.2.4 MIST reports converted into N-Gram data.....	136
4.4.2.5 Sparse matrix Generated.....	137
4.4.2.6 KNN Classification Algorithm applied.....	138
4.4.3 Building Network Part of the Framework.....	139
4.4.3.1 Cuckoo generates dump.pcap file.....	140
4.4.3.2 Snort generates text file out of pcap file.....	140
4.4.3.3 Text file converted into MIST format.....	141
4.4.3.4 MIST file converted into sparse matrix.....	141
4.4.3.5 KNN applied to sparse matrix.....	141
4.4.4 Building Sandbox Evasion Detection part of the Framework.....	142
4.5 Distributive Execution of the Framework.....	143
4.5.1 Introduction.....	143
4.5.2 Proposed Distributive Framework.....	145
4.5.3 Centralized versus Distributed Computing.....	147

4.5.4	Role of Apache Spark in the Framework.....	148
4.5.4.1	AWS M3.....	149
4.5.4.2	M3 medium.....	149
4.5.4.3	Amazon EC2.....	149
4.5.4.4	SSD.....	149
4.5.4.5	HDFS.....	150
4.5.4.6	HDFS Architecture.....	150
4.5.4.7	Non-Computational Data Locality.....	150
4.5.4.8	Data and Processing on each machine.....	150
4.5.5	Working of the Hadoop Cluster.....	152
CHAPTER 5 FINDINGS AND CONCLUSIONS		155
5.1	Execution for a clean sample on the Framework.....	155
5.1.1	Static Analysis.....	155
5.1.2	Dynamic Analysis.....	155
5.1.3	Snort Analysis.....	161
5.2	Execution for a malicious sample on the Framework.....	162
5.2.1	Static Analysis.....	162
5.2.2	Dynamic Analysis.....	163
5.2.3	Snort Analysis.....	167
5.2.4	Sandbox Evasion Detection.....	170
5.3	Final Results.....	170
5.3.1	Results obtained from the Framework after all four steps.....	171
5.3.2	Comparison with the existing Centralized systems.....	175
5.3.2.1	Working of the Distributed Framework.....	177
5.3.2.2	Blockchain for Records and Rewards.....	177
5.3.2.3	Using Ethereum for maintaining Blockchains.....	180
5.4	Conclusions.....	182
5.5	Limitations of the Study.....	182
5.6	Further Research Potential.....	183
REFERENCES.....		184
APPENDIX 1 : PUBLICATION DETAILS		191
APPENDIX 2 : Company Letters and Mail Conversations.....		193
APPENDIX 3 : Screenshots of NoDistribute malware detection of self created malwares....		203
APPENDIX 4 : Programming Codes Used for Complete Execution.....		215

LIST OF FIGURES

- Figure 1.1 Estimate of average economic loss to countries from cyber-attack.
- Figure 1.2 Classifications of Malware Families According to their Detection Techniques.
- Figure 1.3 Malware using dynamic programming with a pool of IP addresses and ports.
- Figure 1.4 Difference between Polymorphic, Metamorphic and Advance Malware.
- Figure 1.5 Structure of the Thesis.
- Figure 3.1 Research Methodology.
- Figure 3.2: Survey Questionnaire
- Figure.3.3: Companies Covered, location and role of respondents during Data Collection and Analysis
- Figure.3.4 Location of organizations
- Figure.3.5: Title level held in the organization
- Figure.3.6: Degree of security responsibility
- Figure.3.7: Type of malicious activities
- Figure.3.8: Origin of malicious activity
- Figure.3.9: Percentage of budget allocated for cyber security
- Figure.3.10: Level till which devices get affected
- Figure.3.11: Level of concern for cyber security
- Figure.3.12: Technologies used for protection
- Figure.3.13: Approach followed for malware detection
- Figure.3.14: Resource allocation to cyber security
- Figure 3.15: Response time after malicious attack
- Figure 3.16 Factors hindering cyber security efforts in the organization
- Figure 3.17 Framework for Detection and Mitigation of Polymorphic Shellcode.
- Figure 3.18 Representation of a Stack Frame.
- Figure 3.19 Creating a vulnerable program.
- Figure 3.20 Output after crashing the program.
- Figure 3.21 Output of GDB.
- Figure 3.22 Information of stack pointer.
- Figure 3.23 Finding beginning of the stack.
- Figure 3.24 Output from NoDistribute with our created shellcode.
- Figure 3.25 Copying 'raadmin.exe' file on desktop.
- Figure 3.26 Getting system information using infection through shellcode.
-

Figure 3.27 Threat Reports of two advance malware created during testing.

Figure 3.28 Output of Nodistribute while detecting 'raadmin.exe' shellcode.

Figure 3.29 Flowchart to demonstrate proposed working of PosDeF.

Figure 3.30 Machine Learning Stub.

Figure 3.31 Balancing the Training-Testing Dataset.

Figure 3.32 The underlying steps in all four analysis parts.

Figure 4.1 Creation and injection of malicious payload for attacking the target system.

Figure 4.2 Process of Attacking the Target System with the created payload.

Figure 4.3 Flowchart showing working of WannaCry.

Figure 4.4 Snapshot of a part of the 'data.csv' file created for training and testing purpose.

Figure 4.5 Features selected on the basis of their variance.

Figure 4.6 Screenshot showing performance of various classification algorithms for testing of static analysis.

Figure 4.7 Depiction of a MIST Instruction.

Figure 4.8 Understanding a MIST Instruction with the help of an example.

Figure 4.9 MIST Levels.

Figure 4.10 A Sample Confusion Matrix.

Figure 4.11 Proposed model for Decentralized Malware Detection.

Figure 4.12 Comparison of performance between a centralized and a distributive system.

Figure 4.13 Traditional Database Processing Structure.

Figure 4.14 Hadoop Cluster.

Figure 4.15 Hadoop Master-Slave Architecture.

Figure 5.1 Python Script for submitting files to Cuckoo.

Figure 5.2 Analysis of files by Cuckoo.

Figure 5.3 Cuckoo processing the file in a virtual sandbox.

Figure 5.5 Conversion of JSON report to MIST format.

Figure 5.6 N Gram Sparse Matrix.

Figure 5.7 A 16 X 7255 Dimension Sparse Matrix.

Figure 5.8 Explanation of a sample Sparse Matrix.

Figure 5.9 Commands to show the formation of Sparse Matrix for our data.

Figure 5.10 KNN algorithm used to predict probability on the transformed matrix.

Figure 5.11 Snapshot of dump.pcap file.

Figure 5.12 Predicting probability of maliciousness using Random Forest Algorithm.

Figure 5.13 Comparison of various machine learning algorithms to show that Random Forest is the best.

Figure 5.14 Result of a file showing malicious behavior.

Figure 5.15 Same file submitted to Cuckoo for dynamic analysis.

Figure 5.16 Sample running in a VM virtual box.

Figure 5.17 A JSON report generated for the sample.

Figure 5.18 JSON report converted to MIST format using cuckoo2mist converter.

Figure 5.19 MIST report for the sample.

Figure 5.20 Graph comparing all algorithms and finding that KNN is best suited for analysis.

Figure 5.21 KNN Algorithm predicting maliciousness of the sample from transformed matrix.

Figure 5.22 A dump.pcap file generated by Cuckoo.

Figure 5.23 Graph comparing all algorithms and showing Random Forest to be the best for analysis.

Figure 5.24 Random Forest algorithm predicts maliciousness of the sample to be 74%.

Figure 5.25 Evasion of sandbox by the sample.

Figure 5.26 Graph showing training on Apache Spark.

Figure 5.27 Distribution of load on 4 nodes aggregated for 1 hour.

Figure 5.28 A part Blockchain of five blocks generated by Implemented model.

Figure 5.29 Comparison of Ethereum Blockchain database architecture without Indexing and with Indexing on top of Hadoop.

LIST OF TABLES

Table 1.1 Comparative analysis of various Malware Detection Techniques.

Table 1.2 Differences between Polymorphic and Metamorphic malwares.

Table 3.1 Qualitative versus quantitative research.

Table 3.2 Companies Covered, location and role of respondents during Data_Collection and Analysis.

Table 3.3: Type of organisations of respondents

Table 3.4: Count of locations of respondents

Table 3.5: Count of title level of respondents

Table 3.6: Count of degree of responsibilities of respondents

Table 3.7: Count of type of malicious activities in respondents organisations

Table 3.8: Count of origin of malicious activities in organisations

Table 3.9: Percentage of Overall budget allocated to cyber security

Table 3.10: Count of level till which devices get affected

Table 3.11: Level of concern for cyber security

Table 3.12: Technologies used for protection

Table 3.13: Approach followed for malware detection

Table 3.14: Resource allocation to cyber security

Table 3.15: Response time after malicious attack

Table 3.16: Factors hindering cyber security efforts

Table 4.1 Single Byte NOP Equivalent instructions.

Table 5.1 Final results obtained by testing sample files.

LIST OF ABBREVIATIONS

IM	: Instant Messaging
ML	: Machine Learning
GPU	: Graphical Processing Unit
TOR	: The Onion Router
DLL	: Dynamic Link Library
JSON	: JavaScript Object Notation
SMB	: Service Message Block
JVM	: Java Virtual Machine
CSV	: Comma Separated Values
ARP	: Address Resolution Protocol
SVM	: Support Vector Machine
IDPS	: Intrusion Detection Prevention System
RAT	: Remote Access Trojan
DDOS	: Distributed Denial of Service
IOT	: Internet of Things
IRC	: Internet Relay Chat
XOR	: Exclusive OR
MAC	: Medium Access Control
PCAP	: Packet Capture
KNN	: K Nearest Neighbor
DNS	: Domain Name Service
IP	: Internetworking Protocol
SSD	: Solid State Device
EC2	: Elastic Compute Cloud
HDFS	: Hadoop Distributed File System
AWS	: Amazon Web Services

CHAPTER 1: INTRODUCTION

CHAPTER 1: INTRODUCTION

1.1 Introduction to the Study.....	1
1.2 Malware Detection Mechanism.....	2
1.2.1 Signature Based Systems.....	3
1.2.2 Anomaly Based Systems.....	3
1.2.3 Emulation/Behavior Based Systems.....	3
1.3 Types of Malicious Attacks	6
1.3.1 Viruses.....	6
1.3.2 Worms.....	6
1.3.3 Trojans.....	7
1.3.4 Rootkits.....	7
1.3.5 Botnets.....	7
1.3.6 Ransomware.....	7
1.3.7 Zombies.....	8
1.3.8 Rats.....	8
1.3.9 Spyware.....	8
1.3.10 Shellcodes.....	8
1.3.11 Polymorphic Shellcodes.....	9
1.3.12 Metamorphic Shellcodes.....	10
1.3.13 Advance Malware-Beyond Polymorphic and Metamorphic Shellcodes.....	11
1.4 Shellcode Infection Mechanisms.....	12
1.4.1 Buffer Overflow.....	13
1.4.2 Code Injection	13
1.5 Machine Learning used for Malware Detection.....	13
1.5.1 Introduction to Machine Learning.....	13
1.5.2 Reasons for Using Machine Learning for Malware Detection.....	14
1.5.3 Supervised v/s Unsupervised Machine Learning.....	14
1.6 Motivation for the Study.....	14
1.7 Scope and Contribution of the study.....	15
1.8 Research Methodology used in the Study.....	16
1.9 Structure of the Thesis.....	17
1.10 Keywords used.....	19

CHAPTER 1

INTRODUCTION

1.1 Introduction To the Study

The Internet world is ridden with all kinds of malware and hackers are constantly trying out new innovative techniques to evade detection mechanisms. When shellcodes became the choice of attack, antiviruses were quick to come up with signature detection mechanisms to catch them. Hackers went one step ahead and created the so called 'Polymorphic shellcodes'. These have different signatures for the same shellcode. Thus it becomes impossible for a signature detection system to detect them. In our study, we are going to create a framework for the detection and mitigation of untraced polymorphic shellcode.

In May 2017, when WannaCry ransomware attacked, 150 countries with 2,00,000 computers all over the world were affected and some \$50,000 was already paid to the attackers by various companies and individuals. The base of this ransomware was shellcode. In early 2017 when Donald Trump won the US elections, powershells were injected in mails which were sent to Hillary Clinton's aide John Podesta. Once the shellcode entered the machine, it infected all the systems in the network and exposed all emails of Hillary, which caused a great reputational loss to her campaign and she paid the price by losing the elections.

In 2016, the game PokemonGo was found to be carrying shellcodes which caused clicking frauds and millions of dollars were earned by the attackers by gaining root access to the phones and clicking all kinds of ads and sometimes stealing network bandwidth and sensitive data from the compromised machine. HummingBad, another kind of shellcode malware, affected ten million phones worldwide and got admin

access to the phone and used that for generating fraudulent advertising revenue up to \$300,000 per month through the forced downloading of apps and clicking of ads.

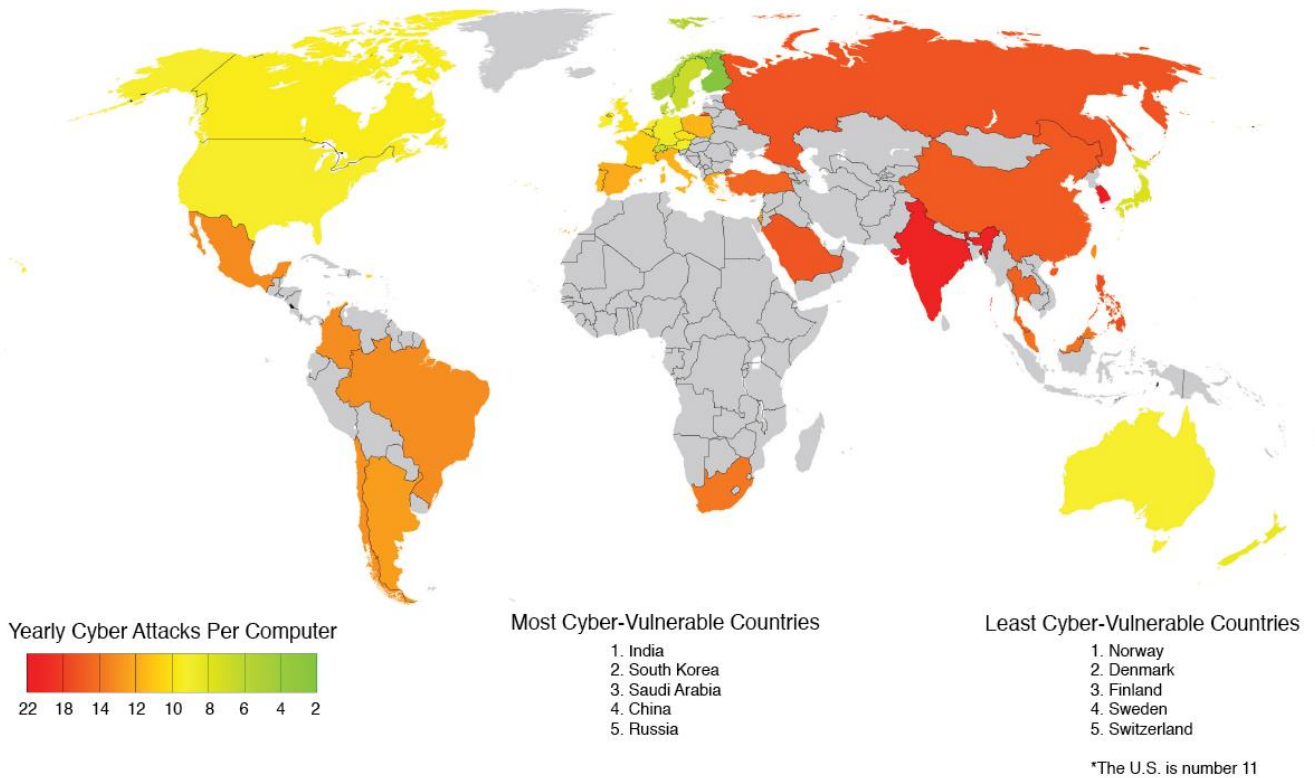


Figure 1.1 Estimate of most and least cyber-vulnerable countries.

According to the ‘Global Cybervulnerability Report’ of 2017 by UMIACS, Figure 1.1 (<http://terp.umd.edu/what-nations-are-most-vulnerable-to-cyberattacks/#.XVFrpvIzapo>, March 29, 2016) shows a rough estimate of which countries are most and least cyber-vulnerable. The global average loss per cyber-attack is \$32,20,000. Out of this a major contribution is that of shellcodes. Thus it becomes imperative that research may be done on the working and successful detection of shellcodes and polymorphic shellcodes.

1.2 Malware Detection Mechanism

Computer security, also known as cyber security or IT security means the protection of information systems from theft or damage to the hardware, the software, or to

the information on them. It also means protecting the systems against denial of services or misdirection of service of any kind.

The computer security field is growing in importance because of our increasing reliance on computer systems as well as the exponential growth of "smart" devices, including smartphones, televisions and

tiny devices as part of the Internet of Things (IOT) and of wireless networks like Bluetooth and Wi-Fi.

There are various ways in which malwares can be detected, chiefly: Signature based detection, Anomaly based detection and Behavior Based detection mechanisms.

1.2.1 Signature based systems

In this type of detection mechanism, the opcode pattern is extracted from the executable file, called the signature of the file. This signature is then matched with a database of already stored signatures. If it matches a malware signature, it is flagged as malicious otherwise it is declared as clean. This technique is fast and accurate and raises very low number of false alarms. However, a new virus cannot be detected. The signature has to be in the database for it to be detected. The database is huge and needs constant updating. Therefore the detection is slow. Also a new virus signature needs 7 hours to get updated in the database.

1.2.2 Anomaly based systems

This type of detection mechanism monitors various processes on systems for any abnormal activity. It detects a malware based upon any anomalous activity it performs. It is quite reliable and can detect even new, previously unknown malware. However it raises large number of false alarms which can be quite cumbersome to deal with.

1.2.3 Emulation/Behavior based systems

These systems make the malware execute in a virtual environment and observe all its behavior-including abnormal activity, connection failures, network telescopes, pattern of destination addresses, and causation. They are able to identify encrypted malware like those of polymorphic and metamorphic nature but are very expensive to implement.

Detection Technique	Strengths	Weaknesses
Signature Based	<ul style="list-style-type: none"> • Fast and accurate. • Low chance of false positives. 	<ul style="list-style-type: none"> • Cannot detect mutated virus like polymorphic or metamorphic. • Cannot detect new and previously unknown malware. • Slow because of huge database.
Anomaly Based	<ul style="list-style-type: none"> • Can detect new malware whose signatures are missing in the database. • Can detect abnormal activity quickly. 	<ul style="list-style-type: none"> • High chance of false positives. • Expensive to implement.
Emulation/Behavior Based	<ul style="list-style-type: none"> • Can detect polymorphic and metamorphic (encrypted) malware 	<ul style="list-style-type: none"> • Expensive to implement. • The malware may detect the virtual environment and stay

	<ul style="list-style-type: none"> • Can show the severity of malware based on behavior 	dormant till it is in the sandbox, not showing its actual behavior.
--	--	---

Table 1.1 Comparative analysis of various Malware Detection Techniques.

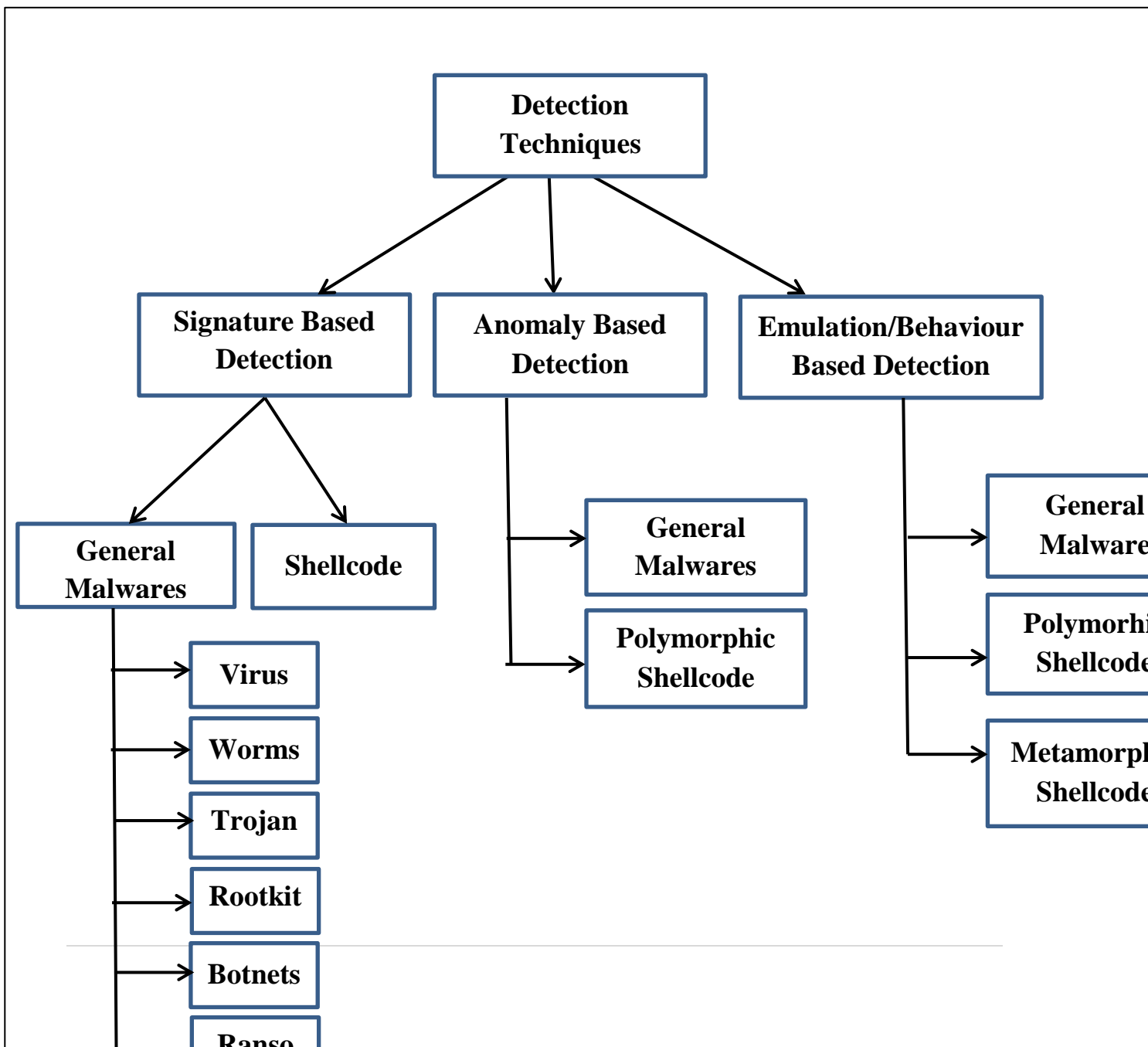


Figure 1.2 : Classifications of Malware Families According to their Detection Techniques.

1.3 Types of Malicious Attacks

Our computers and systems are always under attack from malicious components. Attackers continue planning different ways to enter into systems and do their nefarious activities. They keep developing different kinds of malware which choose different routes, different attack mechanisms and different ways to remain hidden from prying eyes of antimalware software.

There are various types of malicious attacks which are jeopardizing our computer systems everyday like:

- Viruses
- Worms
- Trojans
- Rootkits
- Botnets
- Ransomware
- Zombies
- Rats
- Shellcodes

1.3.1 Viruses

A computer virus is a type of malicious piece of code which propagates by inserting a copy of itself into and becoming part of another program. It spreads from one system to the next, leaving infections as it travels. A virus can be an Overwriting virus, Appending virus, Prepending virus or a Cavity virus depending upon where it attaches its code in the normal program. A virus will normally attach itself to an executable file. Thus, it will be in a system but will not be active or will not spread until a user runs or opens the executable to which it is attached. When that file executes, the virus executes as well.

1.3.2 Worms

Worms are just like viruses because they can replicate functional copies of themselves and damage the systems just like viruses do but they do not require a host. Worms are standalone pieces of malicious code which do not require a host program to propagate.

1.3.3 Trojans

A Trojan is another type of malware named after the wooden horse used by the Greeks used to infiltrate the city of Troy. It is a malicious piece of software which looks perfectly legitimate. It uses social engineering to trick users to get downloaded into their systems. The various functions Trojans can perform are- deleting files, changing desktops, pop-up windows, activating or spreading other malware, stealing data etc. They more often create backdoors but do not self-replicate. They spread by user interaction like opening e-mail attachments or running a file from the Internet.

1.3.4 Rootkits

Rootkits are a collection of tools and programs designed to give administrator-level access of computer or network to the attacker. Root means administrator in UNIX and Linux systems. The software components of rootkits are associated with different malwares like Worms, Trojans, Viruses etc. These malware hide their actions and existence from users.

1.3.5 Bots

Bot is an automated process normally used for gathering information. Bots are actually Internet robots which automate repetitive tasks which may be malicious like launching Dos attack, relaying spam, logging keystrokes or providing information to a c&c(command and control) center. They can form a botnet which is a network of compromised computers to perform these malicious tasks in a very effective and fast manner. They can be self-propagating. They can also work as spiders or crawlers.

1.3.6 Ransomware

Is a type of malware which infects a system and encrypts its data till the required amount of ransom is paid. Thus a system is locked and is only unlocked and the data only decrypted when the ransom is paid. The ransomware is normally loaded into the system using a trojan typically using social networking. Different ransomwares work in different fashion but all of them would work by first getting the admin rights of the shell. The payment is mostly made using bitcoins. Some ransomwares may work as wipers and may not decrypt the system even after the ransom has been paid.

1.3.7 Zombies

Zombie is a computer system which is compromised by an attacker and works for him, unaware that it is working for the attacker. Normally, a system which is used as a zombie will show decreased performance for the unaware users. A network of zombies can be used to perform mass illegal activities. They can be used to send massive amount of spam , Dos attacks or attacking other computers or websites.

1.3.8 Rats

RAT stands for Remote Access Trojan. RATs are malwares that provide a backdoor to the attacker for his administrative control. Thus they are also called 'Creepware'. These are installed invisibly without the user's knowledge and can enable the attacker to monitor behavior, modify system configuration, use Internet connection or perform some criminal activity.

1.3.9 Spyware

Spyware is type of malware which does espionage. It takes search history and sends personalized advertisements and tracks user activities to send them to third parties.

1.3.10 Shellcode

Shellcodes are tiny programs which look like legitimate code and are inserted in code-caves in a program code. Once executed, they provide a shell to the hacker with root privileges. Eg. 'Win32/ShellCode.gen!V' is a shellcode exploit for the Windows system. The most common shellcode instruction is to execute a shell such as /bin/sh, or cmd.exe. The only possible reason for launching such commands is to take control or exploit a compromised machine. When the exploit code causes what would normally be a critical error in the targeted program, the program jumps to the shellcode and is tricked into executing the attacker's commands. There are many harmful effects of shellcode. A shellcode can connect itself with internet, can display promotional adverts and fake messages, can corrupt secured system programs and files and can affect internet speed and system performance. It can install more malwares and fake programs, can show unwanted pop-ups, can crash the system, can help view others desktop, can sniff data from the network, can dump password hashes or use the owned device to attack hosts deeper into the network.

1.3.11 Polymorphic Shellcodes

Shellcodes have typical structures which can be detected through pattern - matching by the IDPS software. Thus hackers have come up with a new type of shellcodes called 'Polymorphic Shellcodes'. Some algorithms are executed on the shellcodes which make them 'look' very different, say by changing their commands, encryption, code transposition, dead code insertion, register reassignment etc. However, the polymorphic shellcode still performs the same functionality that the original shellcode did. Pattern matching fails here and increases worries of IDPS and Antivirus developers.

Suppose we have a malware say:

[NNNN]

If we use it in its original form, its signature may get detected. Therefore, we encrypt it and it becomes:

[LLLL]

But to use it, we have to put the decryption routine, D with it because it has to be deciphered before it can be executed. So now the signature looks like:

[DLLLL]

Now we can change the key and we get a different signature each time

[DLLLL] with key K1

[DFFFF] with key K2

[DAAAA] with key K3

[DBBBB] with key K4

Now the key is different each time but there is a static part 'D' which is constant in each signature. Over a period of time, because of this weakness, a malware may be detected.

Therefore, advanced polymorphic malwares change this decryption routine too, each time a signature has to change:

[D1LLLL] with K1

[D2FFFF] with K2

[D3AAAA] with K3

[D4BBBB] with K4

The polymorphism may be caused by:

- Register renaming
- Code permutation
- Code expansion
- Code shrinking
- Garbage code instruction etc.

1.3.12 Metamorphic Shellcodes

These shellcodes are BODY POLYMORPHIC. They do not have a decryptor. They create a new malware which looks completely different from the original signature. There is no decryption algorithm or a key involved. Following are the differences between Polymorphic and Metamorphic malware:

Polymorphic Malware	Metamorphic Malware
They need to return back to their original form for execution.	They do not need to return to their original form for execution.
There is a decryption routine which uses a key, embedded in the mutated code.	The mutated code has no decryption routine or any key.
It is less difficult to write as compared to metamorphic shellcode	It is more difficult to write.
Their body is constant.	Their body is polymorphic, though functionally same.
Eg. W32/Coke, W95/HPS and W95/Marburg	Eg. Simile, ZMist

Table 1.2 Differences between Polymorphic and Metamorphic malwares

1.3.13 Advance Malware- Beyond Polymorphic and Metamorphic Shellcodes

Polymorphic shellcodes change their signatures with same or different decryptor routines. Metamorphic malware change their body even without decryptor

routines or keys. The changed signatures are functionally equivalent to the original ones. However, a new family of advance malwares is coming up in the wild. This type of malware is not only polymorphic or metamorphic but can change their behaviors also in every iteration. Say in one instance, the malware may delete files, in other, it may open network ports, in still other, it may try to get remote access.

There already are some malwares which use ‘dynamic programming’. Say we have a malware which has infected a system. Now it tries to use an IP address and a specific port, say IP1, P1. If these are blocked by a firewall, the malware has a pool of IP addresses and ports to choose from. It keeps trying other IP addresses and ports till it can get past the firewall. In this manner a different behavior is shown by the malware.

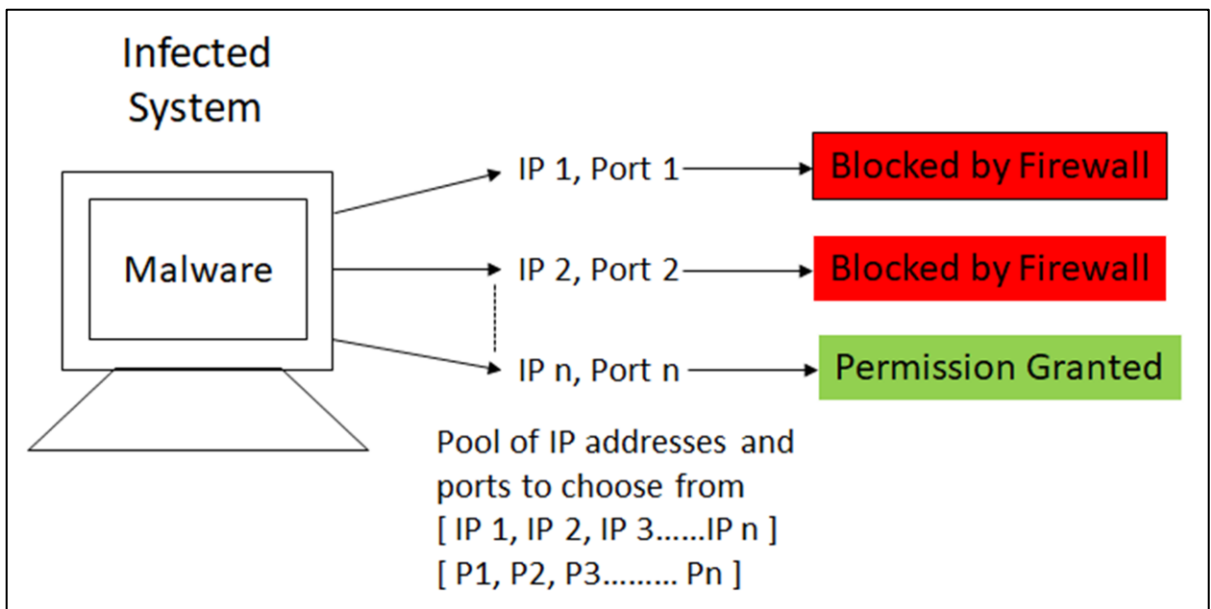
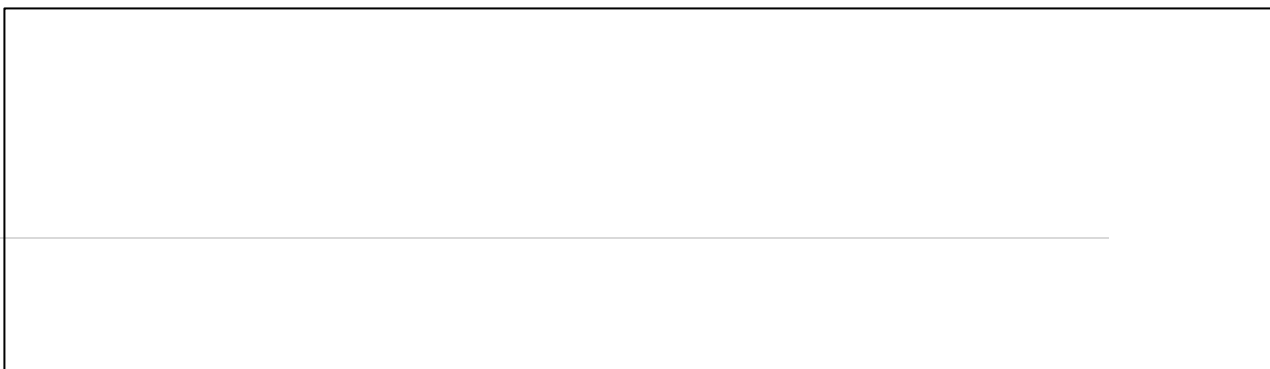


Figure 1.3 : Malware using dynamic programming with a pool of IP addresses and ports.

However more advance behavior mutations are possible in malware in addition to them employing polymorphic and metamorphic techniques.



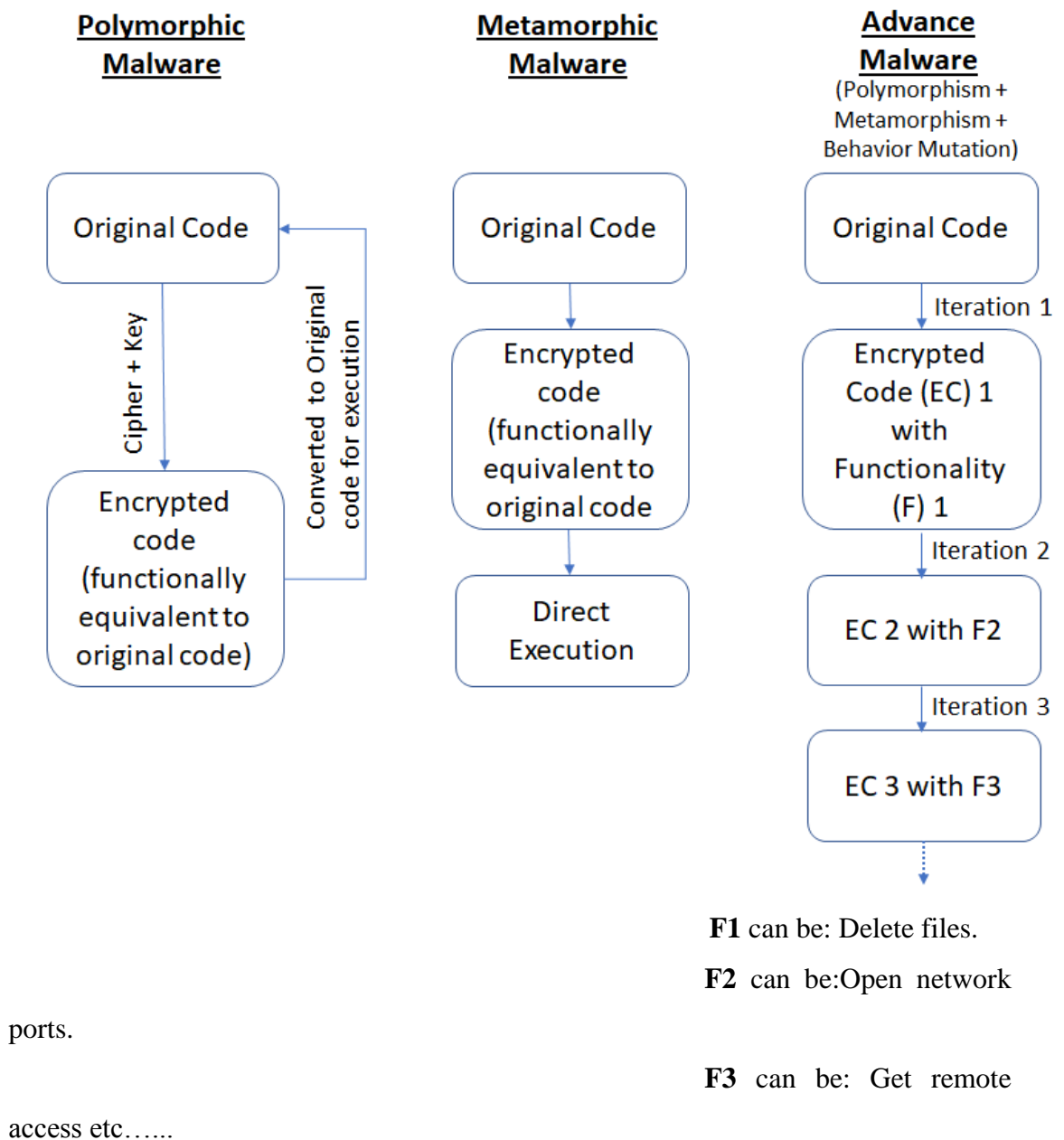


Figure 1.4 Difference between Polymorphic, Metamorphic and Advance Malware.

1.4 Shellcode Infection Mechanisms

There are several ways in which shellcodes can attack a host and take control of it. The two most common ways are Buffer overflow and Code Injection.

1.4.1 Buffer Overflow

There are many possible avenues that would allow a shellcode to take over a remote host, but by far the most common is exploiting a buffer overflow. Despite this style of attack having been a known vector for worm propagation for many years, buffer overflow vulnerabilities continue to show up frequently in software. The essence of a buffer overflow attack is to write more data to a buffer than it has allocated space for. The excess data will then overwrite adjacent memory addresses, and when this is done properly, the overwritten memory areas can be used to execute arbitrary code. Buffer overflow attacks must be targeted specifically at an architecture and operating system. The buffers that are overwritten can be either on the heap or the stack, with different exploitation requirements for the two options. The heap is the pool of free memory that is allocated dynamically to the running program. It is typically referenced indirectly. The call stack stores the information about the execution of the program, but varies greatly with operating system and machine environment.

1.4.2 Code Injection

A second form of attack is known as code injection. It is typically found in web applications. In this attack, a server accepts posted data from a client, and if it doesn't properly sanitize the data for code markers, it can end up executing the posted data as code. This allows the client the opportunity to execute arbitrary code on the server, allowing the client to compromise it and infect it with a worm.

1.5 Machine Learning used for Malware Detection

There are various ways in which Shellcodes can be detected, chiefly: Content Based and Behavior Based detection mechanisms. However, to detect the advance malware that we are predicting, which is polymorphic, metamorphic and functionally different, machine learning algorithms are the only viable choice.

1.5.1 Introduction to Machine Learning

Machine learning is an upcoming field in computer science. It is an application of artificial intelligence that gives the system, a capability to automatically learn and

improve from experience without being explicitly programmed. It is basically a collection of programs meant to learn from examples, also called as 'data set'. Say we want to teach an application to recognize handwritten characters. Now there are two ways of doing it. First is to write down a set of rules for each character specifying all different shapes and styles a character can have. Add to it huge variations in human handwriting. Also there would be separate rules for printed and cursive characters. Writing of such programs would be a humongous task for a programmer and it would also require tremendous computing power.

The second way is to prepare a data set having many different examples of handwritten characters and the computer, by itself learns the rules that best identify a character. This way of learning is called supervised learning. Machine learning is of two types: supervised Learning and Unsupervised Learning.

1.5.2 Reasons for Using Machine Learning for Malware Detection

Computer science is changing very fast and according to some statistics, Artificial intelligence has arrived 10 years prior to the expected time. Researchers have shown an incredible success rate of detecting malwares using machine learning although the problem with machine learning is that no matter how much data you train the machine learning algorithm with, it's not sufficient and also it takes huge amount of time to train a large dataset using machine learning. However recent innovations like deep learning and cheaply available GPUs have made machine learning really fast but still the time to train the model increases linearly as the size of dataset increases.

1.5.3 Supervised v/s Unsupervised Machine Learning

Supervised Learning: Here labeled training data is provided to the system and then it tries to classify the unknown sample.

Unsupervised Learning: Here we do not provide labeled training data and ask the system to cluster the sample in n number of clusters. Unsupervised learning is quite

difficult as it is very hard to analyze how system classifies data as it does not output the parameters on which it clusters the data.

1.6 Motivation for the Study

Among the various malware families, the general malwares and shellcodes are detected using signature detection mechanisms. Polymorphicmalwares can be detected using Anomaly based detection mechanisms. Metamorphic malwares can be detected using Behavior/Emulation based detection mechanisms. However, advance malwares which not only change their signatures, but also their behavior are the most difficult to detect. Thus one defense mechanism is not enough for these malwares. They have to be detected using a multidimensional approach. No current mechanism attacks a malware from all sides so as to perfectly detect it. A framework is required for IPS and IDPS systems which takes care of static analysis, dynamic analysis, network analysis and sandbox evasion analysis.

Many current solutions fail to detect a new variant of malware for which they have no prior information. WannaCry and Petya are the most recent examples. The advance malwares can show a new signature and behavior every time which makes logging impossible.

Malware detection is an ever growing industry where, it is said that antivirus companies makes malwares first and then give patches later. The rich keep getting richer and safety and security seems to be slipping out of the hands of ordinary citizens. Our framework is designed to give safety and security features to every common man in the cheapest possible manner.

The solutions which claim to be able to detect even advance malware require large CPU capacities and very high speeds. Our framework will achieve the same performance using distributive architecture so that the load is shared by all the nodes participating in the detection process.

The database of the malware information will be updated in an instant and the current copy of the database would be available to all the nodes at all times. It is seen that at

times, it takes about seven days for an update to reach all systems across the world. Till that time, the malware is easily able to perform its dirty work.

1.7 Scope and Contribution of the study

The framework covers the security domain and will detect malwares of all types in an efficient, timely and cost effective manner. The range of people benefitted out of this system is unlimited. Big organizations need not spent millions of rupees on expensive detection systems. This framework does the same job in the least expensive manner using distributive framework. The latest malware can be detected as soon as possible and the information travels to all the nodes participating in the process in a very quick manner. The common man on the Internet who is sometimes bluffed by the security agencies will be saved. The latest and the most current detection mechanism will be available to him at the lowest cost possible. Governments, schools, colleges, hospitals, all small, medium or big organizations stand to benefit from this study. The study will not only aim at detecting general malwares like virus, Trojans, rats, worms etc., but will also aim at ransomware, polymorphic, metamorphic and advance malwares.

1.8 Research Methodology used in the Study

A series of interviews were conducted with industry professionals, security auditors, ethical hackers, bankers, developers, database administrators, senior managers, HR professionals, university lecturers, students, and many other people in various organizations, serving at different positions. Their security problems were analyzed. There were a few important findings which are stated below:

1. Many of them have quite often suffered from a ransomware threat. If the organization is big, these threats are not brought to light because of prestige issues.
 2. These people are often under risk from a new, previously unknown malware.
 3. They are spending huge amount of resources on maintaining security within the company and still do not feel completely secure.
 4. They are often worried about losing important organization data and resources to puny hackers.
-

5. The entire security system is centralized thus is expensive to maintain and sometimes slow to counter fat spreading attacks.

The hypothesis was developed by taking a number of malwares from the wild and testing them against popular anti malware solutions. Some were able to detect these malwares and some were not. All these malwares were of general families. Therefore, we developed some polymorphic malwares ourselves and then tested them against these solutions. To our surprise, many of these solutions were unable to detect them. All these malwares were able to do all kinds of malicious activities like installing keyloggers, capturing screens, deleting and sometimes encrypting files and data etc. Any such malware, if launched can be a big threat to the world. Popular ransomwares like WannaCry, Petya, Misha etc. were studied in detail to understand their structure, behavior and attack pattern.

Therefore it was concluded that the present solutions are no match for advance technologies that hackers have at their disposal. So a framework has to be developed which takes care of all these threats effectively. It has to be fast and affordable. It should be distributive and power should be in the hands of common man and not centralized in the hands of anti-malware companies.

The research methodology is experimental in nature and of quantitative type. We would create the framework and would try to verify its working by testing it against various malwares. The model will be trained first by giving it known samples so that it self learns the factors required for detection. Then it would be tested to check whether it is giving correct results or not. Finally, it would be tested by taking random samples from the wild to analyze its effectiveness and performance.

1.9 Structure of the Thesis

Chapter 1. Introduction

The first chapter gives introduction of the study. We classify the malware families based on their detection techniques. We find out about polymorphic and metamorphic

shellcodes which cannot be detected by signature detection alone. Advance malware which can change their behavior also cannot be detected by all present techniques. We understand the scope and contribution of the study and the research methodology used.

Chapter 2. Literature Review

In the second chapter, we do a literature review of the work done till now in various areas like malware detection techniques, working of common antimalware solutions and the concept of CVE's, polymorphic and metamorphic malware, machine learning for malware detection, working of emulators like Cuckoo, studying static and dynamic behavior of malware, studying various ransomware, studying distributive technologies like Hadoop, understanding cloud architecture and working, understanding Blockchains, studying digital currencies like Bitcoins and Ethereum.



Figure 1.5 : Structure of the Thesis

Chapter 3. Research Methodology

The third chapter describes the research approach undertaken for proposal and development of the model. The objective, scope and contribution of the study is also understood, gauged and analyzed. Hypothesis is built based upon interviews, testing with malware samples from the wild and self-created advance malwares. The framework is designed and all the steps of the proposed model are understood in detail. Algorithms for all the steps are built and formulae generated for calculations like threshold, maliciousness probability and severity.

Chapter 4. Analysis and Interpretation

The fourth chapter deals with analysis and interpretation. The process of creation and attack of advance malwares is studied in detail. We have worked with live ransomware samples like WannaCry and Petya and understood their impact, attack process and behavior. We have built every part of the framework starting from the static part to behavior, network and sandbox evasion part. We have shown how the entire model works in a distributive manner and how the blockchain concept can be used for logging the database and also as a means for rewards and recognition.

Chapter 5. Findings and Conclusions

The fifth chapter deals with findings and conclusions of the study. We have executed every part of the trained model with clear and malicious malware samples and found out whether the model is giving correct results or not. The severity is analyzed according to the calculated probability. The concept of alerts is built which gives message to the user about the threat. The study is concluded and limitations are enumerated. Further scope of the study is also analyzed and more research potential of the study is observed.

1.10 Keywords used

Shellcode, polymorphic shellcode, metamorphic shellcode, machine learning, malwares, antivirus, IDPS, IPS systems, framework, virus, Trojan, CVE, severity, alert, profiling, pretraining, static analysis, dynamic analysis, Snort, network analysis, sandbox evasion, VirusShare, NoDistribute, buffer overflow, Cuckoo, JSon, MIST, KNN classifier, Random Forest, Naïve Bayes, N Gram, Apache Spark, Amazon AWS, cloud storage,

Distributive architecture, machine learning, artificial intelligence, Hadoop, Map Reduce, master node, slave node, Ethereum, blockchain, WannaCry, Petya, TOR, DLL

CHAPTER 2: LITERATURE REVIEW

2.1 Analysis of Literature Survey.....	21
2.2 Malware detection techniques.....	22
2.3 The various threats and Working of common antimalware solutions.....	22
2.4 Concept of CVE's.....	23
2.5 Polymorphic and metamorphic malware.....	23
2.6 Machine learning for malware detection.....	24
2.7 Working of emulators like Cuckoo.....	24
2.8 Studying static and dynamic behavior of malware.....	25
2.9 Studying various ransomware.....	26
2.10 Studying distributive technologies like Hadoop.....	26
2.11 Understanding cloud architecture and working.....	26
2.12 Understanding Blockchains.....	27
2.13 Studying digital currencies like Bitcoins and Ethereum.....	27
2.14 Gap Analysis.....	28

CHAPTER 2

LITERATURE REVIEW

2.1 Analysis of Literature Survey:

The purpose of literature survey is to understand the previous work done in the chosen topic. Many papers were read and analyzed to understand in which directions other researchers have moved in this field. When we started to do research on this topic, there were many areas, of which we had little or no knowledge. The concept of polymorphic, metamorphic and advance malware had to be studied. We had to understand the detection techniques that researchers have been using till now. What are the benefits of these techniques and what are the gaps that these techniques are not able to fill. We had to understand the behavior of different kinds of malwares and know their attack methodologies to counter them better. We had to know about centralized and distributive detection methodologies to gauge what are the advantages and disadvantages of each. We divided the literature review in the following sections:

1. Malware detection techniques,
 2. The various threats and Working of common antimalware solutions
 3. Concept of CVE's,
 4. Polymorphic and metamorphic malware,
 5. Machine learning for malware detection,
 6. Working of emulators like Cuckoo,
 7. Studying static and dynamic behavior of malware,
 8. Studying various ransomware,
 9. Studying distributive technologies like Hadoop,
 10. Understanding cloud architecture and working,
 11. Understanding Blockchains,
-

12. Studying digital currencies like Bitcoins and Ethereum.

2.2 Malware detection techniques

2.2.1 In the paper, “**A Survey on Heuristic Malware Detection Techniques**”[1], Z. Bazrafshan explains the widely used methods of malware detection Signature based, Behavioral based and Heuristic based. The signature based and Behavior based methods are explained in detail with their advantages and shortcomings. Heuristic methods gain an advantage over the other two methods because of various disadvantages that they have. The various techniques used in Heuristic methods involve API Calls, OpCodes, N-Grams etc. The working, advantages and shortcomings of Heuristic methods is also explained in detail.

2.2.2 In the paper, “**A Survey on Techniques in Detection and Analyzing Malware Executables**”[2], K. Mathur explains Static, Dynamic and Hybrid analysis of Malware executables, comparing the techniques in terms of where they can be used and how effectively. The paper also discusses the ways of creation of obfuscated malwares. Their technique uses Bi-feature analysis rather than Mono-feature analysis thus reducing the number of false-positives.

2.3 The Various Threats and Working of Common Antimalware Solutions

2.3.1 In the paper, “**Evaluating the Ability of Anti-Malware to Overcome Code Obfuscation**”[3], M. Carson, discusses the various anti malware technologies like Binary file checking, classifying packed and Polymorphic malware, understanding static system calls etc. The paper understands and compares the various techniques in terms of their strong and weak points. Various other related works have also been studied to know the points they

did not cover. The concept of code obfuscation is studied well and the techniques which can handle it are analyzed.

2.3.2 In the paper, “**Internet Attack Methods and Internet Security Technology**”[4], O. Adeyinka, investigates the various Internet attack methods like viruses, system and boot record infectors, eavesdropping, hacking, worms, Trojans, IP Spoofing, DOS, spam, email bombing and phishing. Many security techniques to counter these threats like cryptographic systems, firewalls, IDS, IPSec, SSL etc. are discussed. Their merits and demerits are also analyzed.

2.4 Concept of CVE's

2.4.1 In the paper, “**Managing vulnerabilities in networked systems**”[5], R. A. Martin, explains what are the common vulnerabilities and exposures database and classifies the parts of the database as Vulnerability scanner database, Software vendor patches and updates, Intrusions system detection signature database, Software vendor alerts etc. The paper describes how CVEs work and how the CVE list is built and what are the benefits of CVE compatibility.

2.4.2 In the paper, “**Common vulnerability scoring system**”[6], P. Mell, explains the recommendations of the National Institute of Standards and Technology (NIST) regarding the CVE naming system. It provides guidelines for use and acquisition of CVE databases. It explains that CVEs contain a list of publicly known vulnerabilities, authenticates newly published vulnerabilities and gives a unique name to each vulnerability. The paper explains how CVEs can be used by federal organizations and how they can make their security systems CVE compatible.

2.5 Polymorphic and metamorphic malware

2.5.1 In the paper, “**Classification of polymorphic and metamorphic malware samples based on their behavior**”[7], K Tsyganok, classifies malware on the basis of their behavior characteristics like WinAPI calls, files which are handled by the program, arguments taken by them etc. Clustering technique is used for classification of the samples. The work is tested using actual malware files.

2.5.2 In the paper, “ **Structural entropy and metamorphic malware**”[8], D. Baysa, explains that metamorphic malware is able to change their internal structure without changing their functionality. Thus they cannot be detected using Signature Detection Methods. Thus they use Structural Entropy to find differences in complexity of data in a sample file. They first segment the file using entropy measurements and then find the similarity between samples by finding the edit distance between sequence segments.

2.6 Machine learning for malware detection

2.6.1 In the paper, “ **Automatic analysis of malware behavior using machine learning**”[9], K. Rieck, propose a framework for incremental detection of malware using automatic clustering and classification of novel malware samples based on dynamic behavior analysis. A malware sample is projected in a vector representation and machines learning techniques work in this vector space.

2.6.2 In the paper, “ **Opem: A static-dynamic approach for machine-learning-based malware detection** ”[10], I. Santos, proposes the first hybrid machine learning based malware detector that combines both static opcode frequencies with dynamic execution trace of the binaries to reach to a conclusion whether they are malicious or clean. The files are executed in a sandbox environment to know their behavior. Also a vector of frequency of opcode frequencies is generated to determine maliciousness.

2.7 Working of emulators like Cuckoo

2.7.1 In the book, "**Cuckoo malware analysis**", D. Oktavianto, explains working of the Cuckoo sandbox. He explains that a sandbox is used to analyze a malware without the user worrying about the changes which will happen in the system during the process. Sandboxes work using snapshot technique which saves the virtual state of the machine while it runs. One can revert to the original state after analysis. Cuckoo was started as Google Summer of code project and it is open software. It analyses files like generic windows executables, DLL files, PDF documents etc. As results it produces traces of win32 API calls, memory dumps, screenshots and network traffic trace in PCAP format.

2.7.2 In the paper, "**An android application sandbox system for suspicious software detection**" [12], T. Blasing, explains that a sandbox will work by monitoring system and library calls and logging them. It also generates pseudo-random streams of user events like clicks, touches, or gestures, and also some system-level events. The sandbox is placed in the kernel. The entire system state is recorded so that no malicious activity can be hidden. All the logs are kept in a separate file.

2.8 Studying static and dynamic behavior of malware

2.8.1 In the paper, "**Implementation of malware analysis using static and dynamic analysis method**" [13], Y. Prayudi, uses two methods of malware analysis: static and dynamic. Static analysis is done without running the malware and dynamic analysis is done while running the malware in a secure environment. Both basic and advance static and dynamic malware analysis is done and finally a malware analysis report is generated. Various analysis tools like Anubis, Wireshark, VirusTotal, BinText and OllyDbg are used.

2.8.2 In the paper, “**An approach for malware behavior identification and classification**”[14], M Zolkipli, solves various threats like polymorphic and metamorphic malware threat in the paper. He first does behavior analysis of malware and then classification into families is done. The resulting framework identifies and classifies malwares based on behavior analysis. The malware classification is also optimized using AI techniques.

2.9 Studying various ransomware

2.9.1 In the paper, “**Ransomware digital extortion: a rising new age threat**”[15], A. Bhardwaj, studies Crypto and Locker ransomware, their propagation, attack techniques and new emerging threat vectors like screen lock, Windows and browser lock, encryption ransomware, pop advertisements and URL redirections. The reports generated give ransomware behavior analysis, code analysis and classification. Anti ransomware elastic cloud based platforms were used.

2.9.2 In the paper, “**Ransomware: Studying transfer and mitigation**”[16], R. Shinde, conducts interviews and surveys with victims of ransomware and tries to find the relation between ransomware attacks and victims age, education, company etc. Various methods of transfer of different kinds of ransomware are studied. Many mitigation strategies are analyzed and ways to spread awareness to protect and mitigate from these ransomwares are also suggested.

2.10 Studying distributive technologies like Hadoop

2.10.1 In the paper, “**Analytical review on Hadoop Distributed file system**”[17],K. Dwivedi,explains a step by step process of handling large amount of unstructured data using Hadoop and its Map Reduce algorithm. The paper explains how Hadoop Distributed File System-HDFS, is rapidly

growing and is being used in a variety of different applications. The working of HDFS and Map Reduce is explained in detail along with its various applications.

2.10.2 In the paper, “**Teaching Distributed Systems Using Hadoop**”[18], R. Correia, develops a teaching method using benchmark tests for students using Hadoop framework. Students study complex databases, various network infrastructures and system architectures on cloud based Hadoop systems. The framework proves very effective in this teaching methodology.

2.11 Understanding cloud architecture and working

2.11.1 In the paper, “**Spectrum of cloud computing architecture: Adoption and avoidance issues**”[19], Jangra A., explains the set of technologies used in cloud computing. The architecture of cloud is explained in detail along with the working details and various deployment and service models. Advantages of cloud computing like elasticity, flexibility and scalability are explained in detail. The paper ends with handling cloud deployment and use issues like security, privacy, internet dependency and availability.

2.11.2 In the paper, “**Cloud computing: types, architecture, applications, concerns, virtualization and role of its governance in cloud**” [20], P. Sareen, defines Cloud architecture, various cloud providers, compares cloud computing with grid computing, explains cloud virtualization, applications and concerns regarding cloud computing and detailed working of a cloud. The role of I.T. governance in cloud computing is also discussed.

2.12 Understanding Blockchains

2.12.1 In the chapter, “**Blockchains and the boundaries of self-organized economies: Predictions for the future of banking**” [21], T.J. MacDonald, explains how distributed Blockchains give a self-organized, autonomous economy. The chapter explains that Blockchains are resilient, transparent and distributed public ledger. This decentralized solution is cost effective and more innovative as compared to centralized solutions. It is an open and a dynamic system. It is also robust, flexible and secure. The chapter explains that Blockchains are crypto economic mechanisms which overcome difficulties in the existing economic systems.

2.12.2 In the paper, “**Blockchains and the economic institutions of capitalism**” [22], S. Davidson, explains that Blockchains are a digital technology which uses peer-to-peer network computing and combines it with cryptography to create an immutable and decentralized public ledger. The entries in the ledger can record not only money but other data structures like contracts, certifications, identities, property titles etc. The paper explains the working of Blockchain in detail.

2.13 Studying digital currencies like Bitcoins and Ethereum

2.13.1 In the paper, “**Bitcoin and beyond: A technical survey on decentralized digital currencies**” [23], F. Tschorsch, explains the Bitcoin protocols and its building blocks. The paper also discusses the various applications and impacts of Bitcoins. It explains how the purpose of banks is taken over by these decentralized currencies. Mining of bitcoins is also explained in detail. The various types and chain of transactions is also talked about.

2.13.2 In the paper, “**Bitcoin: Economics, technology, and governance**” [24], R. Böhme, explains that Bitcoins are based on transactional logs in a distributive computing environment where

participating computers are rewarded according to the level of honest participation.

This mechanism is against concentration of power. It is designed using irreversible transactions and a public record ledger. It is free to use and very flexible. The paper explains the design principles of Bitcoins, its uses and risk and regulatory features.

2.14 Gap Analysis

After going through many research papers and books related to the subject, we have found gaps in the research done till now.

- Some papers talk about only signature detection of shellcodes, however, they fail when dealing with polymorphic shellcodes.
 - Many of the techniques like Network Emulation are a high resource-consuming technique. Many models have just been proposed in theory but have not come into practice because of their complexity.
 - Some techniques deal with static and some with dynamic analysis, however, an approach of hybrid analysis is required for the most effective solution.
 - The techniques which are dealing with polymorphic signatures are not dealing with polymorphic blending attacks. These attacks merge the malware traffic with normal traffic so that even anomaly based IDPS systems are not able to detect them.
 - A multi staged attack where a small malware enters first and shows not much action initially but later calls bigger malwares, is also not tackled by many papers.
 - Some papers deal with obfuscation of the NOP sled and some with obfuscation of the rest of the code. What is required, however is that both type of obfuscations must be dealt with for true detection.
 - Some papers fail to take into consideration that the malware will stop showing its true behavior once it detects a sandbox. Various sandbox evasion techniques are being followed by malwares to evade detection.
-

- Also, considering that only signature detection is not the solution, we need behavior analysis, network analysis and sandbox evasion analysis.
- We can consider using machine learning tools and decentralized computing for this framework.

Some papers have taken a machine learning approach to malware detection. However, some have taken only static analysis approach and some are only analyzing the API calls that the malware is making. No paper discusses a four pronged approach of attacking the malware from four sides, static, dynamic, network and sandbox evasion based.

- Also, many of them are training their models on one or the other specific machine learning algorithms. We propose to create a framework where the algorithm is not fixed. We will calculate the score of each machine learning algorithm based on the training and the test data that we have and based on the score we will decide which algorithm to finalize for the framework.
 - Also since the machine learning approach is resource-heavy, thus it is imperative that a distributive way is used for implementing the entire system. This also most of the papers have not taken into consideration. Therefore, without the distributive approach, many machine learning algorithms fail to work when dealing with huge amount of data.
 - Thus a comprehensive solution is required which takes into consideration, all the above mentioned points and thus become a strong detection mechanism. Therefore, we propose a framework for the detection and mitigation of untraced polymorphic shellcode which takes into consideration all these points and thus becomes a hard detection mechanism.
-

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction: Research Approach.....	32
3.2 Method of Data Collection.....	34
3.3 Research Methodology used.....	36
3.4 Analysis Report of data collected.....	42
3.4.1 Type of organization	43
3.4.2 Location of organization.....	43
3.4.3 Title level held in the organization	44
3.4.4 Degree of security responsibility	45
3.4.5 Types of malicious activities	46
3.4.6 Origin of malicious activity	47
3.4.7 Percentage of Overall budget allocated to cyber security	48
3.4.8 Level till which devices get affected	49
3.4.9 Level of concern for cyber security	50
3.4.10 Technologies used for protection	51
3.4.11 Approach followed for malware detection	52
3.4.12 Resource allocation to cyber security	53
3.4.13 Response time after malicious attack	54
3.4.14 Factors hindering cyber security efforts in the organization	55
3.5 Key Understandings.....	56
3.6 Objectives of the Study.....	57
3.6.1 The Theoretical Framework of the Present Study.....	57
3.6.2 Conceptual Model Framework.....	57
3.6.3 Objectives of the Framework.....	60
3.7 Experiments to assess Polymorphic shellcode threat.....	60
3.7.1 Creating a shellcode by Smashing the Stack.....	60
3.7.2 Injecting a polymorphic shellcode in PE file.....	68
3.7.3 Getting privileges after infecting a file with polymorphic shellcode.....	
72	
3.8 Significance of Research.....	76
3.8.1 Need of the Study.....	76
3.8.2 Benefit of the Study.....	77
3.9 Designing PosDeF.....	78
3.9.1 Design Methodology.....	78
3.9.2 Objective.....	78
3.9.3 PosDeF Design.....	79

3.9.4 Proposed Working of PosDeF.....	80
3.9.4.1 Collection of Data for the Formation of the Training Set.....	83
3.9.4.2 Training with our initial dataset.....	83
3.9.4.3 Profiling the files.....	84
3.9.4.4 Convert the dataset into machine learning compatible format	84
3.9.4.5 Pre train the model.....	84
3.9.4.6 Train the dataset using various machine Learning Algorithms...	85
3.9.4.7 Iteratively scan all available files.....	86
3.9.4.8 Reward the user.....	86
3.9.4.9 Balancing the dataset.....	87
3.9.4.10 Endless loop.....	87
3.10 Algorithms Used for Building PosDeF.....	87
3.10.1. The Final Algorithm.....	87
3.10.1.1 Algorithm for Training of PosDeF.....	88
3.10.1.2 Algorithm for Threshold Calculation for the Final Framework during the training phase.....	89
3.10.1.3 Algorithm for Testing of PosDeF.....	91
3.10.2 Static Analysis.....	92
3.10.2.1 Algorithm for Static Training.....	92
3.10.2.2 Algorithm for finding out the Best Classification Algorithm for Static Testing.....	93
3.10.2.3 Algorithm for Static Testing.....	94
3.10.3 Behavior Analysis.....	94
3.10.3.1 Algorithm for Behavior Training.....	95
3.10.3.2 Algorithm for Finding Best Classification Algorithm for Behavior Testing.....	96
3.10.3.3 Algorithm for Behavior Testing.....	96
3.10.4 Snort Analysis.....	97
3.10.4.1 Algorithm for Snort Training.....	97
3.10.4.2 Algorithm for Finding out the Best Classification Algorithm for SnortTesting.....	98
3.10.4.3 Algorithm for Snort Testing.....	98
3.10.5 Algorithm for Sandbox Evasion.....	99

CHAPTER 3

RESEARCH METHODOLOGY

In this section, we will give an overview on the different types of research methodologies. This will be followed by the rationale behind selecting a particular method for this thesis.

3.1 Introduction: Research Approach

Research approaches are plans and the procedures for research that spans from broad assumptions to detailed methods of data collection, analysis, and interpretation. There are three approaches to research which are as follows:

1. Qualitative
2. Quantitative, and
3. Mixed methods

Qualitative research is an approach for exploring and understanding a social or human problem as described by individuals. It is aimed at gaining in-depth understanding of a specific organization or event, rather than surface description of a large sample of a population. This type of research puts more focus on how people feel, think and make their choices.

This research is largely managed with discussion around the concepts with some open questions. Respondents are asked to explain the reasons for their responses. This can reveal underlying motivations, associations and behavioural triggers [25].

Common data collection methods that are used in this research are focus groups, in-depth interviews, uninterrupted observation, bulletin boards, and ethnographic participation/observation

Quantitative research is an approach for testing objective theories by examining the relationship among variables. This type of research is a more logical approach that provides a measure of what people think from a statistical point of view. For example, if you wanted to know how many students use Android phone or services and how strongly they support it, you would do a quantitative research.

This research largely uses methods such as questionnaires and surveys with multiple choice questions where respondents are asked to select one or more of the given options. Answer options may include acceptance scale (strongly agree to strongly disagree), Likert scale, ranking in order of priority, etc.

This type of research can be conducted via telephone, web or with the help of paper questionnaires. The only constraint is that the number of respondents should be significant enough to be able to generate directional results.

Following Table 3.1 illustrates key differences between qualitative and quantitative research

Criteria	Qualitative	Quantitative
Data	Data is in the form of words, pictures or objects.	Data is in the form of numbers and statistics.
Method used	Methods include focus groups, in-depth interviews, and reviews of documents for non-numeric information	Surveys, structured interviews & observations, and reviews of records or documents for numeric information
Process of Research	Inductive approach used to formulate theory or hypotheses	Deductive approach used to test pre-specified concepts,

		constructs, and hypotheses
Response options	Unstructured or semi-structured response options	Fixed response options
Statistical test	No statistical tests are performed	Statistical tests are performed for analysis
Time required	Time spent at the time of planning is lower than that spent during the analysis phase	Time spent at the time of planning is higher than that spent during the analysis phase
Objectivity/subjectivity	Highly subjective	Highly objective
Result	Results depend on skill and accuracy of the researcher	Results depend on the measuring device / instrument
Final Report	Report contains textual details & verbatim from research participants	Report contains statistical analysis of data.

Table 3.1: Qualitative versus quantitative research

3.2 Method of Data Collection

- 1) Semi- structured interview technique: was used for data collection. The respondents chosen were from ISACA. I have a long association with an organization known as ISACA. ISACA stands for Information Systems Audit and Control Association. ISACA is an international organization which works for IT governance and auditing controls. It was formed in 1967 in USA by a group of people working as auditors and computer security personnel who wanted to have a centralized agency which could provide information and guidance in this field. Today it has its branches in 180 countries. It has more than 200 chapters. ISACA members include Information security auditors, Information security professionals, educators, consultants, regulators, CIOs, internal auditors etc. These chapters give education, provide for resource sharing, networking, advocacy etc. ISACA has a CPE- Continuing Professional Education Policy. All certified ISACA personnel have to attain a

minimum of 20 CPE hours for attaining knowledge about the latest technology and advancements in the IT security industry. This can be achieved by webinars, conferences, online and offline trainings etc. I was called as a speaker four times at these conferences to talk about my ongoing research on polymorphic shellcode. In other conferences, I attended as a participant. In all these conferences, I got a chance to interact with security professionals, auditors, CTOs and regulators. Sample size was 30.

These were conducted at every conference. There were not many formalized questions. More open ended questions enabled me to gauge the present security scenario and reach to my final research topic. The discussions allowed me to get clear understanding of the security risks that these professionals deal with every day, how they tackle these risks, what is the cost of applying anti malware solutions, what problems the current anti malwares systems can solve and more importantly what they cannot. They were also asked about their biggest security fears and what is it that is at the most risk. A sample question sequence was:

Q) What is the biggest security risk that you face as security personnel?

A) Fear of a malware, particularly ransomware attacking the organization's systems and encryption of all data.

Q) Which anti malware solutions do you use?

A) We use the latest anti malware solutions.

Q) Aren't those capable of handling these attacks?

A) They are quite efficient in handling known attacks but have sometimes failed in handling new, previously unknown malwares.

Q) What do you think is the reason for that?

A) Well maybe they are feigned by new malwares into assuming they are clean files.

Q) Is the security model centralized or distributive?

A) Well it is centralized.

2) Surveys: were conducted using both Google online forms and hard copy printed forms. Some 361 respondents filled these forms. Some were qualified security

professionals while some were not so qualified. Some were directly responsible for security in their organisations while others were not directly responsible. Variety of organisations was selected. Time was from Jan 2016 to Dec 2016.

- 3) In-Depth interview technique was used with either individual people or a focused set of interviewees. This technique was used because it helps in exploring the respondent's detailed perspective on a particular idea, program or solution. It is very useful when we want to explore new issues and ideas in depth. Laddering technique of question asking was used wherein one question led to another. This technique is particularly useful in the early stages of research as it provides a direction to the researcher.

These types of interviews and surveys helped me a lot in topic finalization and data collection. Each interview was roughly of 15-20 minutes. It would either be an individual interview or a focused group interview. The sample size was of about 25-30 people and snowball technique of sampling was used for sample selection where participants of an interview were asked to identify other potential participants and the chain goes on until the right sample size was found. Snowball sampling is particularly useful where the researcher is not able to find the required number of participants and also a situation where potential participants may be wary of disclosing their identities. Some security people may not want to accept that they were actually attacked by ransomware and they did pay the ransom to recover their encrypted files and data. Here the company's reputation is at risk.

3.3 Research Methodology used

Our research is **exploratory** in nature till the time we are finding the WHAT of the problem. It is also **descriptive** where we describe the WHY of the problem and **experimental** where we give the SOLUTION to the problem. Therefore, the study covers **all aspects of research**. Thus it is a **mixed research methodology**. We are trying to collect information regarding the problem which is being faced by the security industry today. The information is collected through in-depth interview techniques.

We also create our own advance malware and test it under common anti malware solutions to verify our hypotheses that common anti malware solutions are not able to detect advance

malware. Then we study the behavior of common ransomwares and advance malware to see their build, attack patterns and anti-detection strategies. Then we study different techniques used to create advance malwares because we cannot give a solution until we know the problem well.

We also study why common anti malware solutions are not able to detect advance malware and what are the reasons of their failure. The research is **quantitative** because we propose a framework which can detect these advance malware. We build the framework and test it against known and unknown samples. We first train the model and then test the model. We also run the model in a distributive environment and find out that the model is very efficient and cost effective as compared to the models existing in the industry today.

Since the respondents are a mixed bag of primary responsible, secondary responsible and not responsible (for cyber security) employees therefore, these are **perceived notions** rather than actual ones.

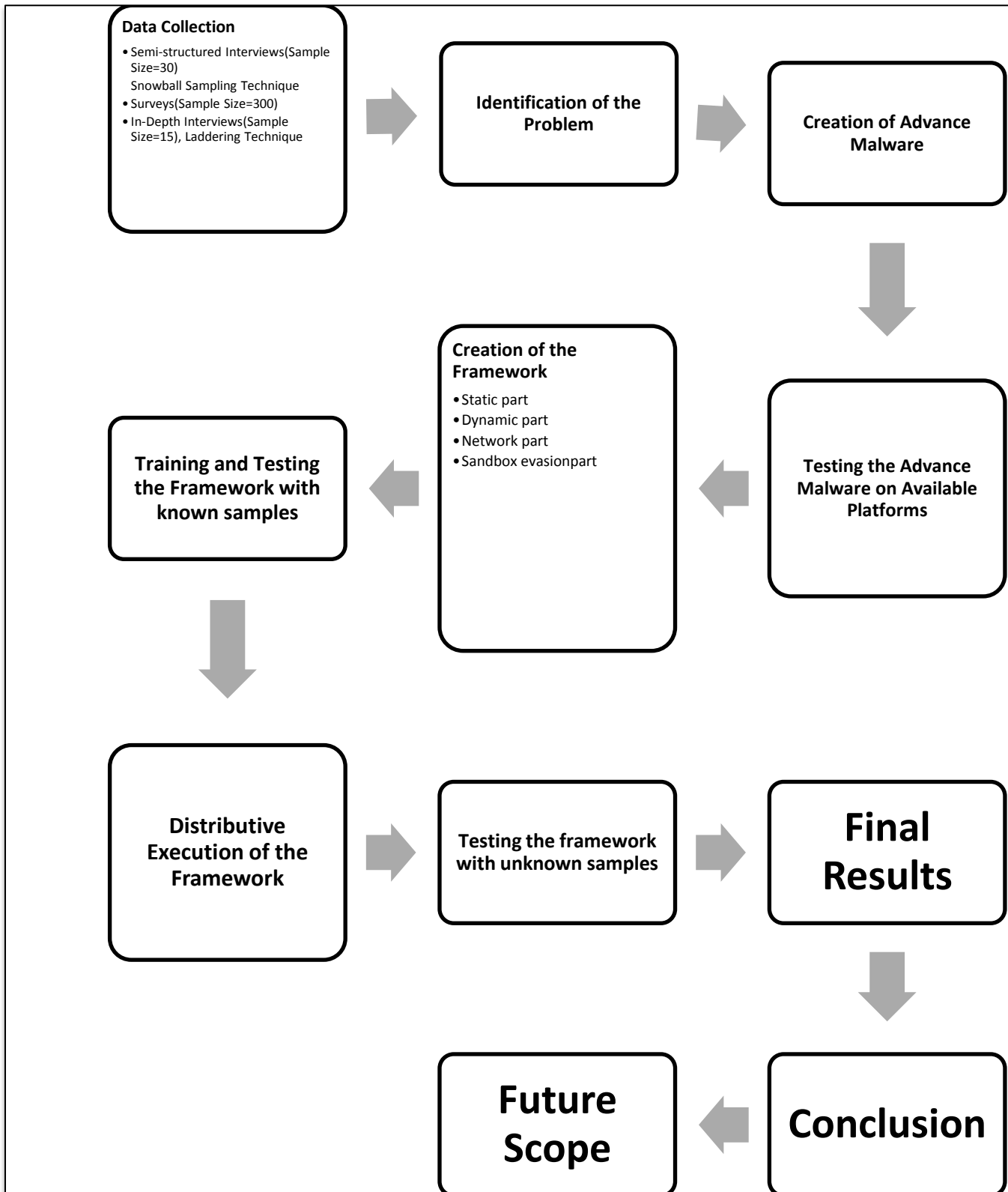


Fig. 3.1: Research Methodology

PREPAREDNESS OF ORGANISATIONS AGAINST CYBER MALICIOUS ACTIVITIES.

1. Email address *

2. Name

3. Email

4. Company Name

5. Location

Mark only one oval.

- New Delhi
- Bengaluru
- Gurugram
- Noida
- Kerala
- Mumbai
- Pune
- Hyderabad
- Other: _____

Fig.3.2: Survey Questionnaire(1/4)

6. Type of Organisation

Check all that apply.

- Telecommunication
- Technology Services
- Financial/Banking
- Education
- Healthcare
- Government
- Retail
- Transportation
- Other: _____

7. Which title do you hold in your organisation?

Mark only one oval.

- Executive Level
- Senior Level
- Middle Level
- Lower Level

8. What is your degree of responsibility for your organisation's cyber security?

Mark only one oval.

- Primary responsibility
- Secondary responsibility
- Sometimes responsible, whenever needed
- No responsibility

9. Which types of malicious activities were most common in your organisation over the past 3 years?

Check all that apply.

- Hacking
- Loss of intellectual property
- Online identity theft
- Ransomware attack
- Damage to hardware
- Denial of service
- Damage to software and applications
- Other: _____

Fig.3.2: Survey Questionnaire(2/4)

10. What is the most common origin of malware activity in your organisation?

Check all that apply.

- Email infiltration
- Email phishing
- Ransomware
- Social media phishing
- Web browser infiltration
- Insider breach
- Other: _____

11. What percent of overall budget is allocated in your company to cyber security?

Mark only one oval.

- <10%
- 10-20%
- 20-30%
- 30-40%
- 40-50%
- >50%

12. Till what level are devices normally affected in your organisation during a malicious attack?

Mark only one oval.

- Only endpoint
- End point and some more devices
- All devices in the network

13. What is the level of concern for cyber security in your organisation?

Mark only one oval.

- High
- Moderate
- Low
- No concern

14. What technologies are used in your organisation for protection against malicious attacks?

Check all that apply.

- Email security installation on all devices
- Anti virus and anti spywares
- Firewalls and IDPS
- Segmentation of the network
- Regular system and network scans
- Updated antiviruses and softwares
- Regular backup
- Outsourced security service provider
- Cloud malware detection

Fig.3.2: Survey Questionnaire(3/4)

15. Which approach is your organisation following for malware detection?

Check all that apply.

- Signature detection
- Static behavior detection
- Dynamic behavior detection
- Network behavior detection
- Don't know

16. What percentage of total resources is your organisation allocating to cyber security?

Mark only one oval.

- <10%
- 10-20%
- 20-30%
- 30-40%
- 40-50%
- >50%

17. What is the average response time of your organisation after a malicious attack?

Mark only one oval.

- <1 hour
- 1 day
- >1 day
- 1 week
- >1 week

18. What is the most important factor hindering cyber security efforts in your organisation?

Check all that apply.

- Resource crunch
- Insufficient training
- Lack of information
- New automated and AI driven attacks
- Others

Fig.3.2: Survey Questionnaire(4/4)

3.4 Analysis Report of Data Collected

The following table shows the companies covered for interviews and surveys, their locations and role of respondents.

S. No.	Organization	Location	Role
1	IBM	Gurgaon, Noida, Bangalore	Security Delivery Lead
2	HCL	New Delhi	Manager information Security
3	Punjab National Bank	New Delhi	Ex. IT Auditor
4	IT Birbal	New Delhi	CIO
5	EY (Ernst & Young)	Trivandarum, Kerala	Information Security Analyst
6	BHEJONA powered by Stadhawk Group	Gurgaon	Head, Information Security Operations
7	Tipping Edge Consulting Pvt Ltd.	Noida	CIO
8	MERI College	New Delhi	Vice President
9	Sedulity Solutions Pvt Ltd	New Delhi, Pune	CEO
10	Deloitte	Gurgaon	Director
11	Internet & Mobile Association of India	Noida	Cyber Security Consultant
12	Network Solutions Pvt Ltd	Bangalore	IT Security Lead
13	Mother Dairy Fruit & Vegetable Pvt Ltd	New Delhi	CIO
14	Cvent	Gurgaon	Senior Manager, Information Sec
15	Blue Pi Consulting Pvt Ltd	Gurgaon	Software Engineer
16	Dalmia Bharat Group	New Delhi	Head - SAP Audits and IT Controls
17	Axis Risk Consulting	New Delhi	Manager information Security
18	Goldman Sachs	Bangalore	Internal Auditor
19	Canara Bank	Bangalore	Chief Information Security Officer
20	Supreme Court Of India	New Delhi	Advocate and Cyber Evangelist
21	WestSide	New Delhi	Operations Manager
22	Concentrix	Gurgaon, Noida, Bangalore	Head Operations
23	TCS	Mumbai	IT Security Lead
24	Tech Mahindra	Pune	Senior Manager, Information Sec
25	Mindtree	Bangalore	Manager information Security
26	Mphasis	Bangalore	Internal Auditor
27	Rolta	Mumbai	Information Security Analyst
28	Hexaware Technologies	Mumbai	IT Security Lead
29	Cognizant	Hyderabad	Internal Auditor
30	Sonata Software	Bangalore	CIO

Table 3.2: Companies Covered, location and role of respondents during Data Collection and Analysis

3.4.1 Type of Organisation

Among all the respondents, the companies that are majorly targeted are Telecommunications, Technology and Financial/Banking services. Though malicious attacks affect all types of organisations, however, the above type of organisations offer lucrative resources to the attackers. Thus these organisations form 77% of our respondent organisations.

Type of Organisation	Count
Telecommunication	9
Technology Services	8
Financial/Banking	6
Education	1
Healthcare	3
Government	1
Retail	2
Total	30

Table 3.3: Type of organisations of respondents

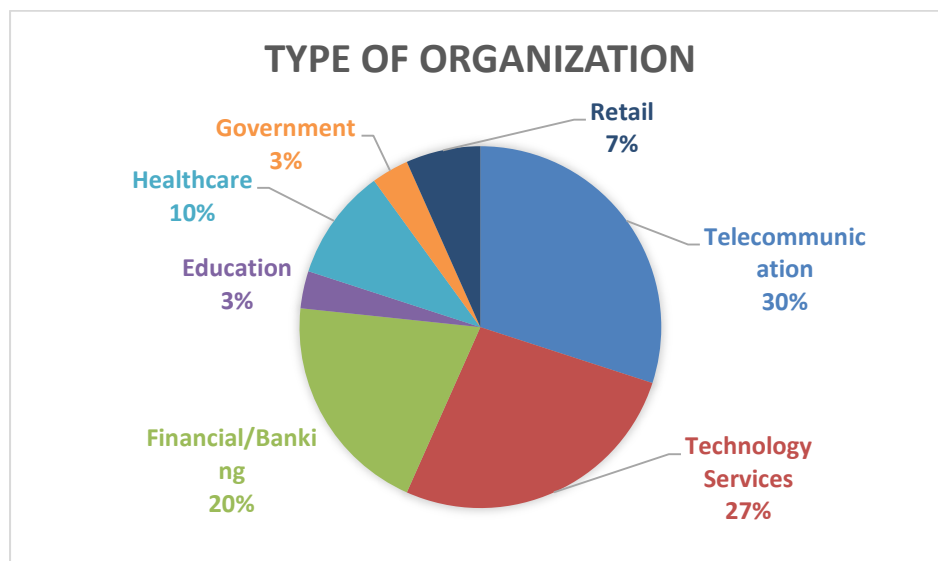


Fig.3.3: Companies Covered, location and role of respondents during Data Collection and Analysis

3.4.2 Location of organisations

Though malicious attacks affect entire India, but Delhi, Gurugram and Bengaluru are prime affected areas. Although Mumbai and Hyderabad are also very important metro cities, 80%

of the respondents are from Delhi,

Gurugram and Bengaluru. Some respondents were given online questionnaires to fill, some filled them in hard copy, printed formats. Some values were taken during interviews.

Location	Count
New Delhi	123
Gurugram	113
Bengaluru	77
Mumbai	44
Pune	24
Hyderabad	11
Total	392

Table 3.4: Count of locations of respondents

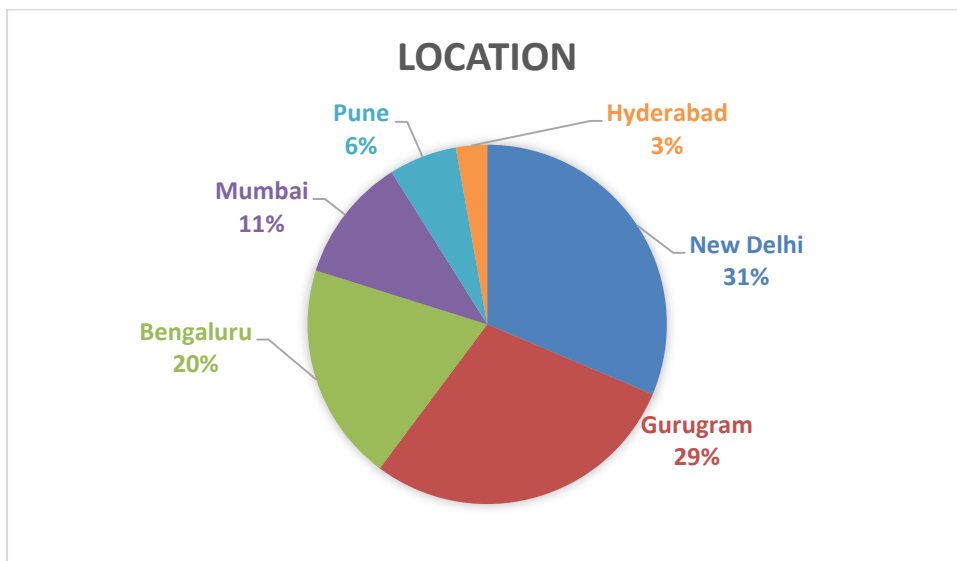


Fig.3.4 Location of organisations

3.4.3 Title level held in the organization

Out of all the respondents, 45% were senior level personnel. 35% were middle level and 16% were executive level. These were decision makers in the organisation and were most affected during a malicious attack. Also, they were primarily responsible for making security policies and compliances to be followed by the entire organization.

Title level	Count
Executive Level	63
Senior Level	176
Middle Level	138
Lower Level	15
Total	392

Table 3.5: Count of title level of respondents

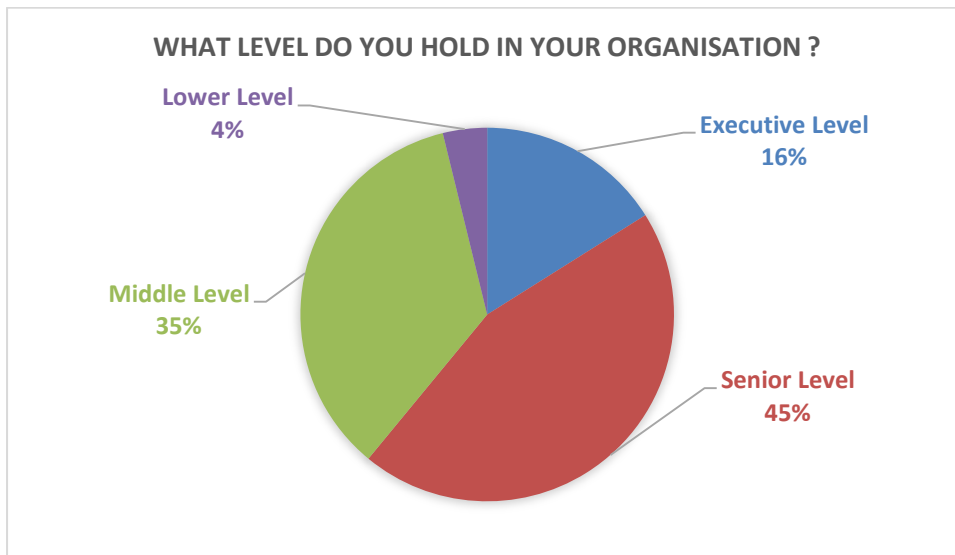


Fig.3.5: Title level held in the organisation

3.4.4 Degree of security responsibility

Out of all the respondents, 49% were primary responsible for security in their organisations. 46% held secondary responsibility while only 8% were those who were not always responsible for security but only at some times.

Degree of responsibility	Count
Primary responsibility	194
Secondary responsibility	167
Sometimes responsible, whenever needed	31
No responsibility	0
Total	392

Table 3.6: Count of degree of responsibilities of respondents

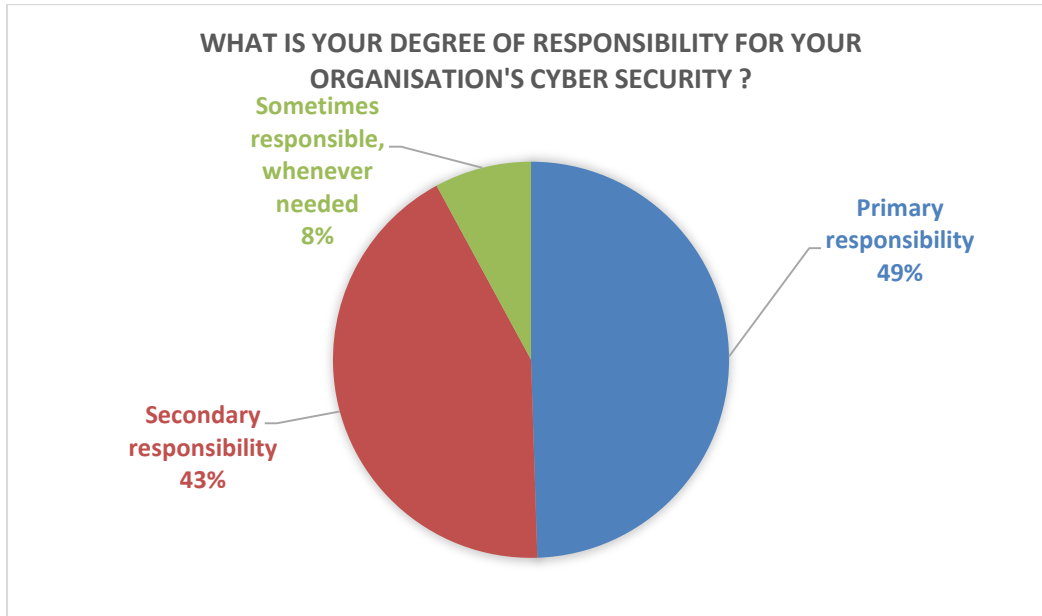


Fig.3.6: Degree of security responsibility

3.4.5 Types of malicious activities

There are many types of malicious activities which are encountered in an organization. Out of these, 53% are hacking activities. 51% of the times, ransomware attacks happen which are primarily for financial gains. Only 10% damage the software whereas 18% damaged software and applications. Three years mean 2013, 2014 and 2015

Type of malicious activity	Count
Denial of service	9
Damage to hardware	38
Loss of intellectual property	56
Online identity theft	61
Damage to software and applications	66
Ransomware attack	186
Hacking	191
Total	607

Table 3.7: Count of type of malicious activities in respondents organisations

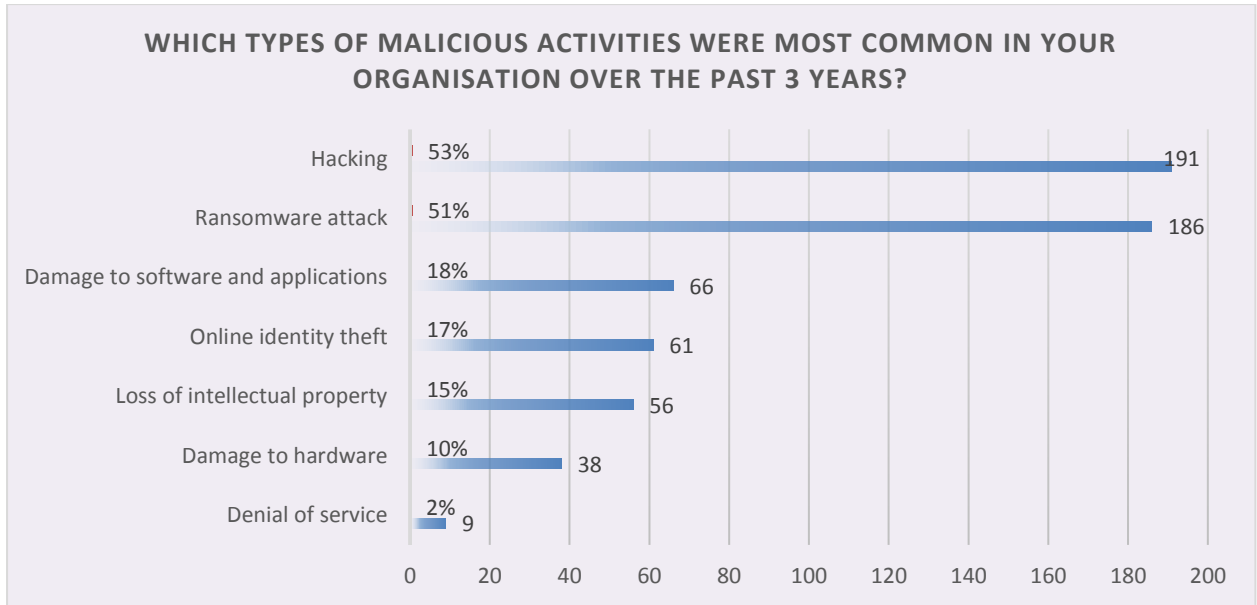


Fig.3.7: Type of malicious activities

3.4.6 Origin of malicious activity

Most of the malicious activity originates through emails either by infiltration(56%) or by phishing(27%). Ransomware accounts for 52% malicious activity whereas insiders do not normally breach security.

Origin	Count
Insider breach	2
Web browser infiltration	23
Social media phishing	89
Email infiltration	201
Ransomware	188
Email phishing	99
Total	602

Table 3.8: Count of origin of malicious activities in organisations

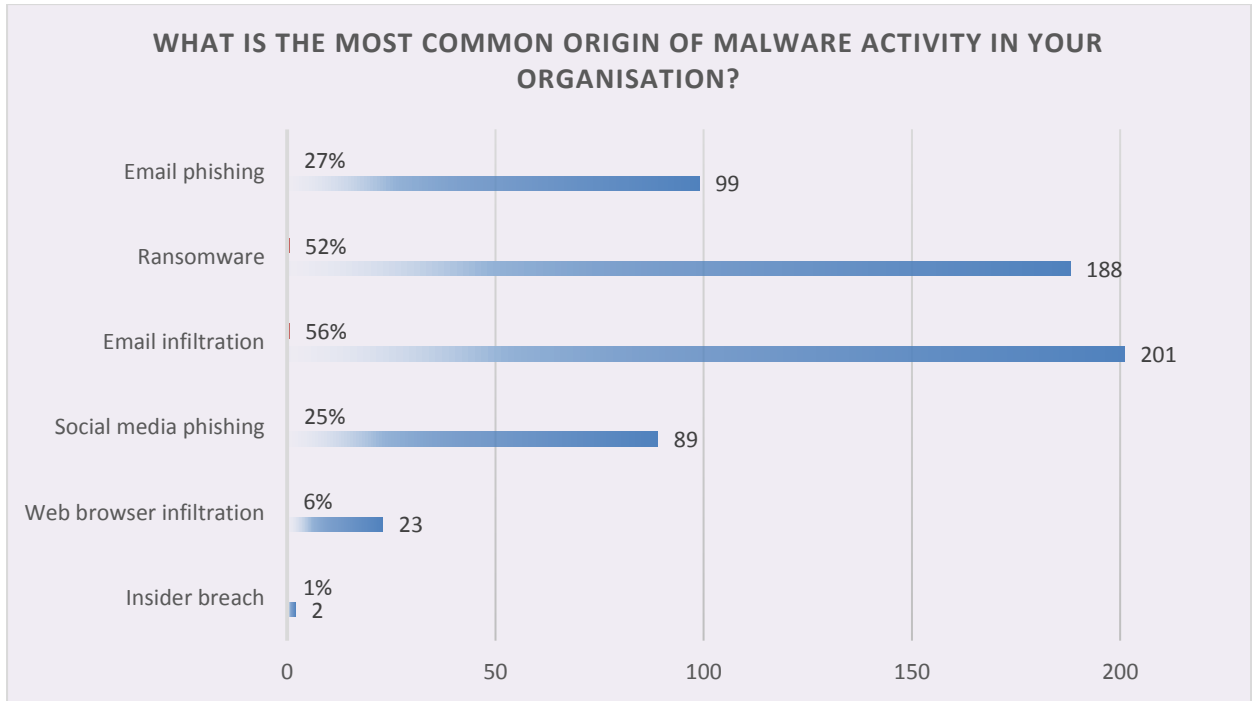


Fig.3.8: Origin of malicious activity

3.4.7 Percentage of Overall IT budget allocated to cyber security

Mostly companies (74%) allocate less than 10% of the overall IT budget to security. Only 20% companies allocate 10-20% budget to security and in all the respondents there was not a single company allocating more than 30% budget to security.

% of Total budget	Count
>30%	0
20-30%	21
10-20%	79
<10%	292

Table 3.9: Percentage of Overall IT budget allocated to cyber security

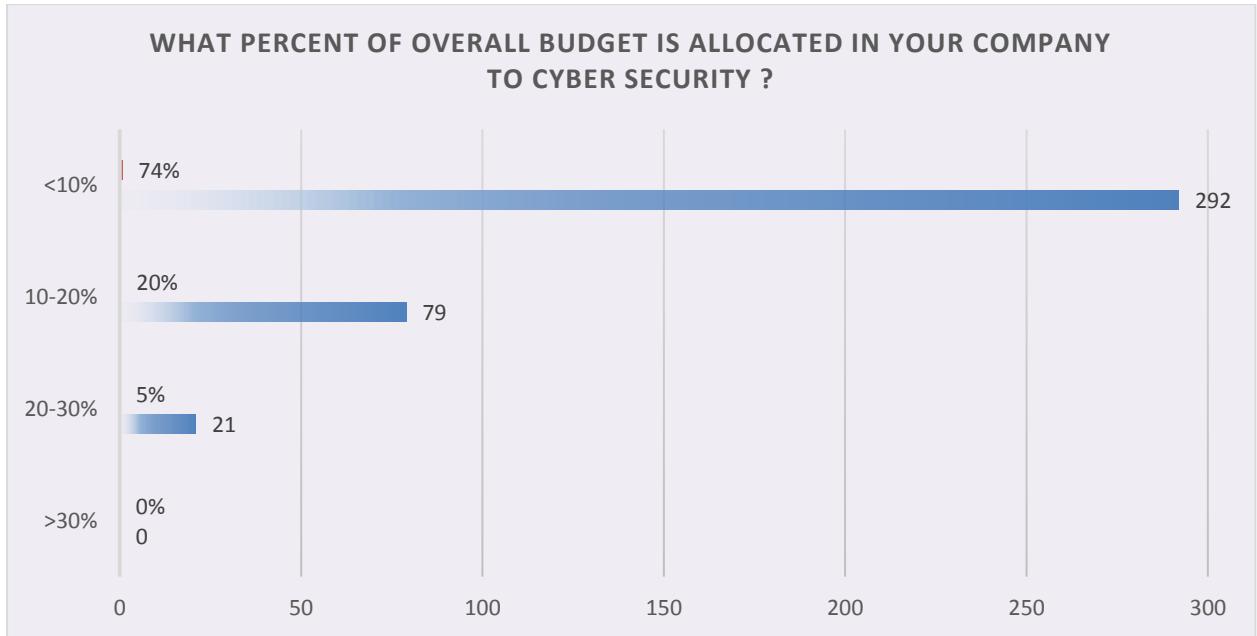


Fig.3.9: Percentage of budget allocated for cyber security

3.4.8 Level till which devices get affected

When a malicious attack happens, 72% of the times it is contained in the end point only. 28% of the time the infection spreads to some other systems as well but not to the entire network, probably due to network segmentation. Only in 8% of the cases, the infection spreads to the entire network.

Level	Count
All devices in the network	30
End point and some more devices	101
Only endpoint	261
Total	392

Table 3.10: Count of level till which devices get affected

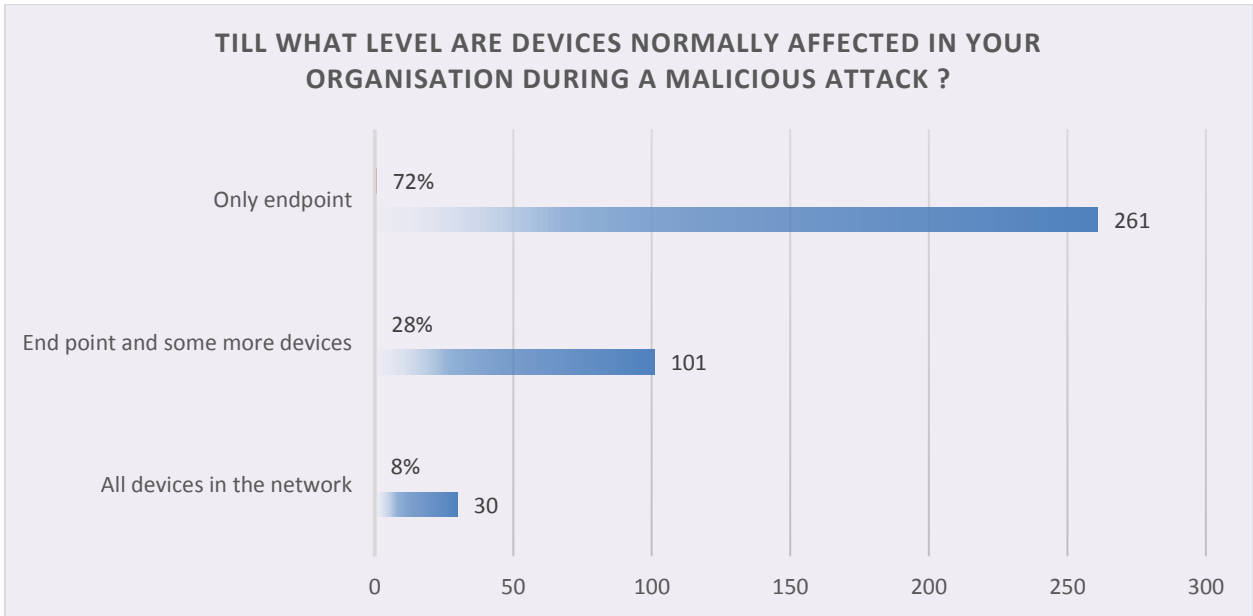


Fig.3.10: Level till which devices get affected

3.4.9 Level of concern for cyber security

62% of the respondents believe, organisations have moderate level of concern for security. Only 32% organisations have a high concern whereas 6% have low concern for security.

Level of concern	Count
Low	24
Moderate	244
High	124
No concern	392

Table 3.11: Level of concern for cyber security

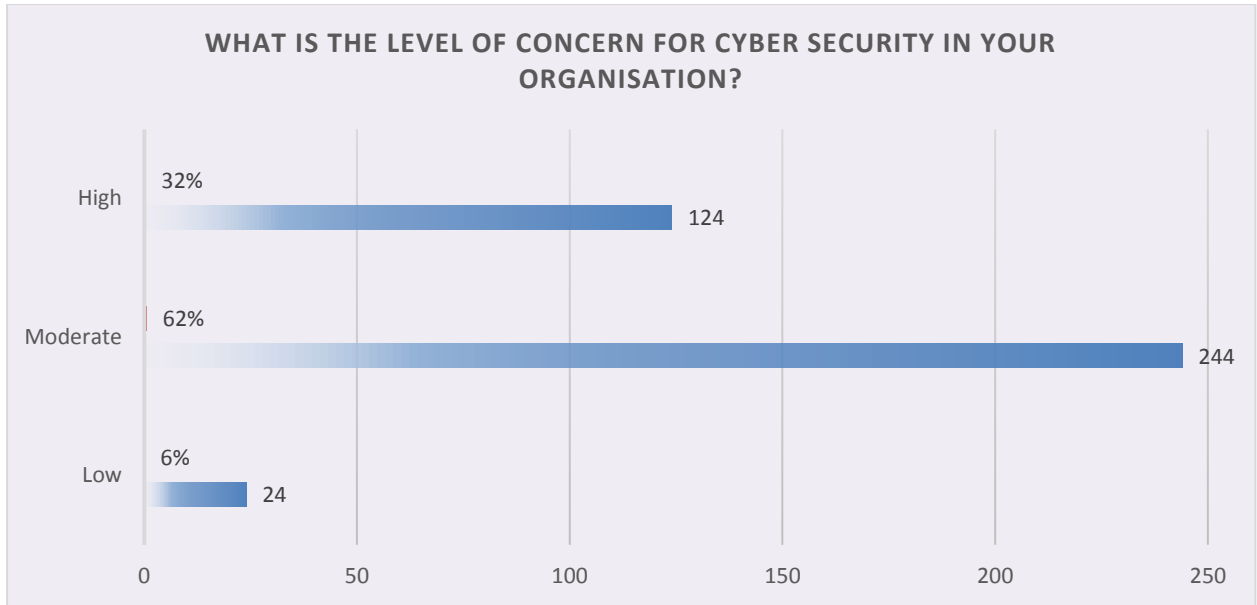


Fig.3.11: Level of concern for cyber security

3.4.10 Technologies used for protection

Out of the total organisations covered, 86% have anti virus and anti spyware technologies installed. 77% have email security both in servers and all systems. 74% have updated anti virus and applications. Only 27% have cloud malware detection and 6% have outsourced security providers.

Technologies used	Count
Outsourced security service provider	22
Regular backup	78
Cloud malware detection	98
Segmentation of the network	109
Firewalls and IDPS	207
Regular system and network scans	221
Updated antiviruses and softwares	268
Email security installation on all devices	280
Anti virus and anti spywares	311

Table 3.12: Technologies used for protection

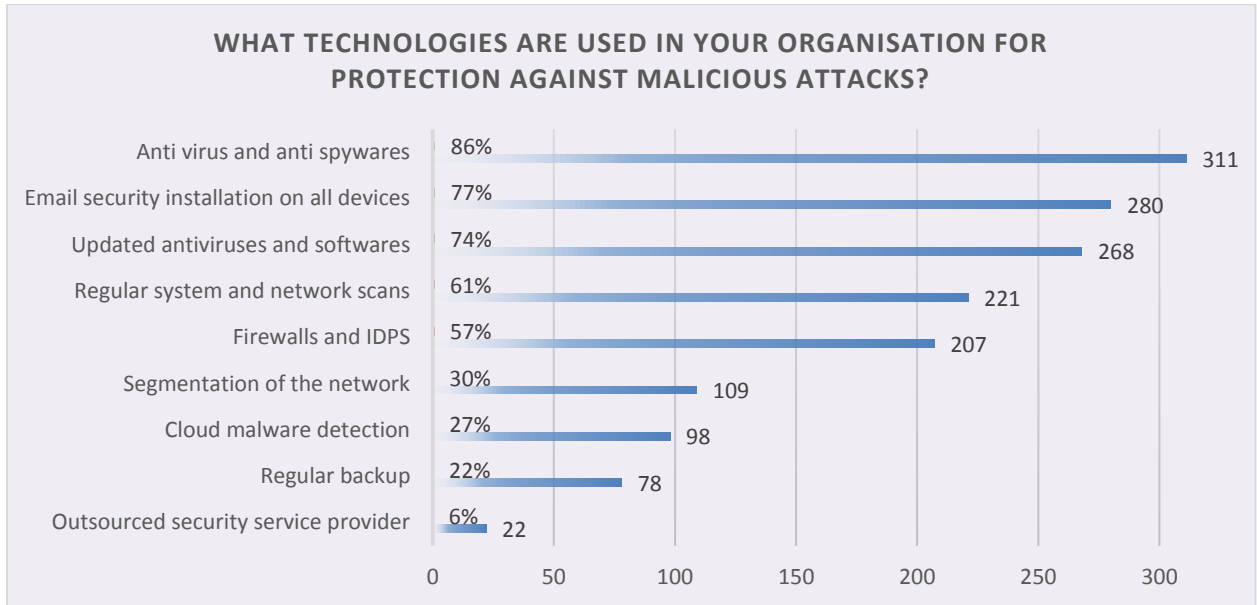


Fig.3.12: Technologies used for protection

3.4.11 Approach followed for malware detection

Out of the total respondents, 80% use signature detection approach. Almost same percentage use behavior (58%) and static (52%) detection.

Approach	Count
Network behavior detection	126
Static behavior detection	190
Dynamic behavior detection	211
Signature detection	289
Don't know	3
Total	819

Table 3.13: Approach followed for malware detection

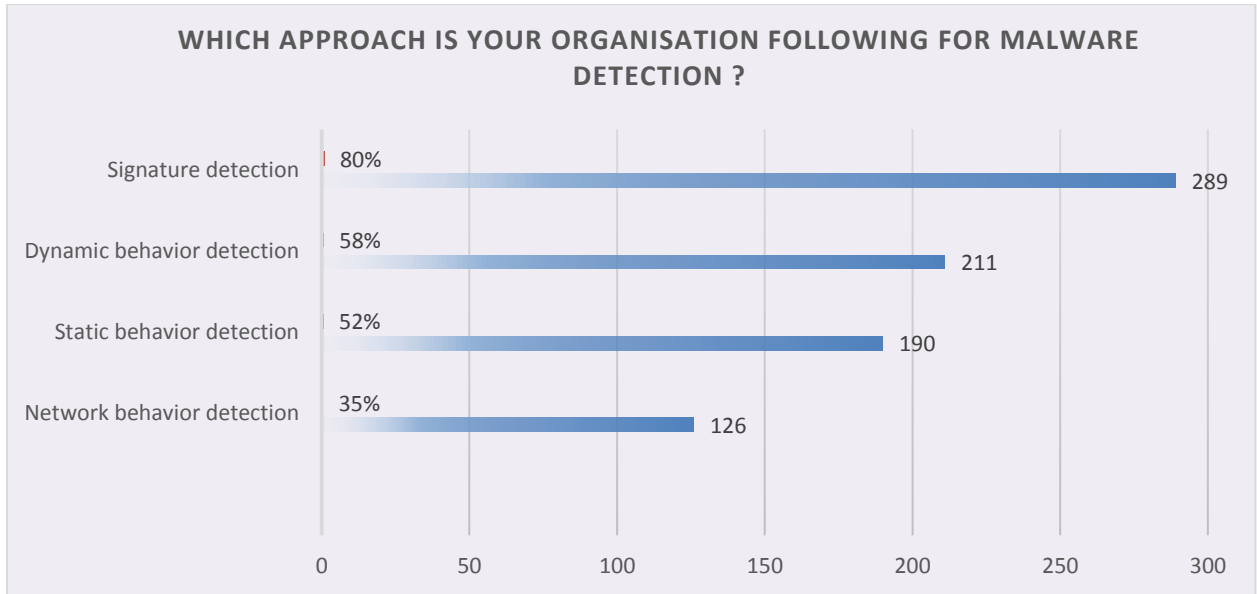


Fig.3.13: Approach followed for malware detection

3.4.12 Resource allocation to cyber security

Out of the total number of respondents, 74% say that their organisations spend less than 10% of the total resources on security. 19% say the percentage to be between 10-20%. 7% say their organisations spend 20-30% resources on cyber security and none of them say their organisations spend more than 30% resources on cyber security.

Resource allocation	Count
>30%	0
20-30%	28
10-20%	73
<10%	291
Total	392

Table 3.14: Resource allocation to cyber security

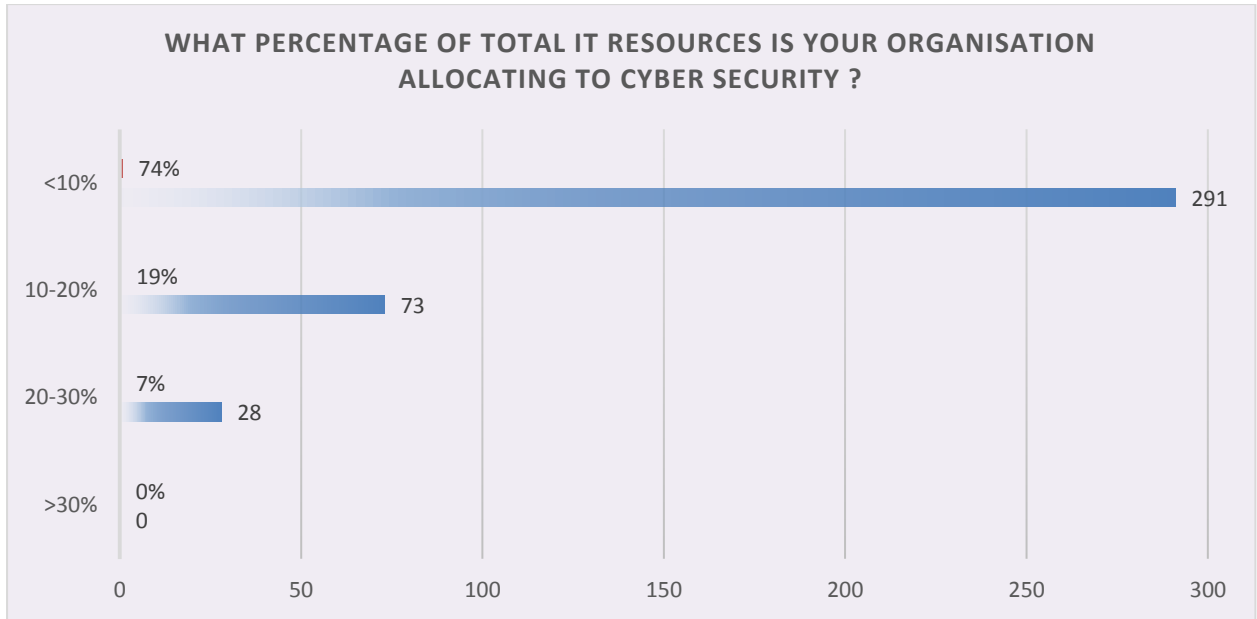


Fig.3.14: Resource allocation to cyber security

3.4.13 Response time after malicious attack

Out of the total number of respondents, 56% say that their security teams take more than a day to respond to malicious attacks. 31% say the team responds within a day. 13% say the response time is less than an hour. Only 5% say it takes more than a week to respond to malicious attacks.

Response time	Count
>= 1 week	18
>1 day	219
<1 hour	51
1 day	122
Total	392

Table 3.15: Response time after malicious attack

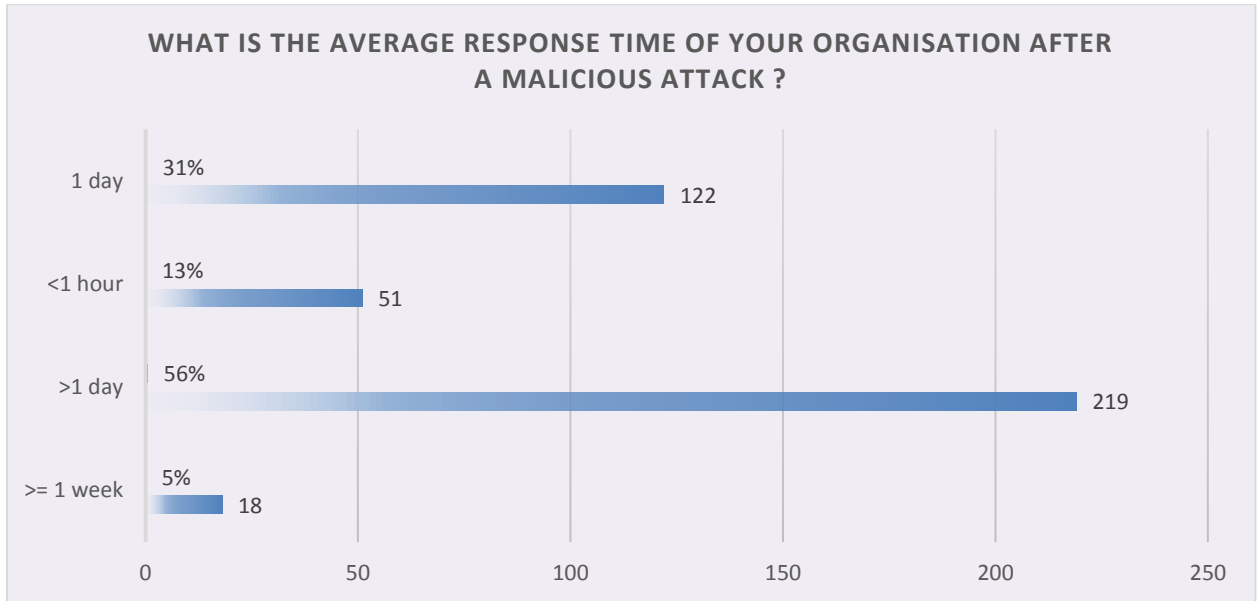


Fig.3.15: Response time after malicious attack

3.4.14 Factors hindering cyber security efforts

Out of the total number of respondents, 85% say resource crunch is the main reason hindering cyber security efforts in their organisations. 80% cite lack of information as the reason. 68% blame insufficient training to be the reason and a percentage of 45% believe new automated and AI driven attacks are the reasons and they are not prepared for such advance attacks.

Factors	Count
New automated and AI driven attacks	164
Resource crunch	307
Lack of information	288
Insufficient training	246
Others	0

Table 3.16: Factors hindering cyber security efforts

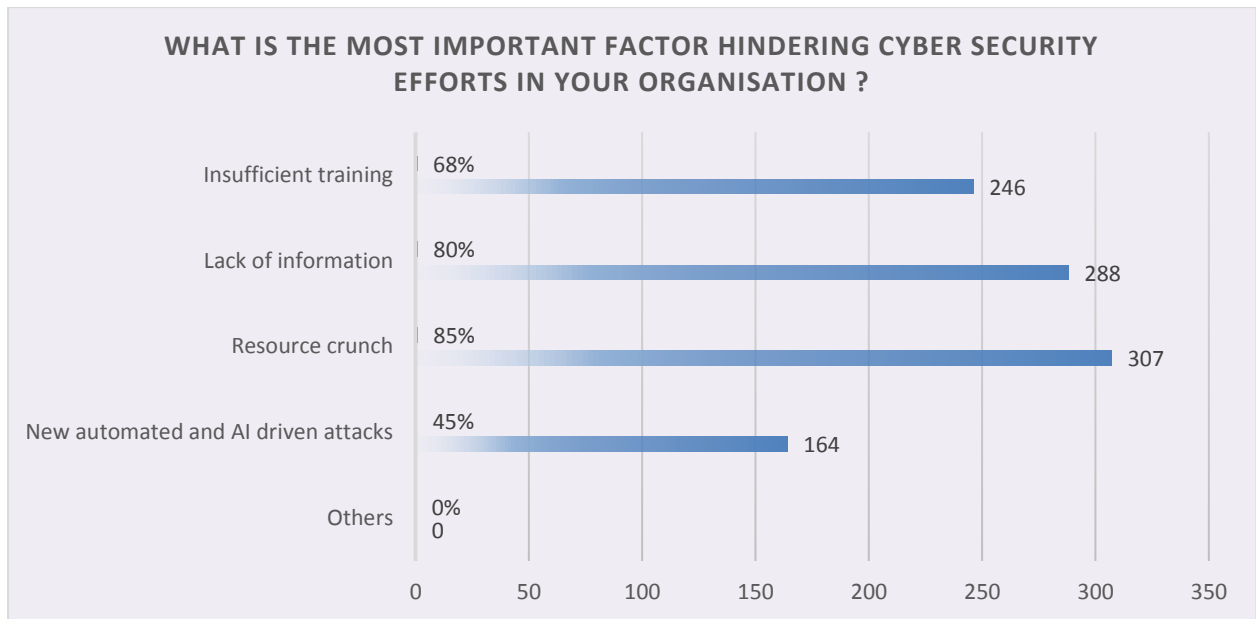


Fig.3.16: Factors hindering cyber security efforts in the organisation

3.5 Key Understandings

The key understandings that we have got after analysis of the collected data are as follows:

- 1) Organisations allocate very **less percentage of overall budgets** to cyber security.
- 2) Organisations face a **resource crunch** when dealing with advance malware.
- 3) Security persons are sometimes **unaware** of impending malicious attacks and futuristic attack mechanisms.
- 4) **Lack of regular training** amongst employees lead them to become weak against malicious attacks.
- 5) Attackers are using **advance AI and ML techniques** for creation of malware whereas the current security mechanisms are unable to handle them.
- 6) The average **response time of security team is very high** which leads to infection spreading in the entire organization.
- 7) The **level of concern** for cyber security in organisations is low to moderate.
- 8) Organisations are **not using a combined approach** of static, dynamic, network and sandbox evasion for detection of malware.
- 9) **Centralized security systems** make mechanisms slow.

3.6 Objectives of the Study

Internet is full of versatile malware, shellcode is the most common technology used by hackers nowadays. Intrusion detection system is able to mitigate the risk level up to certain limit because it relies on signature detection methodology. That's why the new version of shellcode is polymorphic due to which to detect the root cause is almost impossible in the existing detection technique. Due to such scenario we have identified the research gap in the current research and have proposed a framework to detect and mitigate untraced polymorphic shellcode.

3.6.1 The Theoretical Framework of the Present Study

We basically have to design a framework which can attack a shellcode from four different angles. We have to do a static analysis of shellcode, then a dynamic (behavior) analysis, then study its network footprint and finally try to see if it evades the sandbox. Then we aggregate the result obtained from these four stages and give the final result.

3.6.2 Conceptual Model Framework

Our conceptual model framework is a design created with the aim to identify and remove risks of polymorphic shellcodes. It uses advance machine learning algorithms for detection. It is going to attack a shellcode, in fact any type of malware from 4 sides-static, behavior, network and sandbox evasion:

- 1) Static analysis: Means that we will study the static behavior of the sample and match it with already given behavior in our existing databases. If it matches with a malware we calculate the static probability of the sample for maliciousness. The machine learning model will be trained and tested with huge, ever increasing data.
 - 2) Behavior Analysis: There are advance malware like polymorphic and metamorphic malware which change their signatures quite often. In that case, we generally study
-

the behavior of the sample to see if it matches with the behavior that normally malware families show. If it does, we again calculate the behavior probability of the sample for maliciousness.

- 3) Network Analysis: Now we will see the network behavior of the sample to match it with the alerts that network monitors generate. If we see significant number of alerts, we again calculate the network probability of the sample for maliciousness.
 - 4) Sandbox Evasion Detection: Normally, an intrusion detection system will run a doubtful program in a controlled environment which is normally a sandbox. This environment will have all the resources required for the sample to run and all permissions are also given to the file. However, it is not allowed to run in an actual system, it is only run in a virtual system. This is a smart way to know the true characteristics of the sample and judge whether the file is malicious or benign before it enters the actual system environment. However, there are advanced malware which are created to stay dormant whenever they detect that they are running in a virtual environment. There are many techniques which are applied for the detection of sandbox. Thus they try to evade a sandbox whenever they experience one. Therefore it is with surety one can declare a sample as a malware if it tries to evade a sandbox. This detection will give us a Boolean value of 0 or 1.
 - 5) Final Detection: From all the above four steps, we are going to get four probabilities. An average of these four probabilities is calculated to get the final detection probability. A threshold probability is calculated for the model according to the amount and variance of the data it currently has. A sample is declared as malicious if its combined average probability is above the threshold probability of the model. It is benign if the probability is lower. However, if the probability is equal to the threshold value, the sample is termed as unknown and is sent to the model again for further analysis. The model is cyclic in nature and every new sample is used as data for training of the model to make it more effective. Therefore it will always be in an updated state. It will have information about all new malwares all the time.
 - 6) Decentralized Environment: The whole model is going to work in a decentralized
-

environment. Thus the model is very resource efficient. Also because the detection mechanism is completely decentralized, therefore, every node is master node in the network and there is no central authority which can work for selfish gains or unrealistic profits.

- 7) Blockchain network: All the transactions for detection of malware are logged in Blockchain network and the data about the malwares which is stored in each node is stored in the form of blocks in the block chain. Thus this database is immutable, secure, reliable, distributed and always updated. Thus every node participating in the malware detection model gets an always updated data at a very cost effective rate with the latest of machine learning technology.
 - 8) Balancing the model: the model has to be always in a balanced state so that is never biased with more of clean or more of malicious data. Whenever we find such a bias in the model according to our calculations, we either increase the clean samples or malicious samples for training of the model.
-

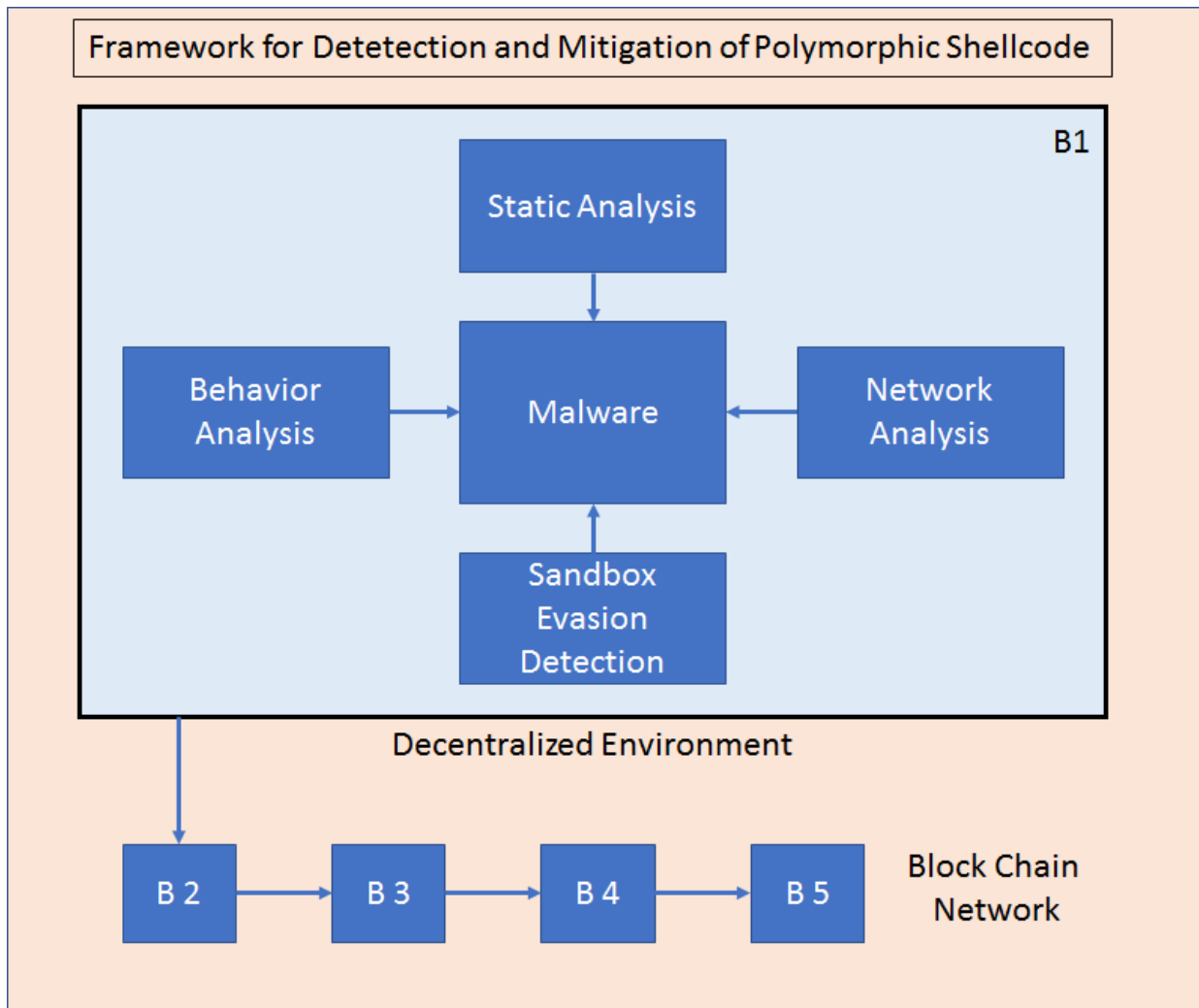


Figure 3.17 Framework for Detection and Mitigation of Polymorphic Shellcode

3.6.3 Objectives of the Framework:

To identify and the mitigate risk factors occurring due to the execution of polymorphic shellcode.

- 1) Injecting the binaries in the code caves of existing (PE) application by using binding, embedding and stubs methodology. This will create polymorphic shellcodes for our study.
- 2) Understanding polymorphic and metamorphic techniques of creating advanced malware.
- 3) Understanding the working of polymorphic shellcode including the attack methodology, hiding methodology replication methodology and metamorphosis.

- 4) Working with live ransomware samples to understand their behavior and attack methodology.
- 5) The static analysis, behavior analysis, network analysis and sandbox evasion detection of polymorphic shellcode on the host as well as on the network.
- 6) Proposing the framework for the detection and mitigation of untraced polymorphic shellcodes using advanced machine learning techniques.
- 7) Proposing how this framework would work best in a decentralized environment.
- 8) Proposing blockchain network usage for recording and logging all transactions to make them secure, updated and reliable.
- 9) Training the framework with already known samples.
- 10) Testing the framework with live samples from the wild.
- 11) Analysis of the proposed framework with its limitation and future scope.

3.7 Experiments to assess Polymorphic shellcode threat:

1. Exp 1: Creating a Shellcode by Smashing the Stack
2. Exp 2: Injecting shellcode in PE file and making it polymorphic thus making it undetectable.
3. Exp 3: Getting privileges after infecting a file with polymorphic shellcode, then running it through antiviruses to get results.

3.7.1 Experiment 1 : Creating a Shellcode by Smashing the Stack

Purpose- Exploiting a program having a buffer overflow vulnerability. Overflowing the buffer by giving a very large input. Then overwriting the 'Return Address' to point to a malicious shellcode which will then give admin access of the system to us.

Hypothesis- The hypothesis is that in some vulnerable programs where bounds checking of the buffer is not strictly performed, the buffer can be overflowed to point to the malicious code. In case, the return address cannot be exactly determined, NOP instructions are added before the malicious code.

Explanation- Buffer overflow exploits are accomplished by mangling the way that C handles memory allocation. When a program in C begins, or starts a function, it allocates a stack of memory for that particular piece of the program. This stack consists of space for variables and data, as well as pointers to return flow control to the proper place in the stack. This allows stacks to grow dynamically as programs fork and carry out subroutines and other processes. This is efficient because the stack doesn't have to be initialized at the start of the program with room for every possible execution path of the program. Instead, as the program runs, memory is allocated on a per needed basis.

Programs don't run in a vacuum, however, and one process can't be allowed to own the stack entirely until its completion. For this reason the return pointer on these individual pieces of the stack (called stack frames) is critical, so that at the end of the frame execution the processor can return to the original programmatic instructions and continue the program.

Because these frames are allocated dynamically and because they are of a fixed size, if a programmer is not careful it becomes possible to pass in more variable data than is reserved on the stack. For instance, if the following represents a frame:

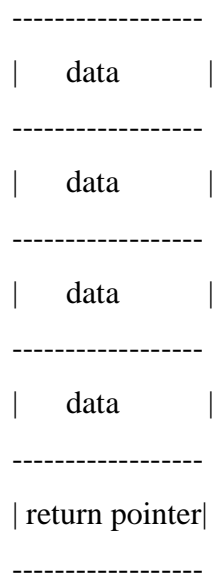


Figure 3.18 Representation of a Stack Frame

You can see that there are 5 'slots' for data in the frame; the sixth slot is for the return pointer. What happens if the program tries to write 6 'slots' of data into the frame? An exception probably, but if the attacker is careful they could arbitrarily send the pointer to a different location in memory, perhaps a location that contains malicious code.

Procedure- The first step when we experiment with shell codes is to create a vulnerable program so let's create one in c

```
#include<stdio.h>//lets include some basic library files
#include<string.h>
#include<stdlib.h>

// IN int main function in gcc the second argument is to accept the //arguments
// As you can see there are no checks made which processing the arguments // hence the
program is vulnerable
intmain(int argc,char*argv[])
{

char buffer[256];// array of 256 characters is pretty standard
strcpy(buffer,argv[1]);
printf("%s\n",buffer);
return0;

}
```

Figure 3.19 Creating a vulnerable program

Let's compile the following code with the following command

```
gcc vulnerable.c -o vuln -z execstack -fno-stack-protector
```

Make sure to disable some stack protection of gcc compiler that's why we have used some parameters

```
-z execstack -fno-stack-protector .
```

After compiling the code successfully we can execute the program in linux with / operator followed by the '.' operator as -

```
./vunl helloworld
```

Output of the following program will be the argument that was entered during the execution of the program

Hello world

The most effective way to do this is to pass in malicious bytecode as part of the 'data' and then overwrite the return pointer with the location of the malicious bytecode. Even this process is tricky though, because the return pointer must point to the exact location of the exploit code or the code will fail. For instance, if the pointer lands in the middle of the exploit code it won't execute properly. A neat trick is to pad the start of the exploit shellcode with NOP (no operation) instructions. When the machine encounters a NOP it simply moves to the next instruction. If there are a series of NOP instructions preceding the malicious shell code then the pointer merely has to hit one of them, and then the instructions will cascade down the NOP's to the shellcode. This technique is called a NOP sled.

Now before we try to crash the program with our infamous buffer overflow attack there is a program called **ulimit** which restrict the length of the argument that is being passed.

A simple way to check what the size ulimit is allows is just entering the ulimit in your linux terminal.

Ulimit

Unlimited

First of all there is a command in the gdb called `i r` which provides us the information about the system registers

As we can see a register called `rbp` at 7th position which means register which contains value of the base variable in stack this case its `0x68686868`.

Now a command `x/20x $rsp` is entered.

Here `rsp` is the register contains the pointer to the stack variable and `x/20x` is the way to print the information.

Now as it can be seen in the above screenshot after printing the information of stack pointer we got

```
0x7ffe8ae94af8: 0x68686868 0x68686868 0x68686868 0x68686868
0x7ffe8ae94b08: 0x68686868 0x68686868 0x68686868 0x68686868
0x7ffe8ae94b18: 0x68686868 0x00000000 0x00000000 0x00000000
0x7ffe8ae94b28: 0x1d6a8fd3 0xecdf79e3 0x00400470 0x00000000
0x7ffe8ae94b38: 0x8ae94bd0 0x00007ffe 0x00000000 0x00000000
```

Figure 3.22 Information of stack pointer

Now after doing this we got the address where our core was dumped we are going to make the note of this address for future purpose

`0x7ffe8ae94af8`

After that we have to find the beginning of our stack so we have to use some sort of brute force approach here so let's execute another gdb command

```
x/20x $rsp -300
```

0x7ffe8ae949cc:	0x00007ffe	0x00400470	0x00000000	0x004005ad
0x7ffe8ae949dc:	0x00000000	0x8ae94bd8	0x00007ffe	0xa50fe8f6
0x7ffe8ae949ec:	0x00000002	0x68686868	0x68686868	0x68686868
0x7ffe8ae949fc:	0x68686868	0x68686868	0x68686868	0x68686868
0x7ffe8ae94a0c:	0x68686868	0x68686868	0x68686868	0x68686868

The program has started finding the values although to find the actual address.

x/20x \$rsp -312

0x7ffe8ae949c0:	0x00000000	0x00000000	0x8ae94af0	0x00007ffe
0x7ffe8ae949d0:	0x00400470	0x00000000	0x004005ad	0x00000000
0x7ffe8ae949e0:	0x8ae94bd8	0x00007ffe	0xa50fe8f6	0x00000002
0x7ffe8ae949f0:	0x68686868	0x68686868	0x68686868	0x68686868
0x7ffe8ae94a00:	0x68686868	0x68686868	0x68686868	0x68686868

0x7ffe8ae949c0:	0x00000000	0x00000000	0x8ae94af0	0x00007ffe
0x7ffe8ae949d0:	0x00400470	0x00000000	0x004005ad	0x00000000
0x7ffe8ae949e0:	0x8ae94bd8	0x00007ffe	0xa50fe8f6	0x00000002
0x7ffe8ae949f0:	0x68686868	0x68686868	0x68686868	0x68686868
0x7ffe8ae94a00:	0x68686868	0x68686868	0x68686868	0x68686868

Here the address of beginning of stack is found.

Figure 3.23 Finding beginning of the stack.

So now we got the address of the end and beginning of our stack time to do some math and find out how much big our buffer is now because Linux shell is too powerful we can do that in single command

```
echo 'ibase=16;' $(echo '7ffe8ae94af8-7ffe8ae949f0'|tr "a-z""A-Z")|bc
```

This command is understood as follows:

Echo is used to output stuff to standard output and `ibase=16` is used to tell `bc` (bash calculator) that we are trying to enter base 16 value then again we are subtracting the address that we have noted previously after removing `0X` from them also we changed the cases from lower to upper by piping the output of `echo` command to another tool called `tr` (transform cases) and finally piped it to `bc`.

The output is:

```
264 which is the size of our stack
```

Observation & Results- In the experiment, it was observed that a buffer in a vulnerable program can indeed be overflowed and the return address can be overwritten to point to the malicious code. Once we are able to reach the malicious code, we can get the admin rights of the attacked machine and thus can use the machine for any malicious purpose.

3.7.2 Experiment 2 : Injecting a polymorphic shellcode in PE file

Purpose:Injecting a process executable file from the Windows binary file with a polymorphic shellcode and scanning this malicious file against a list of antiviruses.

Hypothesis: A hypothesis can be formed that since many antiviruses do not have the required capability to detect dynamically generated polymorphic malwares, thus more than fifty percent of the antivirus software will not be able to detect it.

Explanation- Once a PE file is injected with a polymorphic shellcode, it will become very difficult for the antiviruses to detect it. Signature matching fails and polymorphism is dynamic. In this experiment, we are going to use a tool called Shellter. Shellter is a dynamic shellcode injection tool, and the first truly dynamic PE infector ever created. It can be used in order to inject shellcode into native Windows applications (currently 32-bit applications only).

Shellter is capable of re-encoding any native 32-bit standalone Windows application. Since we are trying to avoid AV detection, we need to avoid anything that might look suspicious to AV software such as packed applications or applications that have more than one section containing executable code.

Shellter is capable of taking any of these 32-bit Windows applications and embedding shellcode, either your custom payload or one available from such applications as Metasploit, in a way that is very often undetectable by AV software. Since you can use any 32-bit application, you can create almost an infinite number of signatures making it nearly impossible for AV software to detect.

The shellcode can be something yours or something generated through a framework, such as Metasploit. Shellter takes advantage of the original structure of the PE file and doesn't apply any modification such as changing memory access permissions in sections (unless the user wants), adding an extra section with RWE access, and whatever would look dodgy under an AV scan. Shellter uses a unique dynamic approach which is based on the execution flow of the target application, and this is just the tip of the iceberg. Shellter is not just an EPO infector that tries to find a location to insert an instruction to redirect execution to the payload. Unlike any other infector, Shellter's advanced infection engine never transfers the execution flow to a code cave or to an added section in the infected PE file.

Procedure- After firing Shellter in Kali Linux, we need to select from among its various modes: Automatic or Manual (A/M/H). After that, Shellter asks for the location of the PE file. Here we can use a PE file from the list of readymade Windows binary files or use our own executable file. We choose 'vncviewer.exe'. Then it asks you to enable the 'Stealth Mode'. This mode makes the file polymorphic. Then we will ask whether we want to use

a listed payload or custom? This is the framework to be used for controlling the host. We select 'L'. Then it asks to 'select the payload by index'. There are many options listed say:

1. meterpreter reverse TCP
2. meterpreter reverse HTTP
3. meterpreter reverse HTTPS
4. meterpreter bind shell
5. reverse shell TCP
6. bind shell TCP
7. WinExec

We choose '1. meterpreter reverse TCP' to get reverse shell.

Then it will ask for LHOST. Enter the local ip of your system. You can find the local ip by using command 'ifconfig'. Next you need to enter LPORT. Enter anything in lport but the traditional port is 4000 and 4444 but you are free to use any port. Once you are done shellter will do the rest of the job itself.

The file which is created is run through NoDistribute which is a scanner which scans your file through 35 antivirus software and tells as to which one detects malicious content in your file.

Observations & Result- When the infected PE file was scanned through 35 antivirus software, only one antivirus was able to detect it out of 35. This goes on to prove that once we make a polymorphic shellcode, then many antiviruses today do not have the required capability to detect malicious content in the file.

NoDistribute

Scan Results

 File vncviewer.exe	 Size 359.5 KB
 MD5 ca80332eaa27a9a97327b4e78ade9574	 First Scanned 09:08:53 08/18/2017
 Detected By 1/36	
<hr/>	
 A-Squared Clean	 Malwarebytes Anti-Malware Clean
 AVG Free Clean	 McAfee Clean
 Ad-Aware Clean	 NANO Antivirus Malware detected
 AhnLab V3 Internet Security Clean	 Norton Antivirus Clean
 Arcavir Antivirus 2014 Clean	 Outpost Antivirus Pro Clean
 Avast Clean	 Panda Security Clean
 Avira Clean	 Quick Heal Antivirus Clean
 BitDefender Clean	 SUPERAntiSpyware Clean
 BullGuard Clean	 Solo Antivirus Clean
 Clam Antivirus Clean	 Sophos Clean
 Comodo Internet Security Clean	 TrustPort Antivirus Clean
 ESET NOD32 Clean	 Twister Antivirus Clean
 F-PROT Antivirus Clean	 VBA32 Antivirus Clean
 F-Secure Internet Security Clean	 VirIT eXplorer Clean
 G Data Clean	 Zillya! Internet Security Clean
 IKARUS Security Clean	 eScan Antivirus Clean
 Jiangmin Antivirus 2011 Clean	
 K7 Ultimate Clean	
 Kaspersky Antivirus Clean	
 MS Security Essentials Clean	

Figure 3.24 Output from NoDistribute with our created shellcode.

3.7.3 Experiment 3: Getting privileges after infecting a file with polymorphic shellcode.

Purpose: Writing an experiment to infect a PE file and then getting privileges of the attacked system through this file and showing that it remains undetected by many antivirus software.

Hypothesis: Using this experiment we try to prove that it is possible to infect any process executable file with polymorphic shellcode. The infected file may then be used to attack a system and get all privileges of the system. After that many attacks can be performed on the compromised machine.

Explanation: Polymorphic shellcodes are dynamically generated and thus bypass simple signature matching antivirus systems. The dynamism and the latest techniques of polymorphism, helps the malicious software to evade many antivirus systems. Thus it becomes simple for the polymorphic shellcode to attack a compromised machine and get all privileges of the machine. Various attacks can then be performed on the machine like keylogging, camera hacking and obtaining information about system files.

Procedure: Here in this experiment, we will take a Windows binary file called radmin.exe. Since we should not meddle with an actual Windows file, therefore, we copy it in root. After firing Shellter, it asks for the operation mode. We choose Auto. In the PE target, we give '/root/radmin.exe'. We are however not limited to these binaries. Before this we have done 'ifconfig' and we know the IP address of our machine is 192.168.0.7. Shellter will infect this PE file.

It will also ask if you want the 'stealth mode'. This mode makes the file polymorphic. Then we will ask whether we want to use a listed payload or custom? This is the framework to be used for controlling the host. We select 'L'. Then it asks to 'select the payload by index'. There are many options listed say:

1. meterpreter reverse TCP
 2. meterpreter reverse HTTP
 3. meterpreter reverse HTTPS
-

4. meterpreter bind shell
5. reverse shell TCP
6. bind shell TCP
7. WinExec

We choose '1. meterpreter reverse TCP' to get reverse shell. Then it verifies the infected file.

Then we copy 'radmin.exe' using pen drive to windows desktop. Enable the pen drive in Kali linux. Then copy the file in it. Then disable it in Kali. It will automatically get enabled in Windows. Copy the file in Windows Desktop.



Figure 3.25 Copying 'raadmin.exe' file on desktop.

Set up **metasploit** in Kali using the command 'msfconsole'. Set lhost to 192.168.0.7. Set Lport to 1. Same as in the exploit. It can be 4000 or other port number also.

Then fire the exploit. Now we get the reverse shell. If we write 'sysinfo', we get remote system's information. We can get a screenshot or click a photo using webcam. Or write 'getPrivs' to get privileges and probably even shut down the system. Just write 'help' to know all options'. Then exit the msf.

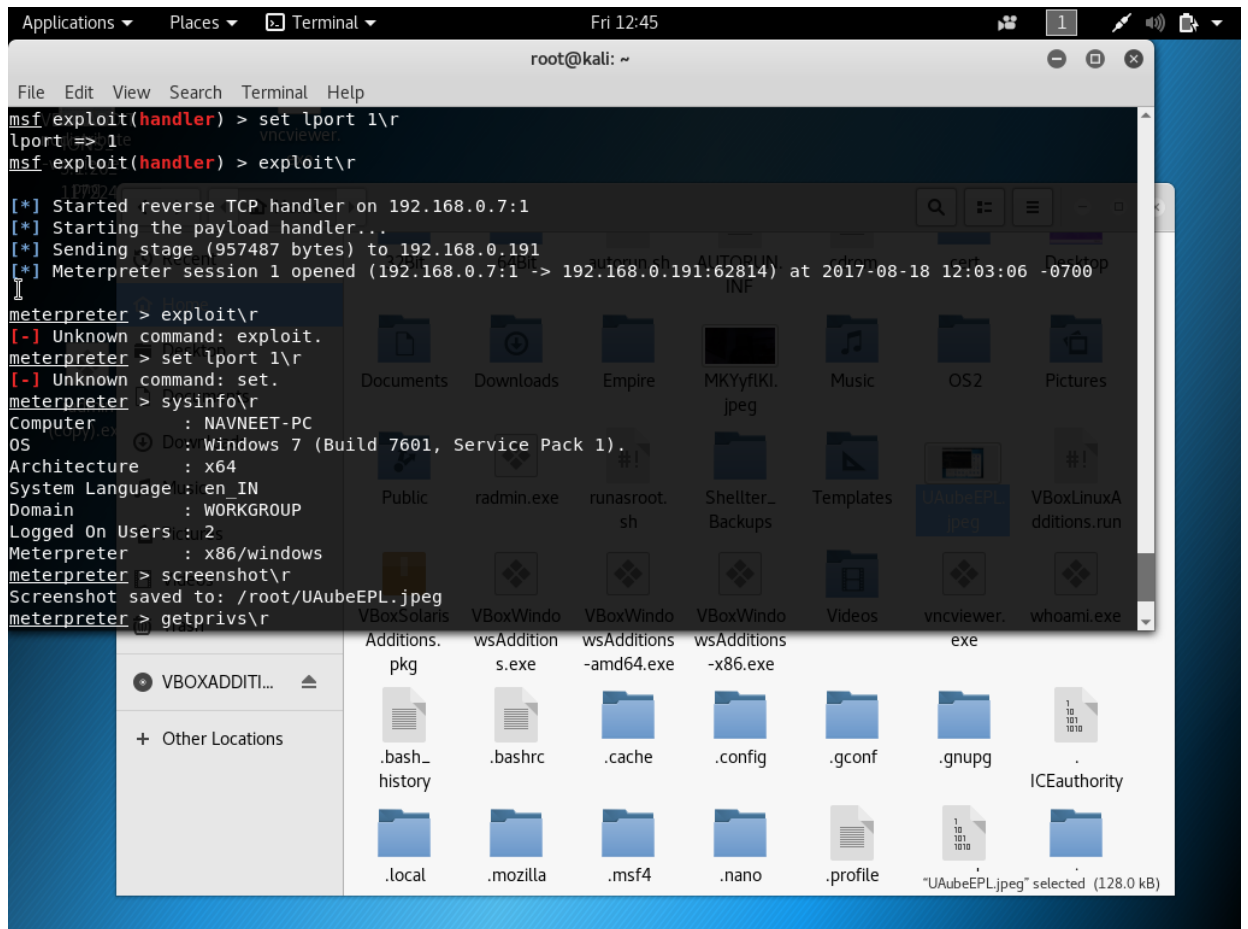



Figure 3.26 Getting system information using infection through shellcode

The file was scanned by 36 antiviruses out of which only 10 were able to detect that it was malicious. Rest all including QuickHeal and Kasperskey were not able to detect it.





Name	radmin.exe	vncviewer.exe
Threat Level	High	Medium
Category	Polymorphic shellcode	Polymorphic shellcode
Propagation Method	Downloaded by users considering it as legit software	Spam emails
Behaviour	<ul style="list-style-type: none"> Degrades system performance significantly. Can cause system to crash or shut down abruptly. Modifies system. Credential stealing. 	<ul style="list-style-type: none"> Consumes system resources thus slowing down the system. Injects its code into all running processes and spreads further. Logs keystrokes.

Figure 3.27 Threat Reports of two advance malwares created during testing



NoDistribute

Scan Results

<p> File radmin.exe</p> <p> MD5 c9cfdc43448980fcd17066bc00b73baa</p> <p> Detected By 10/36</p>	<p> Size 691.5 KB</p> <p> First Scanned 18:08:39 08/21/2017</p>
--	---






























<p> A-Squared Clean</p> <p> AVG Free Win32</p> <p> Ad-Aware Clean</p> <p> AhnLab V3 Internet Security Clean</p> <p> Arcavir Antivirus 2014 Clean</p> <p> Avast Win32</p> <p> Avira Clean</p> <p> BitDefender Clean</p> <p> BullGuard Clean</p> <p> Clam Antivirus Clean</p> <p> Comodo Internet Security Clean</p> <p> ESET NOD32 a variant of Win32/RemoteAdmi...</p> <p> F-PROT Antivirus Clean</p> <p> F-Secure Internet Security Clean</p> <p> G Data Clean</p> <p> IKARUS Security not-a-virus</p> <p> Jiangmin Antivirus 2011 RemoteAdmin.RAdmin.af</p> <p> K7 Ultimate Clean</p> <p> Kaspersky Antivirus Clean</p> <p> MS Security Essentials Clean</p>	<p> Malwarebytes Anti-Malware Clean</p> <p> McAfee RemAdm-RemoteAdmin</p> <p> NANO Antivirus Malware detected</p> <p> Norton Antivirus Remacc.Radmin</p> <p> Outpost Antivirus Pro Clean</p> <p> Panda Security Clean</p> <p> Quick Heal Antivirus Clean</p> <p> SUPERAntiSpyware Clean</p> <p> Solo Antivirus Clean</p> <p> Sophos Clean</p> <p> TrustPort Antivirus RemoteAdmin.CDM(Argon)</p> <p> Twister Antivirus Remotectl.7F200198FBB9227B</p> <p> VBA32 Antivirus Clean</p> <p> VirIT eXplorer Clean</p> <p> Zillya! Internet Security Clean</p> <p> eScan Antivirus Clean</p>
---	---

Figure 3.28 Output of Nodistribute while detecting 'radmin.exe' shellcode

Observation & Results: Using the experiment, it was observed that using a dynamically generated polymorphic shellcode, injected in a PE file, the attacker was able to gain full access to the compromised machine and could do keylogging, getting the camera access or getting all system information and access to important system files of the compromised machine. This infection can easily bypass majority of the common antivirus systems especially those that only employ simple signature matching techniques.

3.8 Significance of Research

Cybercrime and cyber espionage are the biggest threats to businesses today. Not only are big to small organizations getting affected from it but individuals all over the world are also incurring losses. Hacking started as a fun activity and has now developed as a fully funded business. It's used widely by rival organizations to either keep a track of the other organization's business or ruin it partly or completely. Countries are using it for cyber warfare and cyber espionage against other countries.

3.8.1 Need of the Study

It is imperative that a check is put on malicious activities. The biggest threat, cyber world is facing today is that of 'Polymorphic Shellcodes'. These are shellcodes which are polymorphic in nature, meaning that they change their look but have the same behavior. This makes them difficult to detect by Signature based detection systems. Some polymorphic shellcodes are also capable of changing their behavior at runtime. Thus Anomaly based detection systems also fail here.

All the current solutions which are existing today have a some lacunas in common:

1. Signature Matching-Most of them focus on malware detection by signature matching and pattern recognition. Malware authors are now smarter than ever before and signature detection is of no use due to techniques like polymorphism, metamorphism etc.
-

2. No behavioral analysis- Some of them do not take into consideration, the of malware, like file behavior, network behavior and other dynamic behavior of the file to be analyzed.
3. Updating time-Another problem that current antiviruses face is that they take a lot of time to analyze the malware and then update the definition of antiviruses into user's device.
4. Danger to user's privacy- The antivirus companies are sometimes a danger to the user's privacy as they collect data from user on regular basis and use that to make money. The normal user acts only as a data feeder so that these antivirus companies can protect enterprises.
5. Centralized Antivirus Companies-All the antivirus organizations are centralized in nature. This implies that a lot of computation power is required and very few computers are available to provide it. If a distributed system could be designed where all systems in the network contribute to the work of malware detection, things will become faster and more efficient.

Note that the recent ransomware attacks by WannaCry and Petya can prove the above statements, not to mention that none of them were actually polymorphic or metamorphic.

3.8.2 Benefit of the Study

Our proposed model will take care of all kinds of polymorphic shellcodes. It consists of the snapshot technique and the revert-back model. It takes snapshots of the memory and processes and keeps them in the database to understand the working of the malware. Next time when the same malware shows up, we have the list of its behaviors and thus can easily detect it. After that we restore the memory and the processes back to their original state. Machine learning and artificial intelligence are further incorporated to make the work more efficient and detect any new malware also. Decentralized currency was another incredible innovation recently and it led us to the new system of bitcoins. Bitcoins are great but what makes them greater is the technology on which it works. The technology is known as blockchain and blockchain has so many other applications other than just decentralized currency. Here we use the blockchain concept to harness

distributive power of all the systems in the network so that a large amount of computing power is gained with very little cost and everyone in the network is benefitted.

Our work is highly significant both in the present and in the future. Hackers have recently used shellcodes in WannaCry and Petya Malwares. The next step is to use polymorphic shellcode attack. We must be ready for them and our works makes us ready. Also we are proposing a framework of policies for Intrusion Detection and Prevention Systems against such malware to harden them so that they can tackle any kind of threat.

3.9 Designing the Framework (PosDeF)

We are first going to design the framework conceptually using flowcharts. Then we will develop algorithms for it. Finally we would create programs to develop a complete, workable model which can detect any kind of malware with good accuracy.

3.9.1 Design Methodology

This is Applied, Quantitative Research and the research design is Experimental in nature. Various kinds of experiments would be conducted to prove the correctness of our proposed Framework for detection and mitigation of untraced polymorphic shellcodes.

3.9.2 Objectives:

1. Understand the various types of malwares, especially polymorphic shellcodes present in the wild today.
 2. Build a machine learning model for the detection of polymorphic shellcodes.
 3. Train the model using both malicious and legitimate samples.
 4. Do static analysis, behavior analysis, packet analysis and disassembly of the sample in the model.
 5. The machine automatically finds out the relevant features for the detection of polymorphic shellcode.
 6. Test the model with known samples.
 7. Establish a cloud network for distributive analysis in the model.
-

8. Maintain a ledger to keep a track of amount of computation given by each virtual node.
9. Rigorous testing of the model to ensure accuracy.

3.9.3 PosDeF Design

Based on recent innovations we have designed a decentralized system of malware detection in which no central antivirus company is involved and everybody in the network is contributing and is getting benefitted from this system. We use the latest techniques to fight the latest malware, i.e.

1. Machine Learning(Supervised)
2. Deep Learning
3. Distributive Computing
4. Ledger maintenance using Blockchain networks

Machine learning is an upcoming field in computer science[26]. It is an application of artificial intelligence that gives the system, a capability to automatically learn and improve from experience without being explicitly programmed. It is basically a collection of programs meant to learn from examples, also called as 'data set'. Say we want to teach an application to recognize handwritten characters. Now there are two ways of doing it. First is to write down a set of rules for each character specifying all different shapes and styles a character can have. Add to it huge variations in human handwriting. Also there would be separate rules for printed and cursive characters. Writing of such programs would be a humongous task for a programmer and it would also require tremendous computing power.

The second way is to prepare a data set having many different examples of handwritten characters and the computer, by itself learns the rules that best identify a character. This way of learning is called supervised learning. Machine learning is of two types: supervised Learning and Unsupervised Learning.

Supervised Learning: Here labeled training data is provided to the system and then it tries to classify the unknown sample.

Unsupervised Learning: Here we do not provide labeled training data and ask the system to cluster the sample in n number of clusters. Unsupervised learning is quite difficult as it is very hard to analyze how system classifies data as it does not output the parameters on which it clusters the data.

Computer science is changing very fast and according to some statistics, Artificial intelligence has arrived 10 years prior to the expected time. Researchers have shown an incredible success rate of detecting malwares using machine learning although the problem with machine learning is that no matter how much data you train the machine learning algorithm with, it's not sufficient and also it takes huge amount of time to train a large dataset using machine learning [27]. However recent innovations like deep learning and cheaply available gpus have made machine learning really fast but still the time to train the model increases linearly as the size of dataset increases.

3.9.4 Proposed Working of PosDeF

In our approach, we first use supervised learning. We have a machine learning algorithm. It would be given some binary files as a training data set. Although most of our dataset will be contributed by the users of network, we have provided some initial dataset which comprises of malicious data from virus-share database, all .exe files from clean Windows installation and .exe files of popular software from filehippo database as a clean source of data. After getting trained, the algorithm would be given an unknown sample. Through the knowledge it has attained, it would be able to recognize the malicious file. The algorithm can be fine-tuned with better training samples. The whole process can be shown using a flowchart as below.

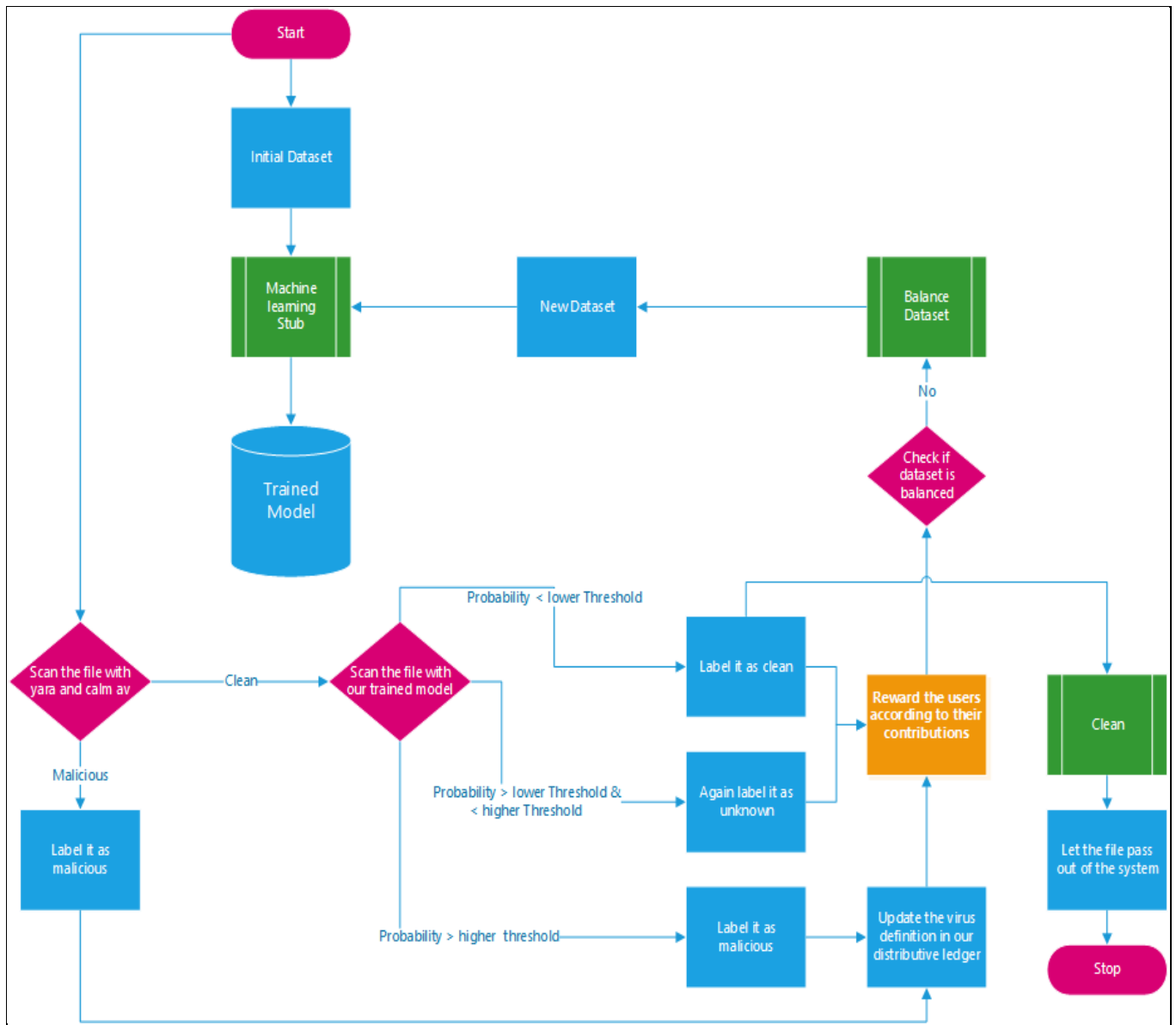


Figure 3.29 Flowchart to demonstrate proposed working of PosDeF.

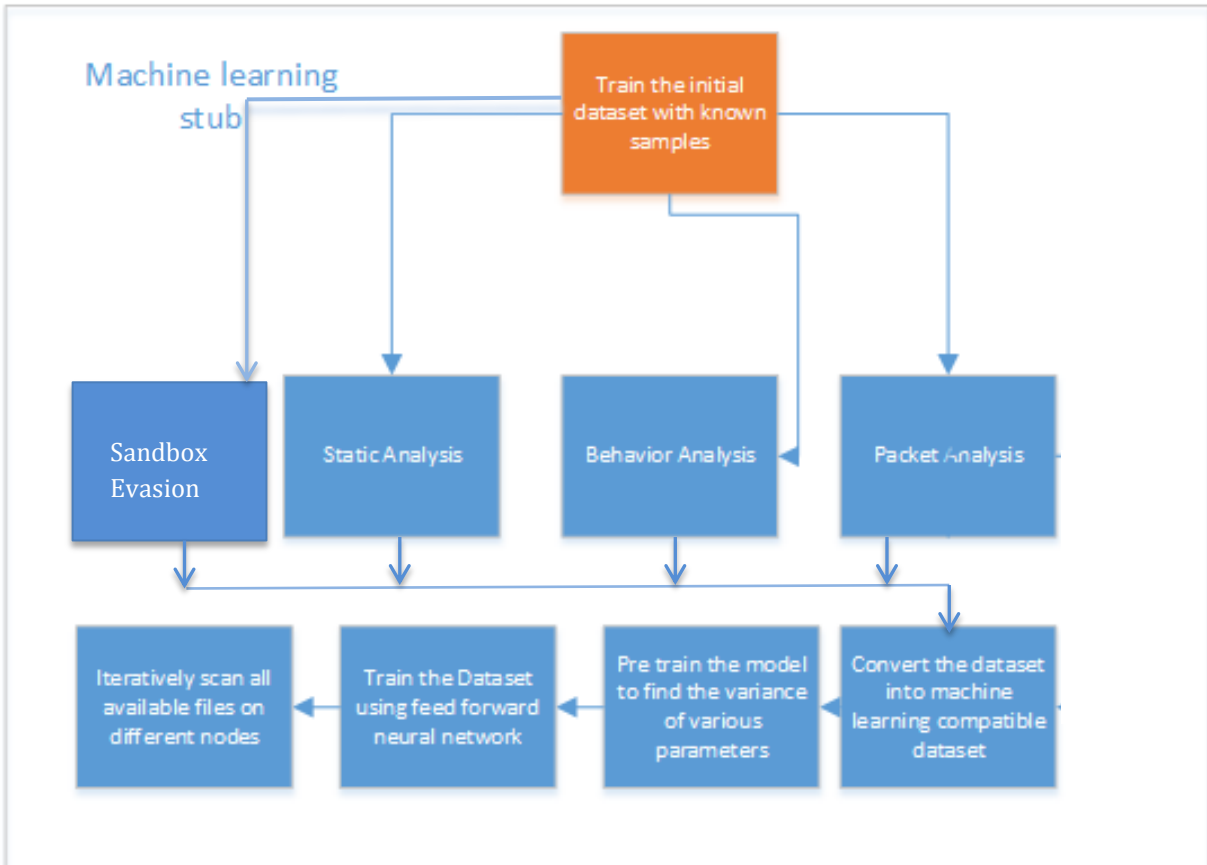


Figure 3.30 Machine Learning Stub

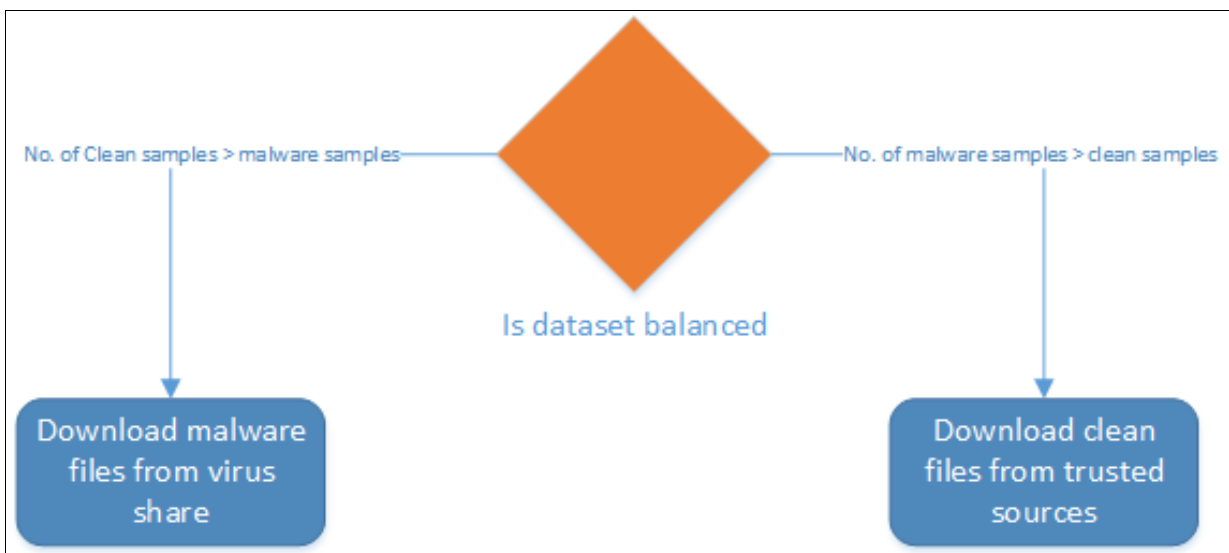


Figure 3.31 Balancing the Training-Testing Dataset

Let's understand the working of PosDeF step-by-step:

3.9.4.1 Collection of Data for the Formation of the Training Set

For creating the training data set, we have to have large number of different files, some malicious and some benign. Having random data set is very important in supervised machine learning algorithm for test model formation. This algorithm is used here. Supervised machine learning adaptively learns from data. It does not need a specific set of rules given by the programmer. It actually learns by examples. We must take care that the data set which is used for training the data must be carefully chosen. It must be an unbiased and random set. If the examples are collected by random sampling only then an accurate model is generated. In case our training data set is biased or partial, the resulting model would also be incorrectly biased. Therefore, for the system to be fair, not only should one have more data samples for fine-tuning of the model, but having the 'right' samples is the most important precedent.

We have taken .exe and .dll files for training. For getting the legitimate files we have collected all binaries from Windows xp, Windows 7 and Windows 2008 which resulted into around 40000 clean files. More clean files can be obtained from trusted sites like FileHippo.com. For getting malicious files, we have used VirusShare.com dataset which contained around 96000 files. Other malicious files can be obtained from Contagio.com, malshare.com, scumware.com etc. Also polymorphic engines like ADMutate, Clet and PhatBot can be used to create polymorphic shellcodes.

In all we have trained our model with around 1.3 lakh files for static analysis and 2000 for dynamic, network and Sandbox evasion analysis. For clean files we have extracted 1000 exe files from clean installation of windows 7. We have used [virusshare_00302.zip](#) file as the collection of malwares we have randomly extracted 1000 files from the above zip.

3.9.4.2 Training with our initial dataset

This step makes our model useful to some extent. We have used the dataset provided from virushare.com containing thousands of malicious samples and all files from a clean Windows 7 installation (around 2500 files). All these files are run through the system and the feature vector which is formed after following the steps in the model is given to the system labeled as malicious or benign. Through this, the model learns that which features distinguish a malicious file from a benign one.

3.9.4.3 Profiling the files

Profiling here refers to extracting information from files which can be useful in malware detection. It has following sub steps:

- **Static analysis** –static analysis here refers to the act of extracting information based on file properties without running it. This is the quickest way to classify the file but not always accurate. We have extracted a total of 52 parameters using a python module called **PE Analyzer 2**.
- **Behavior analysis** –Behavior analysis refers to the act of extracting information at runtime. We have extracted api call graphs, files created and affected at runtime etc. We have used **Cuckoo** for this.
- **Packet analysis** – In this module we are doing the analysis of network traffic of a particular file using **tcpdump** and **snort**.
- **Sandbox Evasion Analysis** – If the sample is trying to evade the sandbox, we consider that this is a sort of malware behavior.

3.9.4.4 Convert the dataset into machine learning compatible format

This step is about converting the raw data into structured data that can be understood by machine learning. We have normalized various parameters using various available techniques like **ngram** [28] and then used **csv** format as our dataset format.

3.9.4.5 Pre train the model

Pre training here refers to finding the variance in parameters so that the training time can be optimized.

3.9.4.6 Train the dataset using various Machine Learning Algorithms

We have tried various algorithms for training of dataset and compared various approaches and according to the data available, we will use a suitable machine learning algorithm.

In all the stages of the model, be it static analysis, dynamic analysis, packet analysis or sandbox evasion, the basic steps remain the same. The underlying idea is shown in the figure below:

- Analysis: We analyze the dataset to find out which features are relevant to our analysis and whether those features are in a usable format.
 - Feature Extraction: If features are not in a compatible format, they are converted into a compatible format.
 - Feature Selection: All relevant features, which are required for the algorithm, are selected and the irrelevant features are dropped.
 - Feature classification: All features are ranked according to their importance for prediction of maliciousness.
-

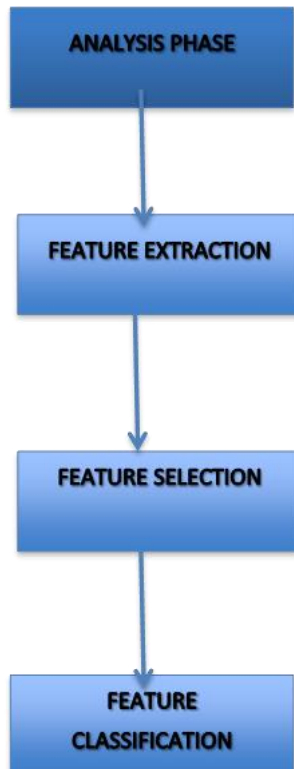


Figure 3.32 The underlying steps in all four analysis parts.

3.9.4.7 Iteratively scan all available files

Since our model is based on distributive properties we will take advantages of our available nodes and constantly improve our model. Yara and Clamav are open source anti malwares available. We will scan the file and if it's detected by existing definition of yara and clamav then label is as malicious. If it's not detected by existing solution then let's give our trained model a try. We will set the malicious threshold value based on our research. For example, if the calculated probability is below 5% then we will mark it as clean and if it is 95% then we will label it as malicious; if it's in the middle of our lower threshold and higher threshold then we will leave it unlabeled as of now. Also if the file is proved to be clean then we will update the ledger of clean files and would not scan the file again but will surely fingerprint it. If the file is proved to be malicious then we will update our distributive ledger with the hash of that file which will act as our always-updated virus definition and will work much faster than current virus definition solutions.

Concept of CVE's: CVE stands for Common Vulnerabilities and Exposures. It is a publicly available database which provides a reference method for a list of public security threats. Most of the antimalware companies get their data from this list. We also use these CVE in the form of analysis by open anti malware systems like ClamAV and Yara. After analyzing our dataset, if we are able to find a new security threat, we update the CVE so that everyone knows of this threat almost immediately at the same time. We also have a reward system described below involving CVE's.

3.9.4.8 Reward the user

Since in our model there is no central authority for which we have to pay our nodes. Users are everything in our network. They are the data providers and they are the ones who will benefit from model. So they must be the ones who should get rewarded for their contribution of bandwidth and computation power. The rewards can be in the form of points. These points can be in the form of CSR points for a company which they can use in any way that they want. Individuals can be awarded with getting their names in CVE files indicating that they helped finding a particular vulnerability and removing it. Later on the points can be used for giving any form of reward and recognition.

3.9.4.9 Balancing the dataset

There can be unbalancing issues with our dataset which can later be turned into undesirable results there can be two cases:

More clean files than malicious – In this case the solution is simple. We will block the extra clean files to go into the training phase and delay them for the next time (whenever required).

More malicious files than clean – We can perform the same step as above but this time we can do it in a better way. In our model, we are assuming that all

executable files hosted on filehippo are clean so we can download some .exe files from filehippo and label them as clean which will further increase the accuracy of our model.

3.9.4.10 Endless loop

The whole process explained in our flowchart is continuous and will never end as when we will get more and more nodes, more and more files will be there and more and more computation power will be there so our model keeps on improving until it becomes better than existing solutions and there is no end to it.

3.10 Algorithms used for Building PosDeF

Here we present the various algorithms that we have created for building of the framework. We start by giving the final algorithm followed by the stub algorithms of static, dynamic, network analysis and sandbox evasion.

3.10.1 The Final Algorithm

The final framework is created using an algorithm which takes the various parts like Static analysis, Behavior analysis, Packet analysis and Sandbox evasion as stubs. The final algorithm predicts the possibility of maliciousness of the sample by calculating the average of probabilities predicted by all the stubs.

Before this, a threshold value of maliciousness is calculated according to the data set that we currently have for training the framework. If our combined predicted value is lower than the threshold value then the sample is considered clean. If it is above the threshold value, the sample is considered malicious. However, if it is equal to the threshold value, the sample is considered unknown and sent for detection again.

3.10.1.1 Algorithm for Training of PosDeF

Let us assume that **D** is a multifactor set

$$\mathbf{D}=\{\mathbf{a} \cup \mathbf{b} \cup \mathbf{d} \cup \mathbf{e}\}$$

Where:

a→selected features of static analysis

b→selected features of dynamic (behavior) analysis

c→selected features of Snort analysis

e→sandbox evasion (Boolean 0 or 1)

We calculate four probabilities of maliciousness for a sample

Pa= Static Probability of maliciousness

Pb= Behavior Probability of maliciousness

Pc= Snort Probability of maliciousness

Pe= Sandbox Evasion Probability of maliciousness

Cp=Combined Probability

Tc= clean.threshold.percentage

Tm=malicious.threshold.percentage

Then we calculate **combined probability** of maliciousness for the sample by calculating average

$$\mathbf{Cp}=(\mathbf{Pa}+\mathbf{Pb}+\mathbf{Pc}+\mathbf{Pe})/4$$

R= Result after testing

N=Total number of samples

Nc= Number of clean samples

Nm= Number of malicious samples

Now, let's see when a sample is passed from our model-

Threshold calculations

4

$$T_c = \sum_{i=1}^4 P_i \{ R \neq \text{Malicious} \} / n$$

4

$$T_m = \sum_{i=1}^4 P_i \{ R \neq \text{Clean} \} / n$$

3.10.1.2 Algorithm for Threshold Calculation for PosDeF during the training phase.

Say Sample[] is our array for training the model. It has both clean and malicious files.

FindThresholdAlgorithm

{

Nm → 0;

Nc → 0;

//Initially we set number of clean and malicious samples to 0

Tc → 0;

Tm → 0;

//Initially set the threshold of both clean and malicious samples to 0

for (i=0 ; i<=Sample.length ; i++)

{

 Sample[i].Pa=Calculate.Static.Probability (Sample[i]);

//Calculate.Static() is a function which calculates static maliciousness probability for Sample[i]

Sample[i].Pb=Calculate.Behavior.Probability (Sample[i]);

//Calculate.Behavior() is a function which calculates Behavior maliciousness probability for Sample[i]

Sample[i].Pc=Calculate.Snort.Probability (Sample[i]);

//Calculate.Snort() is a function which calculates Snort i.e. network maliciousness probability for Sample[i]

Sample[i].Pe=Calculate.SandboxEvasion.Probability (Sample[i]);

//Calculate.SandboxEvasion() is a function which calculates whether the Sample[i] tries to evade the sandbox or not. This value is Boolean 0 or 1

Sample[i].Cp=

(Sample[i].Pa + Sample[i].Pb + Sample[i].Pc + Sample[i].Pe) / 4 ;

If (Sample[i].result == clean)

{
 Tc=Tc+ Sample[i].Cp ;

 Nc ++ ;

}

else

{
 Tm=Tm + Sample[i].Cp;

 Nm ++ ;

}

}

clean.threshold.percentage= $T_c/N_c * 100$;

malicious.threshold.percentage= $((T_m/N_m) - \text{clean.threshold.percentage}) * 100$;

unknown.threshold.percentage= $\text{more_than}(\text{clean.threshold.percentage})$ and
 $\text{less_than}(\text{malicious.threshold.percentage})$;
}

3.10.1.3 Algorithm for Testing of PosDeF

TestAlgorithm(Sample)

{

Pa= Calculate.Static.Probability(Sample)

Pb= Calculate.Behavior.Probability(Sample)

Pc= Calculate.Snort.Probability(Sample)

Pe= Calculate Sandbox Evasion.Probability(Sample)

CP= $(P_a+P_b+P_c+P_d)/4$

Md5Sum= Find Md5Sum(Sample)

If (Cp>ThClean)

{

//means file is malicious

Log(Md5Sum) in clean sample database

Start_BlockChain_Transaction(Md5Sum)

}

Else

{

//means file is clean

```
Log(Md5Sum) in malicious sample database
Start_BlockChain_Transaction(Md5Sum)
}
}
```

3.10.2 Static Analysis

The data for training is in the form of .csv file which we have created using samples by extracting a total of 52 parameters using a python module called **PE Analyzer 2**

This data has to be stored in a 'Panda' frame. Panda is a python package which provides fast and flexible data structure for analysis in python. The two important data structures for Panda are 'Series'(1-Dimensional) and 'Data Frame'(2-Dimensional).

3.10.2.1 Algorithm for Static Training

Extra Tree Classifier is used for feature selection here. Extra Tree stands for Extremely randomized trees. Extra Trees are computationally faster than other methods of feature selection. They select a cut point in the tree at random. Thus they reduce the computation burden of determining the cut point and leads to increased accuracy because of smoothening. Cut point is the point of best split which separates the samples of a node into two groups. The cut point randomization leads to a good variance reduction effect and gives great results in many high-dimensional complex problems. Therefore, in this case, Extra Tree classifier is used for static training.

```
Static_Training (data)
```

```

{
//Read data in Panda Frame for further processing

X=Pandas.read(data);
Y=data(labels);

/* The labels are Boolean values of 1 or 0. Since this is training data we know which file has
a clean and which a malicious label */

//Pretraining phase for shortlisting of features

Feature_Selection=ExtraTreeClassifier.fit(X,Y);
X_new=Feature_Selection(X);

/*Now we apply Cross Validation technique by splitting the data into 80% training and 20%
testing with the data with selected features along with their labels*/

Cross.Validation.split(X_new,Y,0.2);

}

```

3.10.2.2 Algorithm for finding out the best classification algorithm for Static Testing

We have the types of classification algorithms in Machine Learning like Linear Classifiers: Logistic Regression, Naive Bayes Classifier, Support Vector Machines, Decision Trees, Boosted Trees, Random Forest, Neural Networks, Nearest Neighbour etc. We have to check all algorithms and find out the score as to which algorithm works best with our data. The winner algorithm is then applied for the testing algorithm.

BestAlgoStatic()

```

{
algo[]={a0,a1,a2,a3.....an}; // We put all the available algorithms in an array
int i;
float score[10];
for(i=0;i<n;i++)
    {
        score[i]=ai.fit(X.test,Y.test);
//We try to find out the score with which an algorithm best fits a model
    }
j=max(score[]);
WinAlgo=aj; //Algorithm with the maximum score is chosen for static testing
}

```

3.10.2.3 Algorithm for Static Testing

P Calculate.Static.Probability(Sample)

```

{
/*Extract features from the Sample file using python's PE Analyser. Only those features are
used which we have selected during the training phase */

Features[]={f1,f2,f3,.....,fi};
Result=aj.predict.probability(Features[]);
return Result;
}

```

3.10.3 Behavior Analysis

Behaviour analysis or Dynamic analysis, tries to see the behaviour of the sample, rather than its signature and matches the behaviour with known malware behaviour patterns. In this way, even if the sample evades signature detection, it can get caught in behaviour analysis.

3.10.3.1 Algorithm for Behaviour Training

For behaviour training, we submit all our training data samples to Cuckoo. JSON reports are generated for these samples. All these JSON reports are converted into MIST reports. These MIST reports are then converted into an N-Gram sparse matrix using TF-IDF vectorizer object of sklearn.

TF-IDF stands for 'term frequency-inverse document frequency'. This means that the weight assigned to each token not only depends on its frequency in the document but also how recurrent that term is in the whole corpora. Actually in a big file, some words like "the", "a", "is" etc. are very frequently present. Therefore, they give very less meaningful information about the actual contents of the document. If we give the direct count data to the classifier, the very frequent terms will shadow the frequencies of the rarer but more interesting terms. Through TF-IDF, we reweigh the count features to make them more meaningful.

```
Behaviour_Training(data[])
{
reports[];
mist_reports[];
int i=0;
while(data.length)
{
    reports.append(Cuckoo.submit(data[i]); //JSON reports generated
    mist_reports.append(Cuckoo_to_mist(reports[i]));
    //MIST reports generated
    i++;
}
X=Tfidf.vectorizer.transform(mist_reports[]);
// MIST reports getting transformed into NGram sparse matrices
Y=data[].lables; //Boolean 0 or 1
Cross.Validation.split(X,Y,0.4);
//60% data is used for training and 40% for testing
}
```

3.10.3.2 Finding Best Classification Algorithm for Behaviour Testing

```

BestAlgoBehaviour()
{
algo[]={a0,a1,a2,a3.....an}; // We put all the available algorithms in an array
int i;
float score[10];
for(i=0;i<n;i++)
    {
        score[i]=ai.fit(X.test,Y.test);
//We try to find out the score with which an algorithm best fits a model
    }
j=max(score[]);
WinAlgo=aj; //Algorithm with the maximum score is chosen for static testing }

```

3.10.3.3 Algorithm for Behaviour Testing

```

P Calculate.Behavior.Probability(Sample)
{
report=Cuckoo.submit(Sample);
mist_report=Cuckoo_to_mist(report);
transformed_mat=vectorizer.transform(mist.report);
Result=aj.predict.probability(transformed_mat);
return Result;
}

```

3.10.4. Snort Analysis

‘Snort’ analysis is done to obtain network characteristics of the file. Cuckoo creates a dump.pcap file. Snort generates a text report based on this file predicting alerts or malicious content in the sample. This text file is again converted into a sparse matrix of N grams using ‘vectorizer’ object.

3.10.4.1. Algorithm for Snort Training

```
Snort_Training (data)
{
pcap_reports[];
snort_reports[];
int i=0;
while(data.length)
{
    pcap_reports.append(Cuckoo.submit(data[i]);
    //dump.pcap files are generated by Cuckoo

    snort_reports.append(pcap_to_snort(pcap_reports[i]));
    //Snort reports generated from pcap files

    i++;
}
X=vectorizer.transform(snort_reports[]);
// Snort reports getting transformed into NGram sparse matrices

Y=data[].lables; //Boolean 0 or 1
Cross.Validation.split(X,Y,0.4);
//60% data is used for training and 40% for testing

}
```

3.10.4.2 Finding out the best algorithm for Snort Testing

```
BestAlgoSnort()
{
algo[]={a0,a1,a2,a3.....an}; // We put all the available algorithms in an array
int i;
float score[10];
for(i=0;i<n;i++)
    {
        score[i]=ai.fit(X.test,Y.test);
//We try to find out the score with which an algorithm best fits a model
    }
j=max(score[]);
WinAlgo=aj; //Algorithm with the maximum score is chosen for static testing
}
```

3.10.4.3. Algorithm for Snort Testing

```
P Calculate.Snort.Probability(Sample)
{
Pcap.file.Cuckoo.submit(Sample);
Snort.report=Snort(Pcap.file);
transformed_mat=vectorizer.transform(Snort.report);
Result=aj.predict.probability(transformed_mat);
return Result;
}
```

3.10.5. Algorithm for Sandbox Evasion

```
Boolean Find_Sandbox_Evasion(Sample)
{
Report=Cuckoo.submit(Sample);
//here we parse the JSON report to find out whether the sandbox is evaded or not

return parse.sandbox(report);
}
```

CHAPTER 4: ANALYSIS AND INTERPRETATION

4.1 Creation of Polymorphic and Metamorphic Shellcodes.....	102
4.1.1 Obfuscation of NOP sled.....	102
4.1.2 Obfuscation of the shellcode.....	105
4.2 Using the Polymorphic Shellcode for Attack.....	108
4.2.1 Launching a multi-staged attack.....	108
4.2.2 Sandbox evasion techniques.....	111
4.2.3 Polymorphic blending.....	111
4.2.4 Conversion to metamorphic code.....	112
4.3 Working with Live Ransomware Samples- WannaCry and Petya.....	112
4.3.1 Recent Impact of Ransomwares.....	112
4.3.1.1 WannaCry.....	113
4.3.1.2 Petya.....	113
4.3.2 Reasons of Attack of Ransomware.....	114
4.3.3 Behavior Analysis of WannaCry and Petya.....	115
4.3.4 Working of WannaCry.....	119
4.3.4.1 Components of WannaCry.....	122
4.3.4.1.1 DoublePulsar.....	122
4.3.4.1.2 TOR.....	123
4.3.4.1.3 Domain Check.....	123
4.3.4.1.4 Bitcoin Wallets.....	124
4.3.5 Precautions to be safe from ransomware.....	124
4.3.6 Remedy after ransomware attack.....	125
4.4 Building the Framework.....	125
4.4.1 Building the Static Part of the Framework.....	125
4.4.1.1 Feature Extraction.....	126
4.4.1.2 Feature Selection.....	128
4.4.1.3 Feature Classification.....	129
4.4.2 Building the Behavior Part of the Framework.....	132
4.4.2.1 File Analyzed in Cuckoo Sandbox.....	133
4.4.2.2 JSON Reports Generated.....	134
4.4.2.3 JSON Reports converted into MIST format.....	134
4.4.2.4 MIST reports converted into N-Gram data.....	136
4.4.2.5 Sparse matrix Generated.....	137
4.4.2.6 KNN Classification Algorithm applied.....	138
4.4.3 Building Network Part of the Framework.....	139
4.4.3.1 Cuckoo generates dump.pcap file.....	140
4.4.3.2 Snort generates text file out of pcap file.....	140
4.4.3.3 Text file converted into MIST format.....	141
4.4.3.4 MIST file converted into sparse matrix.....	141
4.4.3.5 KNN applied to sparse matrix.....	141
4.4.4 Building Sandbox Evasion Detection part of the Framework.....	142

4.5 Distributive Execution of the Framework.....	143
4.5.1 Introduction.....	143
4.5.2 Proposed Distributive Framework.....	145
4.5.3 Centralized versus Distributed Computing.....	147
4.5.4 Role of Apache Spark in the Framework.....	148
4.5.4.1 AWS M3.....	149
4.5.4.2 M3 medium.....	149
4.5.4.3 Amazon EC2.....	149
4.5.4.4 SSD.....	149
4.5.4.5 HDFS.....	150
4.5.4.6 HDFS Architecture.....	150
4.5.4.7 Non-Computational Data Locality.....	150
4.5.4.8 Data and Processing on each machine.....	150
4.5.5 Working of the Hadoop Cluster.....	152

CHAPTER 4

ANALYSIS AND INTERPRETATION

4.1 Creation of Polymorphic and Metamorphic Shellcodes

As shown in Figure 1, the creation of shellcode is broadly a two-step process in which first a host program is created which has some vulnerability like buffer overflow. This is the code in which the shellcode is injected. These two programs together become the payload which attacks the target computer. Both these codes have to be obfuscated so become undetectable.

Steps in the creation of polymorphic and metamorphic shellcodes:

- A. Obfuscation of NOP sled
- B. Obfuscation of the shellcode

4.1.1 Obfuscation of NOP Sled

Normally the buffer overflow attack follows very simple steps. First the attacker finds a vulnerable program, which is most often a program with a buffer (an array) whose bounds are not being checked by the program. In a typical memory layout of this program, the used buffer will be followed by the EIP (Extended Instruction Pointer), which is the address of the program counter or the return address that the execution would jump to when the current function has finished. The attacker would overflow this buffer with a crafted input so that it overwrites the buffer and the EIP and instead points to the shellcode which the attacker wants to execute. This is a precise address where the shellcode is located. Now, the operating system designers are well aware of the buffer overflow attacks. Therefore, to get over these attacks, they have devised a process called 'Address Space Layout Randomization'. In this process, the address spaces including stacks, heaps and other memory structures are randomly offset. This makes guessing of

the accurate location of the shellcode all the more difficult. To get over this problem, the attackers increase the address space using NOPs. NOP stands for No operation. These are one byte instructions[29] which actually perform no operation but take space in the instruction stream. It does not affect any programmer-accessible registers, flags or memory. It only affects the EIP and just takes the control flow to the next instruction without affecting the program in any way. A series of consecutive NOP instructions is called a NOP sled. The NOP sled is appended before the shellcode which the attacker wants to execute. This ensures that the CPU jumps somewhere in the sled to ultimately reach the shellcode to start the attack.

This boon of NOP sleds can easily become a bane for the attacker. The moment a series of NOP instructions are visible to the Antivirus or the IDS, they will block the shellcode from executing. Here we come to a technique called obfuscation where we replace the NOPs with some other equivalent instructions to bypass the AVs and the IDS.

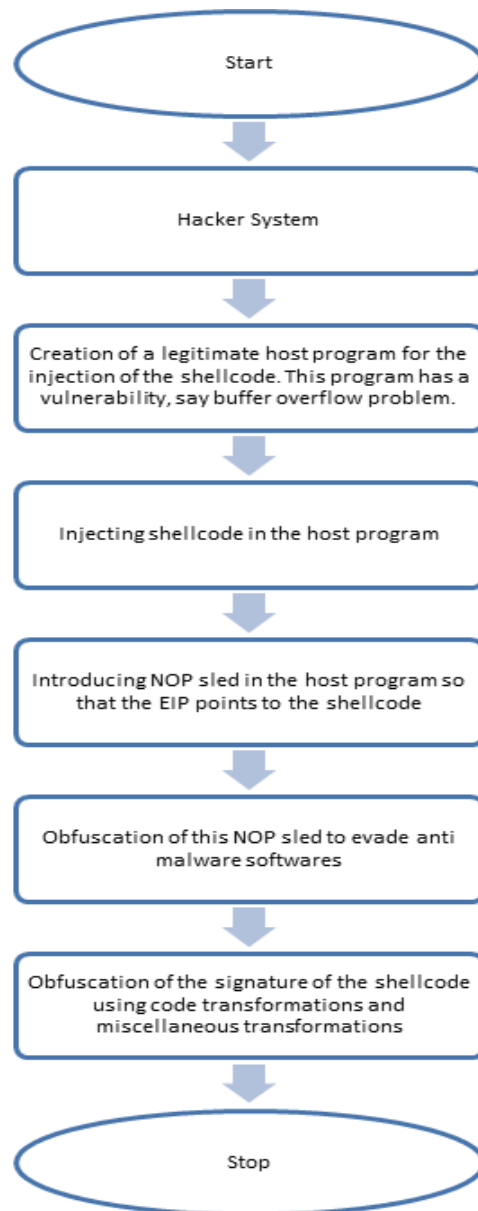


Figure 4.1 Creation and injection of malicious payload for attacking the target system.

There are multiple ways in which NOP sleds can be obfuscated[30]:

- *Single byte NOP equivalent instructions:* single byte NOPs like 0x90 can easily be detected. However, if these 0x90 can be replaced by other equivalent instructions, then the chance of detection can be reduced. There are many single byte instructions which can act as a substitute for 0x90. Here is a list of instructions which can be used:

ASCII character	Instruction	ASCII character	Instruction
A	Inc ecx	W	Push edi
B	Inc edx	X	Pop eax
C	Inc ebx	Y	Pop eax
D	Inc esp	Z	Pop edx
E	Inc ebp	A	Popa
F	Inc esi	B	Bound
G	Inc edi	C	Arpl
H	Dec eax	D	seg=fs

Table 4.1 Single Byte NOP Equivalent instructions

All the uppercase letters can be used to obfuscate the NOP sled. For example, the sentence-” GET THE REGISTER VALUE”, is a valid NOP equivalent sled. K2 states[31] that there are at least 55 suitable replacements for NOP instructions for the Intel architecture. Sometimes, while deciding which equivalent NOP instruction to use, one may consider a number of factors. For example, using capital letters may be dangerous because the first group of capital letters increment or decrement registers[32]. This may affect the shellcode. The next group of capital letters are push and pop instructions which can play havoc with the stack. However, letters ‘abcdefghijklmnop’ don’t map to anything. After that mapping to opcodes begins again. These can easily be used without crashing the system. Also using alphabets can be inefficient if the service is not an alphanumeric service.

- **Multibyte NOP equivalent sleds:** The problem with one-byte fake NOPs is that not many of them are available. So it becomes easy for IDS to detect a fake NOP zone. Therefore to evade detection, the one byte NOPs can also be replaced by

multibyte NOP instructions. The only requirement is that every instruction must be executable at the offset and must take the program counter to the address of the exploit.

- **Four byte NOP aligned sled**-The restriction of multibyte NOP equivalent sleds is that every instruction must be executable. However, one can create an easier sled which needs to be executable after every 4 bytes i.e. after a word. Here only the last instruction needs to advance the program counter.
- **Trampoline sled**- Trampolines are memory locations where whenever execution into them, it bounces out immediately. This technique is used by the attacker to jump quickly to the shellcode address[31]. Actually whenever a system gets initialized, every process loads the contents of the external libraries in their address space. This address space is a reserved section of the memory. Thus these addresses are predictable. Now say the attacker uses an instruction 'jmp ESP' and knows the address of the register pointed by ESP. Now the attacker places his exploit in this register. Then he overwrites the return address by the address pointed by ESP. thus on returning, the application will jump and execute 'jmp ESP' instruction and the exploit at that location would be executed.

4.1.2 Obfuscation of the shellcode

There are various ways in which the shellcode can be obfuscated[33]. These are 'Code transformations':

- **Dead code insertion**-means that you change the signature by adding garbage instructions in the code which does not affect the code functionality. Eg.

a=h; b=23; f=m+n

Can be changed to

a=h; a++; a++; b=23; a=a-2; f=m+n

- **Subroutine and Instruction reordering**- here order of the subroutines is changed so that they are called in a random order. Also the order of the instructions may be
-

changed as long as the interdependencies among instructions is retained.

Instruction reordering example:

a= q+m; b=c-b; d=a*b

Can be changed to

b=c-b; a=q+m; d=a*b

- **Code transposition**-Code transposition reorders the flow of the instructions keeping the result of the program same. It can be achieved using either conditional or unconditional branches. This technique was used by the Zperm virus.

- **Instruction substitution**- means changing instructions in the code with equivalent instructions. Eg. `ADD reg,imm=>SUB reg,(-imm)`

`MOV reg1,reg2=> PUSH reg2 POP reg1`

`x=1=>y=21; x=y-20`

Similarly there are four instructions which would set the register r2 to zero

`CLEAR r2; MOVE 0,r2; AND 0,r2; XOR r2,r2`

- **Code integration**- in this technique, the virus first fragments the code of the program in which it has to get inserted, into small fragments. Then it inserts itself in between these fragments and then compiles the code to create a new code. This technique is used by Zmist virus. It de-compile the PE file into fragments 32 MB long. Some code blocks are removed and Zmist inserts itself into the code. Then it creates the code again and compiles it into a new executable file. This virus then becomes very difficult to detect.
- **Register reassignment**-basically means that you replace one register to the other while keeping the functionality of the code same. The binary sequence of the code changes and makes signature detection difficult. As an example, edx can be replaced by eax, edi can be replaced with ebx and esi can be replaced with edx.

In addition to code obfuscation, for malwares, additional techniques are to be employed.

These are called 'Miscellaneous Transformations':

- ***Entry point obfuscation***- Whenever a virus infects a PE file, it must gain control to start its execution. Normally the easiest way to do this is to change the entry point of the PE file to the virus body. But this method is the best detectable also. Therefore entry point obfuscation changes the entry point to the middle of the PE file and the control is obtained using jump or call instructions.
 - ***Information exfiltration obfuscation***- Different kinds of data can be filtered from the compromised machine to the c&C server. All this web traffic may consist of screenshots, key log details, user id, passwords, email messages, recorded conversations, clicked photographs etc. All this data is hidden using encryption and then sent across.
 - ***Obfuscation of communication to the c&c server*** - Communication with the c&c server may use covert channels like TCP, ICMP and IPv6 tunnels. Also, compromised servers may be used as c&c servers without the knowledge of the administrator. Additionally, TOR can be used for all communications to make them untraceable.
 - ***Avoiding a big return address and hiding it***-normally the return address in a shellcode is a huge zone. The return address comes after the payload after overwriting the original return address and is repeated several times. Also it cannot be encrypted. If the detection methods find this large area, they may get suspicious. The idea is to avoid such a huge return address and hide it from the detecting eyes. Return address polymorphism [34]can be achieved by mutating the lower order bits in the address in each generation so that all these addresses point to somewhere in the NOP zone.
 - ***Self-modification of code***- in many cases, a highly effective obfuscation technique is self-modification of code. Here, the code alters itself during runtime. What it essentially means is that the code can modify itself when it runs and it may not be the same code that was there when it first loaded. Self-modification involves dynamic code generation and subroutine patch management. This is an effective way of bypassing sandboxes. New code can be generated at runtime or even existing code can be modified.
-

One way this can be done in Java rootkits is by modifying the JVM. One first locates the class which is to be modified. Then you extract it and disassemble it. Then you modify the bytecode and assemble the code back again. Then you can deploy the file back to its original location. Additionally, since all classes are children to the Object class, therefore the Object class can be modified to affect all child classes thereafter, say by adding a function like `public void keyLogEventHandler(Event e)` which will act as a key logger for all the classes.

4.2 Using the Polymorphic Shellcode for Attack

As shown in Figure 2, the payload has to enter the target system after getting all information about the system and the connected network. After entering the system, the malware has to perform three functions: first it has to send vital information about the system and the network back to the attacker, it has to infect the connected computers and it has to employ evasion techniques to bypass constant monitoring by the anti-malware systems.

Steps of attack are:

- A. Launching a multi-staged attack
- B. Sandbox evasion techniques
- C. Polymorphic blending
- D. Conversion to metamorphic code

4.2.1 Launching a Multi-staged attack

There are various steps in launching a multi-staged attack:

1. Information gathering
 2. Entering the target system
 3. Privilege escalation
 4. Establishing a connection back to the attacker
 5. Injecting a c&c component into the target
 6. Going deeper into the network
-

7. Cleaning up the mess

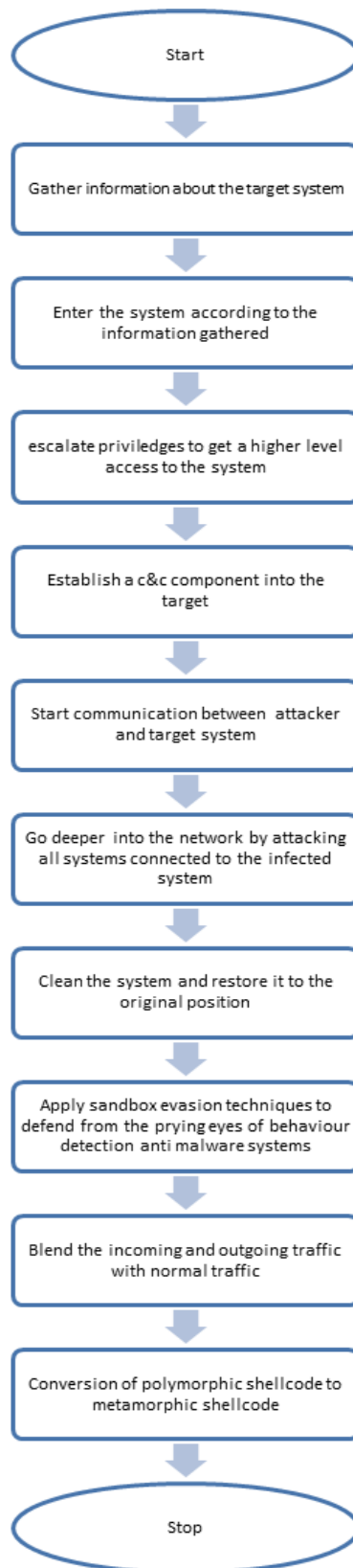


Figure 4.2. Process of Attacking the Target System with the created payload

- **Information gathering-** In this step, the attacker uses various techniques to gather information about the target machine and its network configuration[35]. Social engineering and Google hacking are the most commonly employed techniques. In addition to that, information about the network is obtained using mechanisms like ARP, TCP SYN packets, ICMP echo requests, TCP connect and passive discovery. Port scanning is another useful technique. ‘WannaCry’, for example was a ransomware which hit the world in May 2017. It entered the target systems through open SMB ports. To know the operating system of the target machine, one can use nmap if packets can be sent to it. The command ‘nmap -o’ provides OS fingerprinting whereas if one can eavesdrop network traffic one can use ‘pof’ for passive fingerprinting.
 - **Entering the target system-** After gathering information about the target machine, it is now time to attack and penetrate it in the best possible manner. Normally an exploit is injected into the system and then it starts collecting local information about the compromised machine.
 - **Privilege escalation-** getting the admin rights makes the attacker ‘God’ of the compromised system. Thus the next step is to get root privileges of the machine.
 - **Establishing a connection back to the attacker-** The information collected and the rights obtained are of no use until the attacker can employ them. So the malware now establishes a connection back to the attacker which can further be used for much other exploitation, not just on the compromised machine but also on other machines of the connected network.
 - **Injecting a c&c component into the target-**This command and control component enables the attacker to execute many other commands on the compromised machine.
 - **Going deeper into the network-** The attacker may not stop at this local machine but would want its malware to spread deeper into the connected network.
-

- ***Cleaning up the mess***- To continue the stealth work, the system must not be left in an unstable state and all memory must be repaired before proceeding further. Thus it becomes imperative to clear one's tracks and make the system 'look' like before.

4.2.2 Sandbox Evasion Techniques

A very powerful technique developed by antivirus and IDPS creators is sandbox technique. Using this technique, the suspicious programs can be run in a simulated environment called a sandbox. All programs, whether malicious or non-malicious would show their true nature in this environment. Thus the anti-malware program would be easily able to detect the malicious programs and weed them out of the system. However, attackers have come up with sandbox evasion techniques. They employ various methods to detect whether they are working in a simulated environment or the actual operating system. Accordingly, if they detect a sandbox, they would lie dormant or do very superfluous jobs. It's only when they are sure they are running in the actual operating system, would they show their true working and would launch an appropriate attack.

4.2.3 Polymorphic Blending Techniques

Anomaly based IDPS', especially those that are byte frequency based, find out about the presence of malware in the network stream by checking the payload behaviour anomalies of the malware. The indicators they track are input and output bytes, number of connections, number of packets, type of protocols used and the communication destination. As an example, data exfiltration can be a powerful network statistic to detect a malware. Say a system is baselines in the sense that it transfers data only to the internal systems. If a large amount of data starts going out of the system to an external server on the Internet, this change in behaviour can raise an alarm and alert the detection system. Also, increase in the packet frequency may indicate a denial of service attack. For detecting host level attacks, we detect protocols in the packets to see if they have ambiguous options, are too small or violate some application layer protocols. Attackers, can evade anomaly based detection systems by blending their traffic with normal traffic. They prepare the payloads and the behaviours of their packets in such a manner that the packets are indistinguishable from normal traffic. The message HTTP headers can be

spoofed to merge with legitimate traffic. For example instead of the name of a c&c server, after connection, the host name can be given as google.com. Also TOR traffic can be disguised to look like a normal HTTPS traffic. Such blending techniques can be used to easily blend malware with normal traffic and evade anomaly based IDPS.

4.2.4 Conversion to Metamorphic code

Both polymorphic and metamorphic malwares can change their codes in each iteration as they propagate. Although the code changes but its ability and functionality remains the same. Even while the malware is in the system, it can change its code periodically and this makes it difficult to detect by antivirus and IPS systems[36]. Polymorphic malwares consist of two parts: VDR(Virus Decryption Routine) and EVB(Encrypted Virus Body). When a malware is hidden in a legitimate code, it is in its encrypted form to remain hidden from the AVs. To launch the attack, the VDR decrypts the encrypted virus body back to its original form so that the virus can perform its actions. Now the virus decryption routine VDR remains constant each time, though the key may keep changing. This static part of the code makes it possible for an antivirus program to identify the malware. Another problem is that when the decryption is over, the original code remains naked in the memory. So although it is very complex for the cracker to understand and debug polymorphic code which may have taken months to be written, he just has to wait a few hours till a decrypted , clean and comprehensive code is visible to him.To get over this problem, the attacker can divide the code into smaller parts and put each part into its own polymorphic envelop. The cracker would never see the complete code at once but still parts of original code would still be visible to him later. Thus comes metamorphism. The principle difference between polymorphism and metamorphism is that polymorphism doesn't change the original code. It just hides it. However, metamorphism morphs or changes the code body itself. In other words, metamorphic code is body polymorphic. Thus the polymorphic code has to be changed to metamorphic code to evade detection completely.

4.3 Working with Live Ransomware Samples- WannaCry and Petya

Here we are working with live ransomware samples, specifically the most recent ones to attack the computer systems the world over-WannaCry and Petya. We do behavior analysis of these samples and study their behavior in detail so that we know their attack strategies and working. Through this study we try to understand the latest malware in a better manner so that we can develop appropriate strategies to create defenses against them.

4.3.1 Recent Impact of Ransomwares

Ransomware pose a grave threat to an organization's profitability as well as reputation. Not only are security managers forced to pay the ransom, they sometimes lose their precious data and most of the times are not even

in a position to make the loss public. In May 2017 WannaCry hit the global markets badly and immediately after, Petya came into light. The impact that they made will be discussed in the following sections.

4.3.1.1 WannaCry

Friday, May 12, 2017 was the day when about 40 hospitals of National Health Service in UK had to shut down their operations because of a major attack on their computer systems by a ransomware called 'WannaCry'. The entire data in these systems was encrypted and lives of many patients were in danger. Similar was the case of 'FedEx' in US, 'Telefonica' in Spain, 'Deutsche Bahn' in Germany and 'Latam Airlines' in South America. Ultimately 150 countries with 2,00,000 computers were attacked by WannaCry and by Monday, some \$50,000 were already paid to the attackers by various companies and individuals. However not everyone's data was recovered even after the ransom was paid. The reason for that is, one cannot tie payment to who you are making it to. The malware goes by various names like 'WannaCry', 'WannaCrypt' and 'WannaCryptor'. It is also called 'EternalBlue'. Wannacry exploits vulnerability in the Service Message (SMB) Block of Microsoft Windows Operating System. After infecting, it encrypts files in the system and renders the system useless unless ransom is paid. It can also spread across network, affecting all computers connected to that network. Microsoft had released a patch earlier itself for this vulnerability. People who failed to patch their system on time were affected. Also, a patch was not released for Windows XP, which has stopped getting support from Microsoft three years back.

4.3.1.2 Petya:

Petya attacked in June 2017, just one month after WannaCry using almost the same SMB vulnerability exploit and NSA's Eternal Blue. However, since many systems were already patched, its impact was very limited. The bitcoin payment system was very simplified and

traceable and not much money was earned in bitcoins. This makes one ponder about whether the real purpose of the attack was actually monetary profit? Petya's first target were the Ukrainian systems, both local and government. It entered the systems through a poisoned update for MeDoc accounting software. MeDoc is primarily used in Ukraine and Russia. It affected Ukrainian banks, airports, power companies and even Chernobyl Nuclear Plant. Outside Ukraine, it affected Danish Shipping company 'Maersk', French company 'Saint Gobain', British advertising company 'WPP', Russian oil production company 'Rosneft' and even 'Jawaharlal Nehru port' in Mumbai, India. Till June 28, less than 150 organisations in Ukraine and less than 50 in US were affected. Thus Petya's main focus was not a fast lateral spread or financial gain but to destroy data, especially in Ukraine systems.

4.3.2 Reasons of Attack of Ransomware

There are many reasons why ransoms like WannaCry and Petya are successful in attacking the systems. Some of the reasons are:

1. **Entering by social engineering:** These ransoms are normally delivered via e-mails which will lure the recipient to click on malicious attachments to open them. This will release the ransomware on the machine and then it would start spreading across the network after infecting the first machine. The ransomware can also enter the machine by downloading a bad app or getting attachments from malicious sites.
 2. **Not updating systems regularly:** All operating systems regularly release patches for their updation after analysing different security threats. In the case of WannaCry, Microsoft had already released a patch called MS17-010 after realizing about the NSA leak. However, many people failed to update their machines with this patch and were thus vulnerable to ransomware attack. Three years ago Microsoft stopped supporting Windows XP systems. Thus those were the firsts to be attacked. A patch was released for them also after the attack.
 3. **Non-Updation with Patches:** This security lapse was known earlier this year and Microsoft had released a patch called MS17-010. However the attackers
-

chose Windows XP operating systems because Microsoft had stopped supporting Windows XP three years ago. Also no patch was released for this operating system. So all systems still using Windows XP or higher versions of unpatched Windows systems were attacked. Therefore Microsoft released an emergency patch over the weekend for Windows XP systems.

4. **SMB vulnerability:** In MS Windows, there are some ports which are open by default[37]. One of them is the SMB(Service Message Block) port used for file and printer sharing. WannaCry and Petya both spread using this open port. SMB stand for Service Message Block. It runs as a thin layer on top of TCP protocol. It provides file and printer sharing facilities between different Windows machines. This is done using Inter Process Communication. It can run either directly on TCP(port 445) or using NetBIOS protocol on UDP ports (137,138) and TCP ports(137,139) or on legacy protocols like NBF and IPX or SPX. It is also called CIFS- Common Internet File System. Another protocol called 'Samba' was created for communication between Windows and non-Windows systems. Samba is an open-source, free reimplement of SMB for non-windows systems like UNIX and Linux. Actually, Windows opens the SMB and the NetBIOS ports by default for both local and outside access. WannaCry scans for an open SMB port to enter into the system. After that, it spreads just like any other worm[38]. Microsoft had released the second version of SMB called SMB v.2. United States, National Security Agency(NSA) had discovered the SMB vulnerability long time ago. However, instead of reporting it to Microsoft, it created 'EternalBlue'. EternalBlue is an exploit based on this vulnerability. It was built to be used in future as a cyber-weapon for NSA. Now, there is a group called 'Shadow Brokers'. This group leaked EternalBlue. After the leak, Microsoft released a patch for this. However, companies and individuals who failed to update, were attacked. Not only was EternalBlue leaked, which was used in WannaCry, but many other NSA exploits like EternalChampion, EternalSynergy, EternalRomance, EmeraldThread and EducatedScholar were also leaked. These can easily be used for much bigger hacks. In fact, The Shadow Brokers have claimed to have many more leaks under their belt waiting to be released.
-

5. **No Network Segmentation:** If the network is not properly segmented, the ransomware spreads at a very fast rate. However, if the network is segmented then the spread of the ransomware can be contained and the mitigation may also be easily possible.

4.3.3 Behavioral analysis of WannaCry and Petya:

- **Wannacry:**

1. **Attack Process:** Once WannaCry enters the victim computer, it scans heavily for the SMB port vulnerability. If the port is found open, an SMB connection is set up. Then the patch MS17-010 is searched for. If it is not found then an encrypted shellcode is prepared in Base64 encoding and is heap sprayed in the memory. Once found, the shellcode looks for DoublePulsar. This backdoor changes permission to remote access and calls a particular domain to see if it is registered. This is done to find out whether the malware is working in a sandbox or not. If the domain is not registered, SMB exploitation starts getting performed. A file called 'taskche.exe' is created which will carry out file encryption and will also help the malware spread across the network. In the meantime, TOR is used for all communications, bitcoin wallets are loaded, public and encryption key is prepared. The files start getting encrypted. WannaDecryptor.exe is set up and Please_Read_Me.txt file is created. The screen is locked and ransom message is displayed.
 2. **File system analysis:** The payload is 'taskche.exe' which is created from the worm's resource '1831'. It has a huge size because of bundled TOR executables along with other tools and configuration files. The actual name of this file is 'DiskPart.exe'. This is a MS utility for disk partitioning. WannaCry creates two DLL files in the memory- a 32 bit dll and a 64 bit dll. Both of them contain an export called 'PlayGame'. This
-

writes a copy of the original worm to C:\WINDOWS\mssecsvc.exe and executes it. There are many files which are created in the process. B.wnry is a bitmap image containing a ransom note in it. C.wncr is a binary configuration file. It also has addresses of the TOR sites. R.wnry is a text file with a ransom note in it. S.wnry is a zip file with TOR executable. T.wnry is a dll. It is actually an encryption tool. Taskse.exe starts @WannaDecrytor@.exe. It is a support tool. U.wnry is a decryptor executable that opens GUI with a ransom note in it. Taskdl.exe is a support tool for deleting temporary files. msg\m_*.wnry is a directory with ransom note in different languages like English, Chinese, Bulgarian etc.if WannaCry is launched with less than 2 arguments, it installs a service called mssecsvc.exe. This is Microsoft Security Center Service, version 2.0. It drops the WannaCry binary and runs it. If the malware was run with two or more arguments, it enters the service mode. Files are encrypted with the extension of .wncrypt. Some libraries are loaded at runtime like kernel32, user32, advapi32.dll, shell32.dll, msvcrt.dll, mscpl60.dll.

3. Persistence analysis: Persistence ensures that the malware will continue to run even after the machine is rebooted, restarted or is logged off. For persistence, entering into the run keys in the registry is important. Two registry entries are required to ensure persistence:

- Key:HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\<Random> Value:<Full_path>\tasksche.exe . CU stands for Current User. %Random is a pseudorandom name derived from the current computer name. This is a user level entry.
- Key:HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\<Random> Value:<Full_path>\tasksche.exe. LM stands for Local Machine. This is a system level entry. Full_path is the path of the exe file.

If the DiskPart.exe file finds that it was running without the '/i' switch, this means that the worm has not executed it. Therefore, the file registers itself as a service so that it gains persistence so that the worm is no longer required to run it.

4. Network analysis: The network analysis is obtained by observing the pcap file. The important property of this ransomware which makes it different from the previous ransoms is that it does not need to call a malicious server for encryption of the files. If the binary is in the system, it does not need to be connected to the Internet for encryption of the files. Other ransoms before it, used to connect to the c&c servers to get the encryption key. Wannacry starts its execution by trying to connect to a domain 'www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com'. If the machine fails in making the connection, the exploitation continues. However, if the connection is successful, the exploitation is stopped immediately. Actually, this is a way of 'Sandbox Evasion' technique. Once WannaCry is sure that the domain is unregistered, the exploitation begins. However, it searches for a backdoor called 'DoublePulsar' before actual exploitation. So the attacker sends a SMB 'trans2SESSION_SETUP' request to the infected system. This is a Transaction 2 Subcommand Extension. This is to find out whether the system is already compromised with DoublePulsar or not. The system responds with a "Not Implemented" message and a "Multiplex ID" is returned as part of this message. This has a value of 65(0x41) for the normal systems and 81(0x51) for infected systems. If the system is infected, then SMB can be used as a covert channel to install WannaCry. Then the exploitation starts by setting up Windows Socket APIs. Wannacry uses MS17-010 exploit to laterally spread to other machines through open NetBIOS ports[39]. Once it finds a machine with an open NetBIOS port, it sends three NetBIOS session setup packets to it. One contains IP address of the machine to be exploited and the other two contain two IP addresses which are hardcoded in the malware only. It then spawns 2 threads, the first thread enumerates the network adapters and finds the address of the subnet of the compromised system. The first thread then generates threads for each IP address on this subnet. Each of these threads try to get connected to port 445 and if successful, the entire process of exploitation starts again on the newly infected machine.128

instances of the second thread can be created within 2 seconds. This explains the widespread of this malware within a short span of time. WannaCry also downloads files from the dist.torproject.org and all further communications for bitcoin transfer happens through Tor only.

- **Petya:**

1. **Attack Process:** Petya was a ransomware which came in March 2016. The strain of this ransomware which came on 27 June 2017 was very different from the original version. Thus it was called 'NotPetya'. The ransomware uses Eternal Blue exploit. The infection starts with a poisoned update for the MeDoc accounting software[40]. Petya infects the MBR and executes a malicious payload that encrypts the file system of the hard drive and does not let Windows system to boot. Like WannaCry takes the help of DoublePulsar for completing its execution, in a similar manner, Petya has joined hands with a traditional file-based ransomware called 'Misha'. Petya needs to have administrative privileges to carry out low-level encryption of files and other malicious works. If those privileges cannot be obtained, Misha is launched. A deep analysis of Petya shows it is actually a 'Wiper'. Normally files encrypted by Petya cannot be recovered. It wipes away the files and they can never be found again.
 2. **File system analysis:** Petya's infection mechanism is to attack low level structures. It overwrites the beginning of the disk which is the Master Boot Record(MBR) by its own boot loader. This boot loader then loads its 32 sector kernel which then starts the encryption process of the files. AES-128 is used for encryption and the 128 bit key is randomly generated. This key is again encrypted with RSA 2048 encryption algorithm. Petya starts with the file explorer.exe. Then ezvit.exe affects the Medoc software. Then rundll32.exe is dropped and unicryptC.exe is used for encryption.
-

3. Persistence analysis: MBR and NTFS boot record are the only persistence mechanisms in case of Petya.

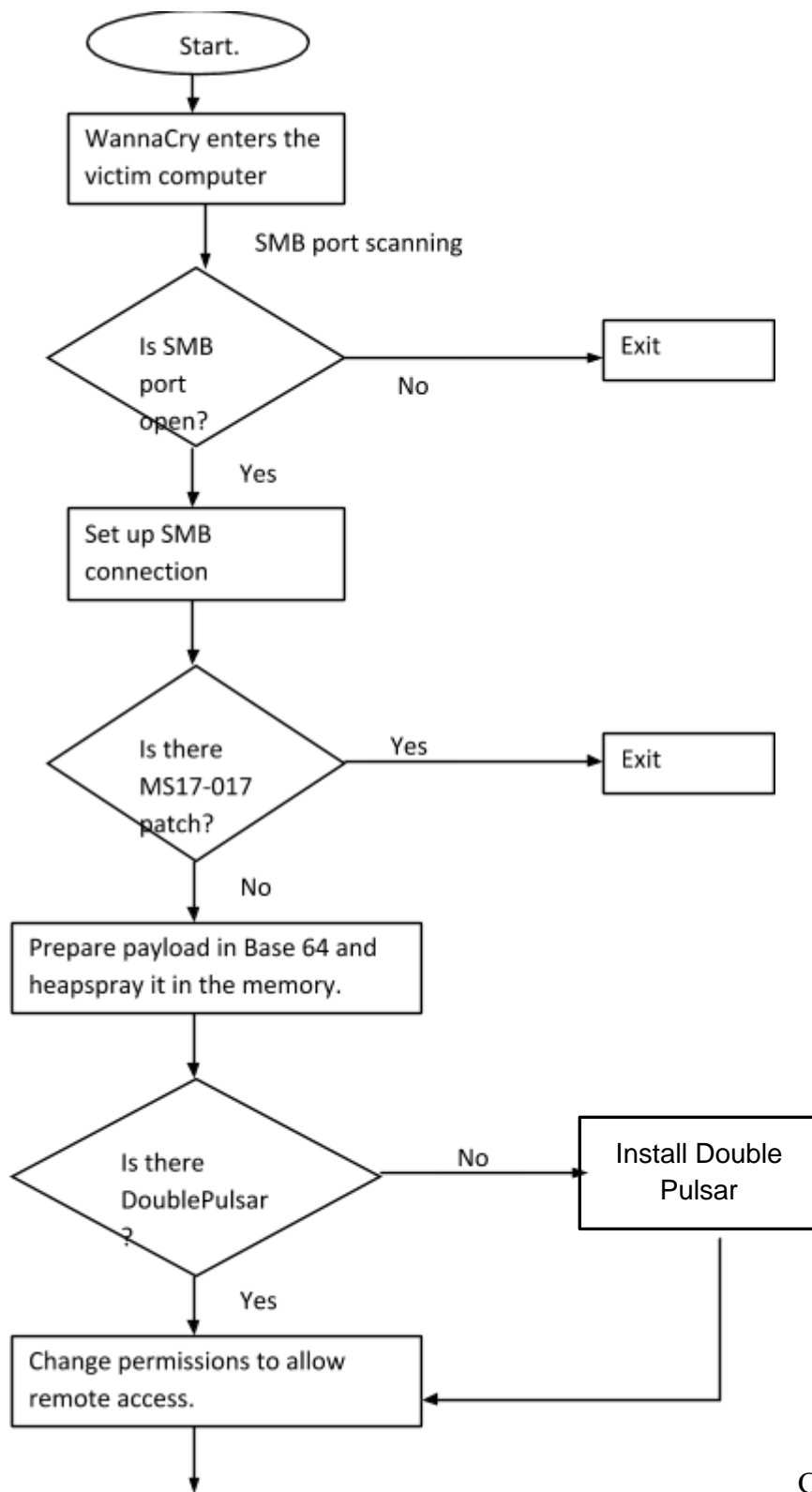
4. Network analysis: For lateral propagation, in addition to EternalBlue exploit, WMI Commands, Mimikatz and PSEXEC were also used. Unlike WannaCry which infected a small number of computers and was designed to spread at a fast rate to the Internet, Petya is designed mostly for lateral propagation in the local network. It infected a large number of computers but the spread was slow and local rather than global. The SMB ports were scanned. If they were found to be unpatched for the EternalBlue exploit, this vulnerability was used to compromise the systems. However, if a patch was found, WMIC (Windows Management Instrumentation Command) was used for stealing local machine credentials using Mimikatz and then finding remote shares to spread. The WMI is a complete infrastructure for the Windows operating system which does management and administration of the operating system both locally and remotely. It is a very powerful tool but the same power can be used by attackers for remotely executing ransomware in target machines. There are various WMIC commands which were used by Petya for several purposes like detection of antivirus and virtual machines, persistence, theft of data, lateral movement and execution of various commands[41]. Mimikatz is used for stealing user credentials and through them executing Petya on infected machines. One of the WMIC commands used by Petya for executing powershell on a remote machine is:

```
wmic /node: [IP Address] /user: "[user name]" /password: "[password]" process call create "cmd /c powershell.exe user"
```

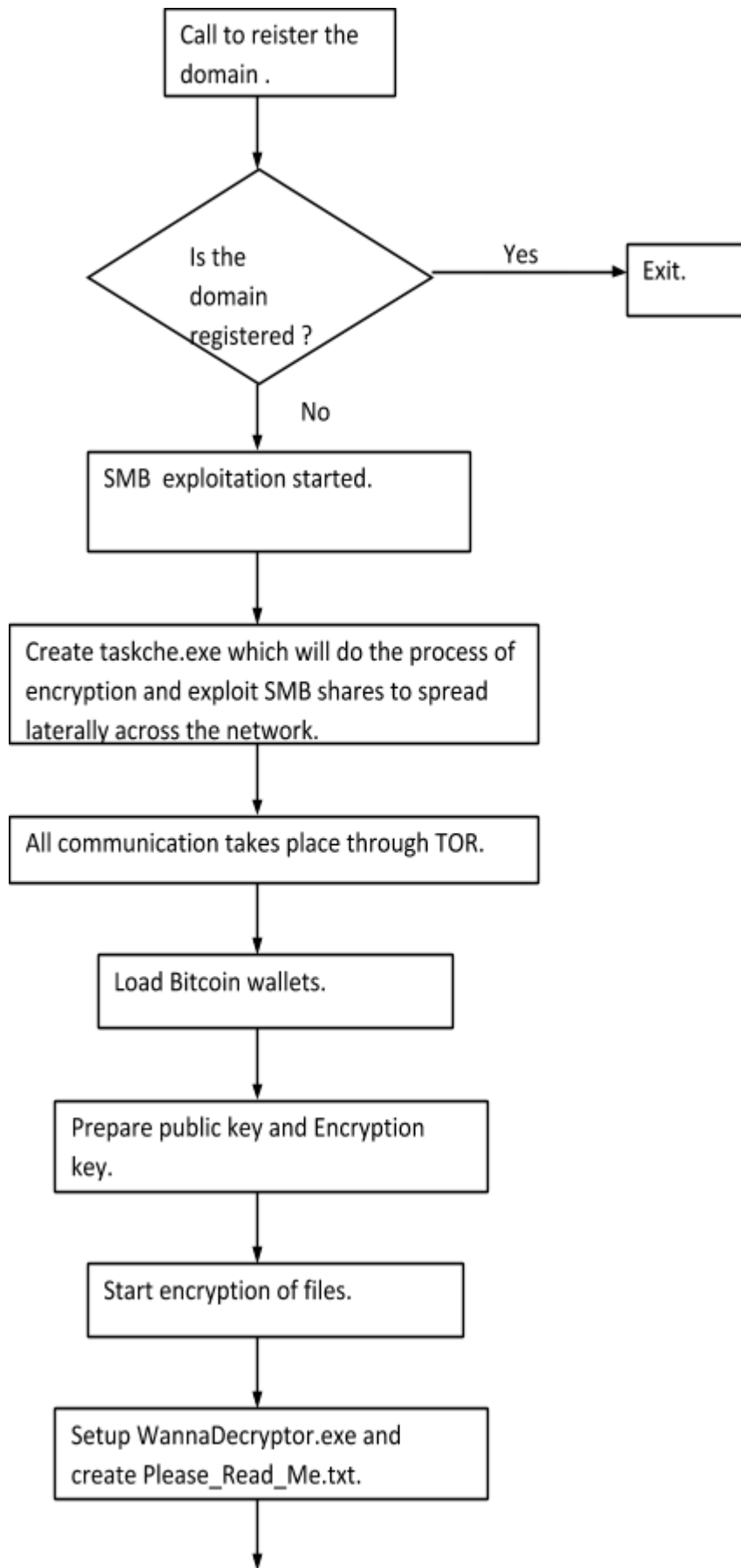
4.3.4 Working of WannaCry

Once WannaCry enters the victim computer, it scans heavily for the SMB port vulnerability. If the port is found open, an SMB connection is set up. Then the patch MS17-010 is searched for. If it is not found then an encrypted shellcode is prepared in Base64 encoding and is

heapsprayed in the memory. Once found, the shellcode looks for DoublePulsar. This malware downloader changes permissions to remote access and calls a particular domain to see if it is registered. This is done to find out whether the malware is working in a sandbox or not. If the domain is not registered, SMB exploitation starts getting performed. A file called 'taskche.exe' is created which will carry out file encryption and will also help the malware spread across the network. In the meantime, TOR is used for all communications, bitcoin wallets are loaded, public and encryption key is prepared. The files start getting encrypted. WannaDecryptor.exe is set up and Please_Read_Me.txt file is created. The screen is locked and ransom message is displayed.



Continued on next page.



Continued on next page.

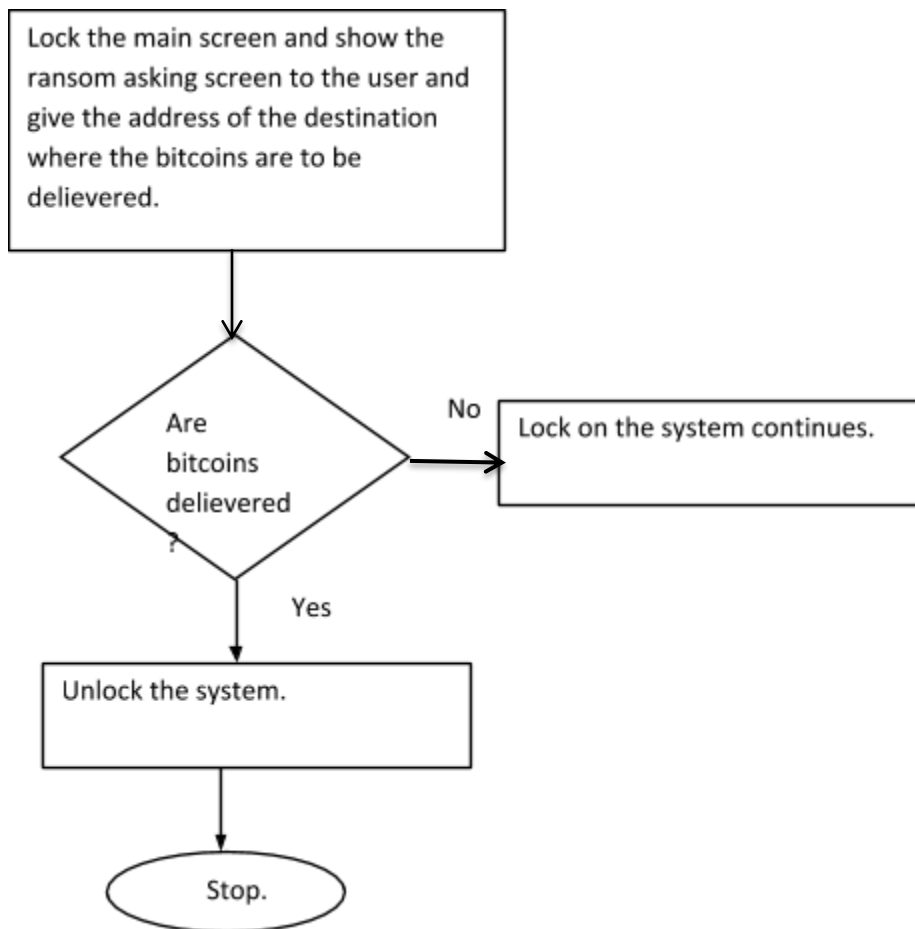


Figure 4.3 Flowchart showing working of WannaCry

4.3.4.1 Components of WannaCry

WannaCry is made up of the following components:

4.3.4.1.1 DoublePulsar

DoublePulsar is a malware downloader. Its basic purpose is to download additional malwares into the system. DoublePulsar is normally loaded into the machine before WannaCry and it is running as a background process undetected. WannaCry, after getting loaded, searches for DoublePulsar. After getting permission from WannaCry, DoublePulsar alters user mode process permissions and sets up remote access. Once connected, DoublePulsar deletes itself. DoublePulsar was originally developed by 'Equation Group' and was leaked by 'The Shadow Brokers' along with 'EternalBlue' in early 2017 [42]. So it was used with EternalBlue to

carry out WannaCry attacks. Also, once a system is rebooted, DoublePulsar does not persist. DoublePulsar runs in kernel mode. However, this kernel payload does not load the actual DLL(Dynamic Link Library). It actually sets up an Asynchronous Procedure Call(APC) to another shellcode that performs the load. Since it does not make a LoadLibrary call, which is a local call, therefore the DLL is not written to the disk. So there is no entry in the Process Entry Block(PEB). Thus, it remains hidden.

4.3.4.1.2 TOR

TOR stands for 'The Onion Router'. It is a worldwide network of computers, initially developed by the U.S. Navy that helps people to browse the Internet anonymously[43]. TOR hides your identity because it moves your traffic across random TOR servers and encrypts the traffic for many times including the next node address and sends it through a virtual circuit comprising successive, randomly selected TOR relays. Each relay decrypts a layer of encryption to reveal only the next relay in the circuit.

4.3.4.1.3 Domain Check

Just a few days after WannaCry was launched, a man who goes by the name @MalwareTech, discovered a Kill Switch to stop the attack. This slowed down the destruction by the malware to a large extent. Actually, before WannaCry starts the SMB exploitation, it tries to connect to a domain addressed, '<http://www.iuqerfsodp9ifjaposdfjhgosurijfaewrwergwea.com>'. If the machine fails in making the connection, the exploitation continues. However, if the connection is successful, the exploitation is stopped immediately. Actually, this is a way of 'Sandbox Evasion' technique. Whenever a malware attacks a machine, it ensures it is not running in a sandbox. A sandbox is a way employed by Antiviruses and IDPS systems. In this, any suspected program runs in a protected, silo environment, detached from the main operating system with limited resources. If any permissions for connections are asked for, example in this case, the malware asked for permission to connect to a domain, the permission is granted. This is to trick the malware into assuming that it is running in an actual environment and not in a sandbox. Therefore, the registration of this domain acted as a kill switch for Wannacry.

4.3.4.1.4 Bitcoin Wallets

Bitcoin is cryptocurrency and a digital payment system invented by a programmer called 'Satoshi Nakamoto'. It is an open software and was released in 2009. Wannacry asks it's victims to pay in bitcoins for getting the decryption key. Before that, Bitcoin wallets are created in your system. It is a peer-to-peer system which takes place between users directly without an intermediate person. The transactions are recorded and verified in a public distributed ledger called 'Blockchain'. Bitcoin is thus a decentralized digital currency.

4.3.5 Precautions to be safe from ransomware

After the huge destruction caused by WannaCry, some precautionary steps can be taken to prevent such an outbreak in the future[44]. First of all, any ports which are not required to be open at all times must be closed and opened only when that service is requested for. Second, one must keep all her systems, completely updated using patches whenever they arrive. Regular updation is likely to protect your systems from many vulnerabilities and malwares. Third, network segmentation plays a key role in preventing an infection to spread across other systems in the network thus protecting other systems and also containing the infection. Preventive actions like not clicking on suspicious files in your mails and protecting your system's user id and password are anyway, very important. Lastly any individual or organization which finds a vulnerability must make it public for larger interest of the society and not hoard it for creation of cyber warfare.

1. **Good AV or IDS solution:** A good and reliable antivirus and IDS solution goes a long way in protecting the network from known and unknown attacks. Antiviruses keep updating themselves when new strains of malwares are discovered. Most of the malwares are successfully blocked by good antiviruses.
 2. **Backup and Restore:** Always take a regular backup of your data so that even if the ransomware attacks, you have the most current backup and are able to restore your data without paying the ransom amount and becoming a victim to the attacker.
 3. **Make a vulnerability public:** As in the case of EternalBlue, the SMB vulnerability was known to NSA but they chose not to disclose it to Microsoft,
-

rather, they made cyber weapons for them. These cyber weapons were leaked by Shadowbrokers group and WannaCry and Petya were created out of them. It is always in the interest of the public and government both that any vulnerability in any operating system, if known by an individual or an organisation, should be made public so that systems can be protected from it.

4.3.6 Remedy after ransomware attack

Though one can only take precautionary steps to get saved fully from WannaCry, however once infected, one can recover the keys only till the time the system is rebooted. Actually for encryption, WannaCry creates two keys -a 'public key' and a 'private key'. These are based on prime numbers and are used for encryption and decryption of files. These keys remain lingering in the memory and are not deleted till the system is rebooted. These can therefore be recovered and be used for decryption.

4.4 Building PosDeF

There are various tools that we require to build the framework. For our virtualization need we have used virtualbox with Windows 7 installed. Dynamic analysis is carried by Cuckoo framework. Cuckoo framework makes it easy to carry dynamic analysis on malwares it comes inbuilt with various packages like tcpdump which is used to extract network communication between malwares and its author, volatility framework for extraction of system calls during the execution, python pillow library for extraction of screenshot during execution and also automating the clicks on various clickable items during the execution. All these factors can contribute into prediction of type of file.

4.4.1 Building the Static Part of PosDeF

Static analysis refers to the act of extracting information based on file properties without running it. This is the quickest way to classify the file but not always accurate. These are metadata of a file. We have extracted a total of 52 parameters using a python module called **PE Analyzer 2**. PE Analyzer is a tool which is used to analyze the portable executable format of a file. Some technical indicators like file name, MD5 checksums or hashes, file

type, file size provide the preliminary view of the file to determine whether it is malicious or benign.

There are various steps involved in building the Static part of the framework:

1. Feature Extraction
2. Feature Selection
3. Feature Classification

4.4.1.1 Feature Extraction

Extract the static features from the binary files by using PE Analyzer. These are the features, which distinguish between a malware and a benign file.

1	Name	md5	Machine	SizeOfOptional	Characteristic	MajorLinkerV	MinorLink	SizeOfCod	SizeOfIniti	SizeOfUnii	AddressO	BaseOfCo	BaseOfDa	Im
2	memtest.exe	631ea355665f28d4707448e442fbf5b8	332	224	258	9	0	361984	115712	0	6135	4096	372736	4
3	ose.exe	9d10f99a6712e28f8acd5641e3a7ea6b	332	224	3330	9	0	130560	19968	0	81778	4096	143360	7
4	setup.exe	4d92f518527353c0db88a70fddcfd390	332	224	3330	9	0	517120	621568	0	350896	4096	811008	7
5	DW20.EXE	a41e524f8d45f0074fd07805ff0c9b12	332	224	258	9	0	585728	369152	0	451258	4096	798720	7
6	dwtmgr20.exe	c87e561258f2f8650cef999bf643a731	332	224	258	9	0	294912	247296	0	217381	4096	536576	7
7	airappinstaller.exe	e6e5a0ab3b1a27127c5c4a29b237d823	332	224	258	9	0	512	46592	0	4488	4096	8192	4
8	AcroBroker.exe	dd7d901720f71e7e4f5fb13ec973d8e9	332	224	290	9	0	222720	67072	0	219331	4096	229376	4
9	AcroRd32.exe	540c61844ccd78c121c3ef48f3a34f0e	332	224	290	9	0	823808	650240	0	587663	4096	831488	4
10	AcroRd32Info.exe	9afe3c62668f55b8433cde602258236e	332	224	290	9	0	4096	7168	0	6751	4096	8192	4
11	AcroTextExtractor.exe	ba2e21a96e44f6558c08cf25b40cb1bd4	332	224	290	9	0	29696	12800	0	27055	4096	36864	4
12	AdobeCollabSync.exe	bf0a35c0efcaf650550b9e346dfcbdb33	332	224	290	9	0	917504	316928	0	833800	4096	921600	4
13	Eula.exe	1556a34d117a80bdc85a66d8ea4fbcf2	332	224	290	9	0	53248	34816	0	53601	4096	57344	4
14	LogTransport2.exe	c4005b63df77068bce158ac8ef7c522b	332	224	258	9	0	206848	102400	0	110150	4096	212992	4
15	reader_sl.exe	e595f220ed529885d8bc0ef42e455e4d	332	224	259	9	0	14848	14336	0	16529	4096	20480	4
16	AcrobatUpdater.exe	0e9dee95fdf47d6195da804a0deeda5b	332	224	258	9	0	178688	134144	0	78084	4096	184320	4
17	AdobeARM.exe	47c1de0a890613ffcff1d67648eedf90	332	224	258	9	0	413184	518144	0	160191	4096	417792	4
18	armsvc.exe	11a52cf7b265631deeb24c6149309eff	332	224	258	9	0	37376	20992	0	30988	4096	45056	4
19	ReaderUpdater.exe	5ed9b78b308d302c702d44f4505b3f46	332	224	258	9	0	178688	134144	0	78084	4096	184320	4
20	Adobe AIR Application	2da20164a6912ca8a11bb3089d0f3453	332	224	258	9	0	32256	91136	0	5926	4096	36864	4
21	Adobe AIR Updater.exe	397ef02798d24bf192997b5f7d8ed8ca	332	224	258	9	0	31744	64512	0	5275	4096	36864	4
22	template.exe	86fbd3c4793f2b2e85bcc000fafbb7	332	224	258	9	0	32256	26624	0	5926	4096	36864	4
23	csscan.exe	04541302c404ec763380481346cbe0fb	332	224	258	8	0	34304	54784	0	30939	4096	40960	4
24	dainstall.exe	444caa5fc97729ef7da6965bae11c184	332	224	258	8	0	37376	30720	0	39387	4096	45056	4
25	entvutil.exe	5d0df5cbffd37853e623565cd081e47b	332	224	258	8	0	13824	7168	0	6123	4096	20480	4
26	fwinfo.exe	c3879b7359dd566ed8cd6d65846ae1bc	332	224	258	8	0	82944	95744	0	50929	4096	90112	4

Figure 4.4 Snapshot of a part of the ‘data.csv’ file created for training and testing purpose

This is an example of the feature vector for a file:

Feature vector:

Name|md5|Machine|SizeOfOptionalHeader|Characteristics|MajorLinkerVersion|MinorLinkerVersion|SizeOfCode|SizeOfInitializedData|SizeOfUninitializedData|AddressOfEntryPoint|BaseOfCode|BaseOfData|ImageBase|SectionAlignment|FileAlignment|MajorOperatingSystemVersion|MinorOperatingSystemVersion|MajorImageVersion|MinorImageVersion|MajorSubsystemVersion|MinorSubsystemVersion|SizeOfImage|SizeOfHeaders|Checksum|Subsystem|DllCharacteristics|SizeOfStackReserve|SizeOfStackCommit|SizeOfHeapReserve|SizeOfHeapCommit|LoaderFlags|NumberOfRvaAndSizes|SectionsNb|SectionsMeanEntropy|SectionsMinEntropy|SectionsMaxEntropy|SectionMeanRawsize|SectionsMinRawsize|SectionMaxRawsize|SectionsMeanVirtualsize|SectionsMinVirtualsize|SectionMaxVirtualsize|ImportsNbDLL|ImportsNb|ImportsNbOrdinal|ExportNb|ResourcesNb|ResourcesMeanEntropy|ResourcesMinEntropy|ResourcesMaxEntropy|ResourcesMeanSize|ResourcesMinSize|ResourcesMaxSize|LoadConfigurationSize|VersionInformationSize|**legitimate/malicious**

Legitimate File:

AcroRd32Info.exe|9afe3c62668f55b8433cde602258236e|332|224|290|9|0|4096|7168|0|6751|4096|8192|4194304|4096|512|5|0|0|5|0|24576|1024|28316|233088|1048576|4096|1048576|4096|0|16|5|4.14491201014|0.393689010804|5.97744194837|2252.8|512|4096|2177.6|664|3880|3|61|0|0|4|5.09749900993|3.43599300049|5.92981166069|616.5|94|1164|72|15|1

Malicious File:

VirusShare_a69e89bbf39a25966660881912ec1a84|a69e89bbf39a25966660881912ec1a84|332|224|258|10|0|119808|385024|0|61532|4096|126976|419430

4|4096|512|5|1|0|0|5|1|528384|1024|564364|2|33088|1048576|4096|1048576|4
096|0|16|5|5.6663371314|4.18953861211|7.96311398057|100966.4|9728|330
752|102644.2|9418|339748|3|90|0|0|6|3.77675976631|2.45849222582|5.3175
5235629|2740.16666667|48|9640|72|15|0

4.4.1.2 Feature Selection

The features that are chosen are of the type integers or floats. The text based features are dropped. These are vectors because only computation on vectors can be used by algorithms. These features are then used by our system. Out of 54, 14 features are selected. Rest of the features which were not important for analysis was dropped.

In the pre training phase, some features have to be selected according to their variance. Feature selection means selecting only those features in the data that contribute most to the output in which one is interested. The ‘**Skikit Package Tree Classifier**’ is used here. Tree classifier gives a simple set of rules to categorize data. During training, a node-splitting criterion is utilized to partition the input space so as to classify the training data points in each position. This process is applied recursively within each resulting partition not meeting a stopping condition [45]. In the training phase the Tree Classifier algorithm creates a decision tree by identifying patterns in an existing dataset and using that information to create the tree.

When we select features, those which have a very high or a very low variance are not useful in machine learning. These are rejected. Skikit package tree classifier is used to select only those features which are helpful in the separation of two classes.

It is important to understand that all these features must vary for them to be selected. Variance measures how a set of numbers is spread out. The features which are selected according to their variance are as follows:

1. DllCharacteristics
-

2. Machine
3. ResourcesMaxEntropy
4. MajorSubsystemVersion
5. Subsystem
6. Characteristics
7. VersionInformationSize
8. ImageBase
9. SectionsMaxEntropy
10. MajorOperatingSystemVersion
11. SectionsMinEntropy
12. MinorSubsystemVersion
13. SizeOfOptionalHeader
14. SectionsMeanEntropy

4.4.1.3 Feature Classification

These features differ according to their variances. Therefore, these features are ranked according to the feature showing the highest variance till the feature showing the lowest.

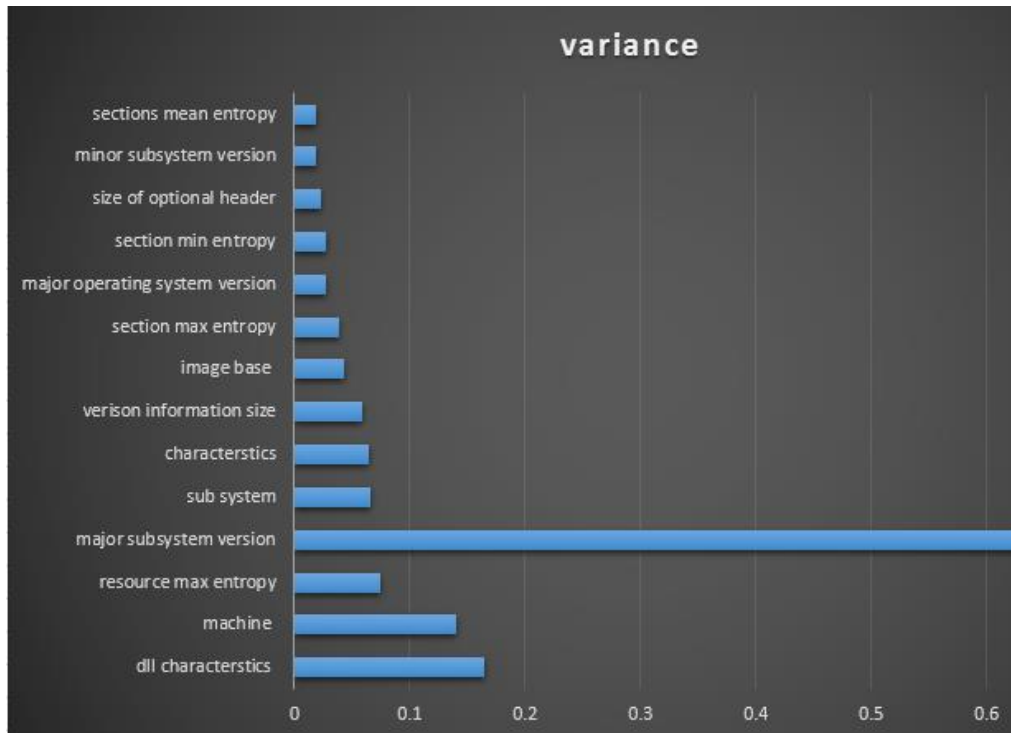


Figure4.5: Features selected on the basis of their variance

Let us understand these features one by one:

1. **DllCharacteristics:** This field comes under ‘Optional Header Windows – Specific Fields’ [46]. It contains information about dynamic linking and loading behavior of a file. All information in DLL is encoded as flags which are represented as on/off bits in 16 bits. For example, 0X0040 shows that the DLL can be relocated at load time.
2. **Machine:** Architecture type of the computer.
3. **ResourcesMaxEntropy:** The increased entropy in a PE file is an indicator that the file has been processed by a packager or a protector. It is most likely to be malicious.
4. **MajorSubsystemVersion:** Tells the major subsystem version for a system.
5. **Subsystem:** This is a 16 bit value which tells the subsystem for the operating system. For Eg. Windows 95 binaries will always use the Win32 subsystem.
6. **Characteristics:** Collection of flags valid for libraries and object files.

7. **VersionInformationSize:** Tells the size of the version information. Version has information like file version, operating system version, original filename etc.
8. **ImageBase:** This tells what is the preferred address of the first byte of image when it is loaded into the memory.
9. **SectionsMaxEntropy:** Tells the maximum allowed entropy for a section.
10. **MajorOperatingSystemVersion:**The major version number of the required operating system
11. **SectionsMinEntropy:** Tells the minimum allowed entropy for a section.
12. **MinorSubsystemVersion:** Contains the minimum subsystem version required to run the executable. A typical value for this field is 3.10 (meaning Windows NT 3.1).
13. **SizeOfOptionalHeader:** This is required for executable files and not for object files. Its value is zero for object files.
14. **SectionsMeanEntropy:** Tells the mean entropy for a section.

In the training phase we will test various algorithms and split the dataset into 80% for training and 20% for testing. We have to test all the classification algorithms with the default parameters and the results that we get are given in the screenshot below.

```
Researching important feature based on 54 total features

14 features identified as important:
1. feature DllCharacteristics (0.164553)
2. feature Machine (0.140578)
3. feature ResourcesMaxEntropy (0.074606)
4. feature MajorSubsystemVersion (0.067075)
5. feature Subsystem (0.065116)
6. feature Characteristics (0.064077)
7. feature VersionInformationSize (0.058503)
8. feature ImageBase (0.042890)
9. feature SectionsMaxEntropy (0.038094)
10. feature MajorOperatingSystemVersion (0.027357)
11. feature SectionsMinEntropy (0.026837)
12. feature SizeOfOptionalHeader (0.023347)
13. feature MinorSubsystemVersion (0.018744)
14. feature SectionsMeanEntropy (0.018684)

Now testing algorithms
DecisionTree : 98.953278 %
RandomForest : 99.319087 %
GradientBoosting : 98.721478 %
AdaBoost : 98.478812 %
GNB : 70.137631 %

Winner algorithm is RandomForest with a 99.319087 % success
Saving algorithm and feature list in classifier directory...
```

Figure 4.6 Screenshot showing performance of various classification algorithms for testing of static analysis

Therefore, in this case, we find that RandomForest classification algorithm is giving us the best result of a 99.35% success. This was being tested with default parameters. The False positive/ False negative rate on the test data is as follows:

- a. False positive: 0.568241
- b. False negative: 0.830565

4.4.2 Building the Behavior Part of PosDeF

Static analysis of malware is not enough because malwares have become very smart in the recent decade. A polymorphic or a metamorphic malware changes its signature and behavior at runtime. Therefore, in addition to checking the signature of the file to be analyzed, we need to check its behavior also to determine whether it is malicious or benign. We will use various tools in a simulated environment to discover the behavior of a particular file at runtime. After

discovering the behavior, we will use various machine learning algorithms to predict the type of file. When we are considering self-modifying and polymorphic code, static analysis fails and we have to move to dynamic analysis for the right detection.

Steps:

- 1) Files in binary format enter the Cuckoo system and are analyzed.
- 2) Based on the analysis results, API call sequences are extracted and a behavior report is generated for each binary.
- 3) These reports are converted to MIST format.
- 4) This MIST format is converted into sequential data in a high-dimensional vector space by N-gram algorithm.
- 5) Similarity of vectors is calculated and a sparse matrix is generated.
- 6) Many classification algorithms work on this sparse matrix and according to our dataset; KNN (K nearest neighbor) algorithm shows best accuracy.
- 7) KNN is applied on this sparse matrix.

4.4.2.1 File Analyzed in Cuckoo Sandbox

Cuckoo is a very popular sandbox [47] which is free and open-source system, provided by the Cuckoo Foundation. After analysis of the sample, it gives a detailed report of it. Not only does Cuckoo help in reporting whether a sample is malicious or benign, but it also help a researcher in understanding the way a malware operates, reasons for the breach and the final motive. Cuckoo can analyze only malicious files under Windows, OSX, Linux and Android.

Operations that Cuckoo can perform are [48]:

- 1) Analyze a malicious file sample.
 - 2) Trace all API calls and behavior of the sample.
 - 3) Analyze even SSL/TLS encrypted traffic.
 - 4) Do memory analysis of the infected system.
-

- 5) Can find IP addresses, domains, file hashes etc. which tell about network-related compromises.

Cuckoo comes inbuilt with various packages like tcpdump, volatility framework and python pillow. Tcpdump is a network sniffer which captures the traffic of the malware during execution and dumps it into a file. Volatility Framework extracts system calls during execution [49]. It starts working after we have dumped the memory. This memory is volatile memory-RAM. Digital artifacts can be extracted from it. Rootkits can normally be detected using Volatility framework [50]. Python pillow library is used for taking screenshots of the OS desktop during execution.

Cuckoo Architecture:

The architecture of Cuckoo sandbox will typically consist of hosts and guests connected together by a virtual network. A Cuckoo host takes care of guests and analysis management. It starts analysis, dumps traffic and generate reports. A guest provides a clean environment where a sample file can be run. The behavior of the analyzed sample is reported back to the host. After submitting a sample to the host, the analysis is launched in a fresh and isolated machine.

4.4.2.2 JSON Reports Generated

- 1) We have used virtual box with Windows 7 installed and dynamic analysis of samples is carried out by the Cuckoo framework. Cuckoo is set upon Ubuntu host.
 - 2) We have used 2000 sample size. 1000 files are extracted from virusshare_00302.zip which is a collection of malwares. 1000 .exe files are extracted from a clean installation of Windows 7.
 - 3) We start submitting the samples to Cuckoo. Instead of submitting the 2000 samples manually, we have created a small python script to automate the process. We have created two directories- one for clean and the other for malicious files. Our code will submit files one by one to Cuckoo from these directories.
-

- 4) After the analysis, Cuckoo generates a JSON (JavaScript Object Notation) file. A JSON file is normally a 25-30 page file.
- 5) After we submit a sample in Cuckoo, it will open a VM (Virtual Machine) and execute the file. Then it will start collecting stats about that file.

4.4.2.3 JSON Reports converted into MIST format

A JSON file, though human readable, but is very lengthy and will take plenty of resources to be processed by a machine learning algorithm. This is because it is unstructured. The behavior patterns are not recognizable. Also complex and lengthy textual representations negatively impact the run time of analysis. Konard Reich at the University of Gottingen have developed a method called ‘Malheur’[51] in which they have created MIST format which is a very efficient format for representing Cuckoo generated JSON files. MIST stands for ‘Malware Instruction Set’. It is a feature generation technique that represents the behavior analysis of a sample as a series of integers [52]. They have used it to classify malware into different families and equally good for binary classification i.e. benign and malicious samples.

MIST creates a smaller file size for analysis and reduces the processing time. MIST format is also not human readable but stores the same information in a structured manner in a very limited space. It is like the instruction sets used in processor design [53]. For conversion of the JSON file into MIST format, we have used an open source tool called ‘cockoo2mist’.

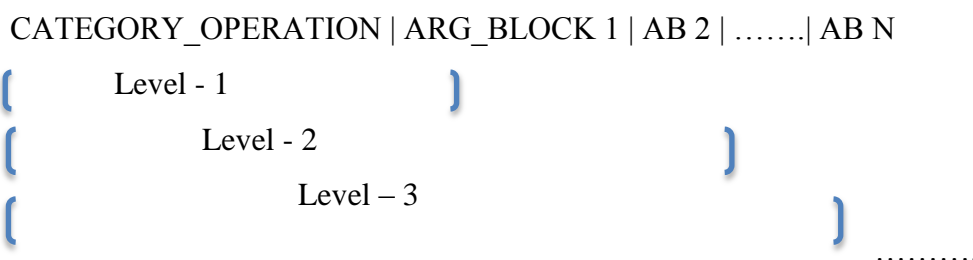


Figure 4.7: Depiction of a MIST Instruction.

Understanding a MIST instruction

Level-1: Shows category and name of a monitored system call. For eg.

03 05

Here category is 'filesystem'(03)and system call is 'move_file'(05).

Operation – is a particular system call.

Arg_Block- arguments like file and mutex names.

The following figure represents a CWSandbox representation of a load_dll command being converted into MIST format.

```
<load_dll filename="C:\WINDOWS\system32\kernel32.dll" successful="1"
address="#7C800000" end_address="#7C908000" size="1081344"
filename_hash="c88d57cc99f75cd928b47b6e444231f26670138f"/>
```

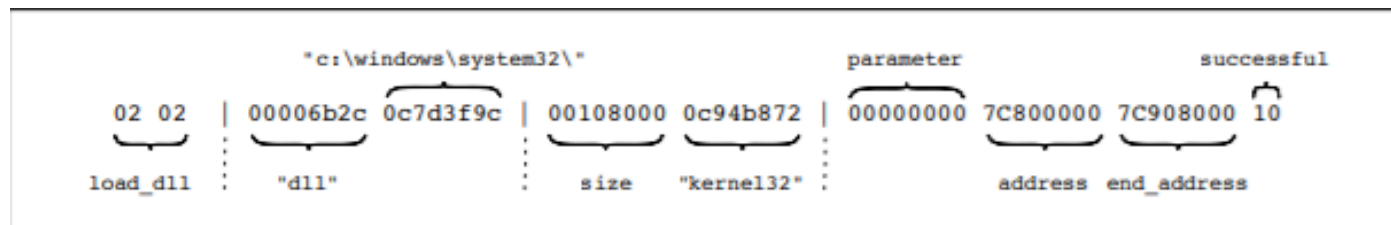


Figure 4.8 : Understanding a MIST Instruction with the help of an example.

Level -2: This level contains constant information, say file extension and file path.

Level -3: This level contains information which varies often. For example, file size and file name are values which differ even when two variants of the same program are considered. So this information is stored here.

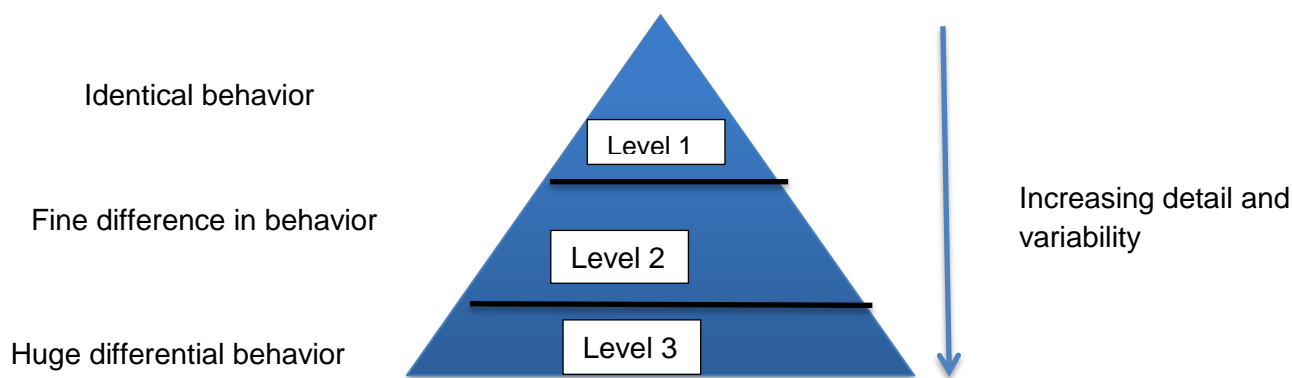


Figure 4.9 MIST Levels

4.4.2.4 MIST reports converted into N-Gram data

Unstructured Information: Random Forest and Tree Classifier algorithms would normally work with structured data. However, if you have unstructured data, for eg. in our case, we have JSON reports generated by Cuckoo sandbox. Now this data is in the text form and is unstructured as per the requirements of most of the classifier algorithms. Therefore for such unstructured data, we go in for N-Gram approach of classification.

MIST instructions give a feature representation report for each binary. There are typical behavior patterns of malware like changing registry keys or modifying system files. These behaviors are represented in a particular sequence in these reports and thus these are very useful for malware detection.

However for analysis techniques of machine learning, this is not a suitable format. We need to operate on vectors of real numbers[54]. So we apply a technique called n-gram to convert MIST report into a vector space.

N-Gram approach- In our model we are using a 1-gram approach. Say we have 10 sentences. We break all these sentences into single words. We remove duplicates and find out all unique words. These are called tokens. Then we check the frequency of words in the sentence. This gives us the count of unique tokens. Now say we have to find similar sentences in some other

file. If the same tokens occur in the other file also, we find the similarity index between these two files. This is the basic concept of n-grams.

The same concept is used in our model also. Say we have some 10 malicious files. Let's assume that a 'crypt API' call of code 11007 is being called in all these 10 files. Then according to the similarity with API calls, we can deduce that a file calling this particular 11007 API call can be a malicious file.

N-gram approach is based on the vector space and bag-of-words model. It finds shared behavior patterns. For example, function calls can be extracted from each analysis report and a sparse matrix can be generated.

Say we represent a feature set by S and a set of all behavior analysis reports generated by MIST as M [55]. If there is a word $s \in S$ and a report $m \in M$, we have to calculate number of occurrences n in m to calculate frequency

$$f=(m,s)$$

4.4.2.5 Sparse matrix Generated

N-gram breaks all available text into tokens. How we convert text into tokens depends upon our application. In our case, we are using 1-gram. After extraction of tokens, n-gram converts these tokens into sparse matrix. A sparse matrix is just the count of these tokens per file.

Bag of Words feature extraction technique: This technique represents a string as a vector of token frequencies. For example- let's take a sentence:

“The pencil is sharpened by the sharpener”

This is a string. It is represented as a vector:

{the:2, pencil:1, is:1, sharpened:1, by:1, sharpener:1}

Bag of words technique determines a token count for each string in isolation. Vectorization assigns a unique index for each token observed in the dataset as a whole. For example:

File 1: a, x, c, d, a

File 2: b, a

Corpus: a, b, c, d, x

Sparse Matrix: [[2, 0, 1, 1, 1]

[1, 1, 0, 0, 0]]

This matrix is created with respect to the corpus. Corpus is every unique word in every file in the directory.

4.4.2.6 KNN Classification Algorithm applied

KNN is a machine learning classification algorithm which is:

- Simple
- Non-parametric
- Lazy
- Based on feature similarity

Simple: KNN is a simple whose purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point.

Non-parametric: KNN does not make any assumptions on the underlying data distribution. Therefore in many real world problems for KNN can be used for classification if there is little or no prior knowledge about the distribution data.

Lazy: KNN is not an eager algorithm. It does not use the training data points to do any generalization. There is no explicit training phase or it is very minimal. This also means that the training phase is very fast.

Based on Feature Similarity: KNN Algorithm is based on feature similarity. Feature similarity means how closely out-of-sample features resemble our training set determines how we classify a given data point.

KNN can be used for classification—the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors.

4.4.3 Building Network Part of PosDeF

For creating the network part of the framework, we use a tool called Snort. Snort is an open tool used for network traffic analysis. It combines signature, protocol and anomaly inspection. It is both an IDS and an IPS. It can detect many variety of attacks and probes like :

- 1) Buffer overflows
- 2) Stealth port scans
- 3) CGI attacks
- 4) SMB probes
- 5) OS fingerprinting attempts etc.

Snort allows creating rules for detecting malicious traffic and alerting the user. These rules are used by SNORT functions for performing protocol analysis, content searching and content matching on network traffic[56].

Policies-When these rules are implemented, these implementations are called policies.

Alerts- Snort raises alert in real time if a rule matches the content of a payload in the traffic.

Snort rule-



Snort rule header-



alert UDP 192.168.0.2 any -> any 80 (msg : “Twitter” ; content: twitter.com ; sid:20003)

alert: action

UDP: protocol

192.168.0.2: source IP

Any: source port

Any: destination IP

80: destination port

Msg: rule options

This rule header contains information which can send an alert for a packet in a UDP stream whose source IP address is 192.168.0.2 towards any destination IP, having port number 80. Rule Options tell the rule name and rule id and also the context that needs to be detected. Snort will take the associated action when the rule header and rule options match the content of the packet. In this example, if in the traffic a packet from the UDP stream contains the twitter server name an alert will be generated.

4.4.3.1 Cuckoo generates dump.pcap file

PCAP stands for Packet Capture. PCAP files are created by Cuckoo when they dynamically analyze any file as dump.pcap file and they contain network data which is created during a live network capture [57]. These files are used for packet sniffing and analyzing the network traffic. PCAP has an API for capturing network traffic from ports. It also monitors IP address and other network related parameters. Protocols are also investigated.

4.4.3.2 Snort generates text file out of pcap file

Snort uses this dump.pcap file and converts it into a text file. Actually dump.pcap is a binary, non-human readable file. Snort converts it into a readable format consisting of alert messages. A tcpdump tool is used for this. Snort is supporting IPV6 format too.

```
tcpdump -n -ttt -r /snortLogFilePath/snortLogFileName > /pathWhereToStore/File.txt
```

-r reads a single pcap file.

A snort.conf file is the default configuration file which contains all configuration rules which are to be matched. All rule sets are defined in it.

An alert file is generated from the pcap file. Below is a snapshot of such a file.

```
[**] [1:2010935:2] ET POLICY Suspicious inbound to MSSQL port 1543 [**]  
[Classification: Potentially Bad Traffic] [Priority: 3]  
06/20-20:31:31.817215 232.10.237.105:2000 -> 230.29.20.24:1543  
TCP TTL:102 TOS:0x0 ID:260 IpLen:20 DgmLen:40  
*****S* Seq: 0x43EE0000 Ack: 0x0 Win: 0x4000 TcpLen: 20  
[Xref =>http://doc.emergingthreats.net/2010935]
```

4.4.3.3 Text file converted into MIST format

This text files ha to be converted into the MIST format. MIST format is machine learning compatible format and is required if we want to give this data to various machine learning algorithms.

4.4.3.4 MIST file converted into sparse matrix

The MIST file again has to be converted to N Gram token sparse matrix for easy analysis. Relevant tokens are identified and a sparse matrix is created so that machine learning algorithms find these matrices efficient to operate upon.

4.4.3.5 KNN applied to sparse matrix

K Nearest Neighbor is the classification algorithm applied to this sparse matrix. When we compared it to other algorithms, it was giving an accuracy of 80.7%. Therefore this was the best algorithm to be applied with the current data. Also according to its confusion matrix, it gave the least number of false positives and false negatives.

Confusion Matrix- A confusion matrix is a table which describes the performance of a classification model based upon a set of data for which we already know the true values. For example:

	Predicted: NO	Predicted: YES	
Actual: NO	78 TN	5 FP	83
Actual: YES	3 FN	200 TP	203
	81	205	

Figure 4.10A Sample Confusion Matrix

True Positive- Say in case of testing of files, the number of true positives was 200. What that means is that out of 286 files, 200 were predicted by the model as malign and they were actually malign.

True Negative- 78 predicted as benign were actually benign and not malicious.

False Positive (Type1 error)- 5 predicted as malicious but were actually non-malicious i.e. benign.

False Negative (Type2 error)- 3 predicted by model as non-malicious but were actually malicious.

Accuracy-Means how often is the classifier correct.

$$(TP+TN)/Total=(200+78)/286=0.972$$

That means our model has a 97.2% accuracy.

4.4.4 BuildingSandbox Evasion Detection part of PosDeF

A very powerful technique developed by antivirus and IDPS creators is sandbox technique. Using this technique, the suspicious programs can be run in a simulated environment called a sandbox. All programs, whether malicious or non-malicious would show their true nature in this environment.

Thus the anti-malware program would be easily able to detect the malicious programs and weed them out of the system. However, attackers have come up with sandbox evasion techniques. They employ various methods to detect whether they are working in a simulated environment or the actual operating system. Accordingly, if they detect a sandbox, they would lie dormant or do very superfluous jobs. It's only when they are sure they are running in the actual operating system, would they show their true working and would launch an appropriate attack.

In our model, we would find out whether sandbox is being evaded by the sample or not. Accordingly, a probability of 1 or 0 would be allocated if the sandbox is being evaded or not respectively. This calculated value will be used in the final calculation of maliciousness probability of the sample. A JSON file generated by Cuckoo has a sandbox evasion parameter. It has a Boolean value of 0 or 1. By parsing this JSON file one can find out whether the sample tried to evade the sandbox or not. If it tried to evade the sandbox, there is a 100% chance that it can be a malware.

4.5 Distributive Execution of PosDeF

Over the last decade, with the advent of advanced malware which change their behavior and structure at runtime, signature detection is no longer valid. For such polymorphic and metamorphic malwares, one needs complex detection involving static analysis, dynamic analysis, network analysis and sandbox evasion detection. Using machine learning for this process is the only viable solution. However, this complete processing also requires large computing power and resources.

Distribution of processing is thus required to achieve an optimized solution. In this paper we have proposed Apache Spark as a distribution framework for machine learning malware detection. We have used Amazon AWS as a platform for execution. Blockchain is used for providing an immutable dataset of malware information to all nodes participating in the

distributed malware detection system. A comparative study is also conducted between performance of non-distributive analysis and distributive analysis.

4.5.1 Introduction

Malwares are as integral to the digital space as are normal applications. To counter every new type of malware is necessary for the proper working of all applications.

Malwares are evolving day by day and antivirus companies are finding it difficult to keep up the pace. It is imperative that a check is put on malicious activities. The biggest threat, cyber world is facing today is that of 'Polymorphic and metamorphic malwares'. These are advanced malwares which change their look (signature) but have the same behavior. In the case of metamorphic malwares, they are capable of changing their signature as well as behavior at runtime. This makes them difficult to detect by Signature based detection systems. Thus Anomaly based detection systems also fail here.

All the current solutions existing today have some lacunas in common:

1. Signature Matching-Most of them focus on malware detection by signature matching and pattern recognition. Malware authors are now smarter than ever before and signature detection is of no use due to techniques like polymorphism, metamorphism etc.
 2. No behavioral analysis- Some of them do not take into consideration, file behavior, network behavior and other dynamic behavior of the file to be analyzed.
 3. Updating time-Another problem that current antiviruses face is that they take a lot of time to analyze the malware and then update the definition of antiviruses into user's device.
 4. Danger to user's privacy- The antivirus companies are sometimes a danger to the user's privacy as they collect data from user on regular basis and use that to make money. The normal user acts only as a data feeder so that these antivirus companies can protect enterprises.
-

5. Centralized Antivirus Companies-All the antivirus organizations are centralized in nature. This implies that a lot of computation power is required and very few computers are available to provide it. If a distributed system could be designed where all systems in the network contribute to the work of malware detection, things will become faster and more efficient.

Note that the recent ransomware attacks by WannaCry and Petya can prove the above statements, not to mention that none of them were actually polymorphic or metamorphic.

4.5.2 Proposed Distributive PosDeF

Our proposed framework will take care of all kinds of malwares including polymorphic and metamorphic. We are analyzing static behavior, dynamic behavior, network behavior and sandbox evasion behavior. Machine learning is incorporated at every step to make the work more efficient and detect any new malware if current antivirus solutions label it as clean. Decentralized currency was another incredible innovation recently and it led us to the new system of bitcoins. Bitcoins are great but what makes them greater is the technology on which it works. The technology is known as blockchain and it has so many other applications other than just decentralized currency. Here we use this concept to harness distributive power of all the systems in the network so that a large amount of computing power is gained with very little cost and everyone in the network is benefitted.

Our work is highly significant both in the present and in the future. Hackers have recently used shellcodes in WannaCry and Petya Malwares. The next step is to use polymorphic and shellcode attack. We must be ready for them and our works makes us ready.

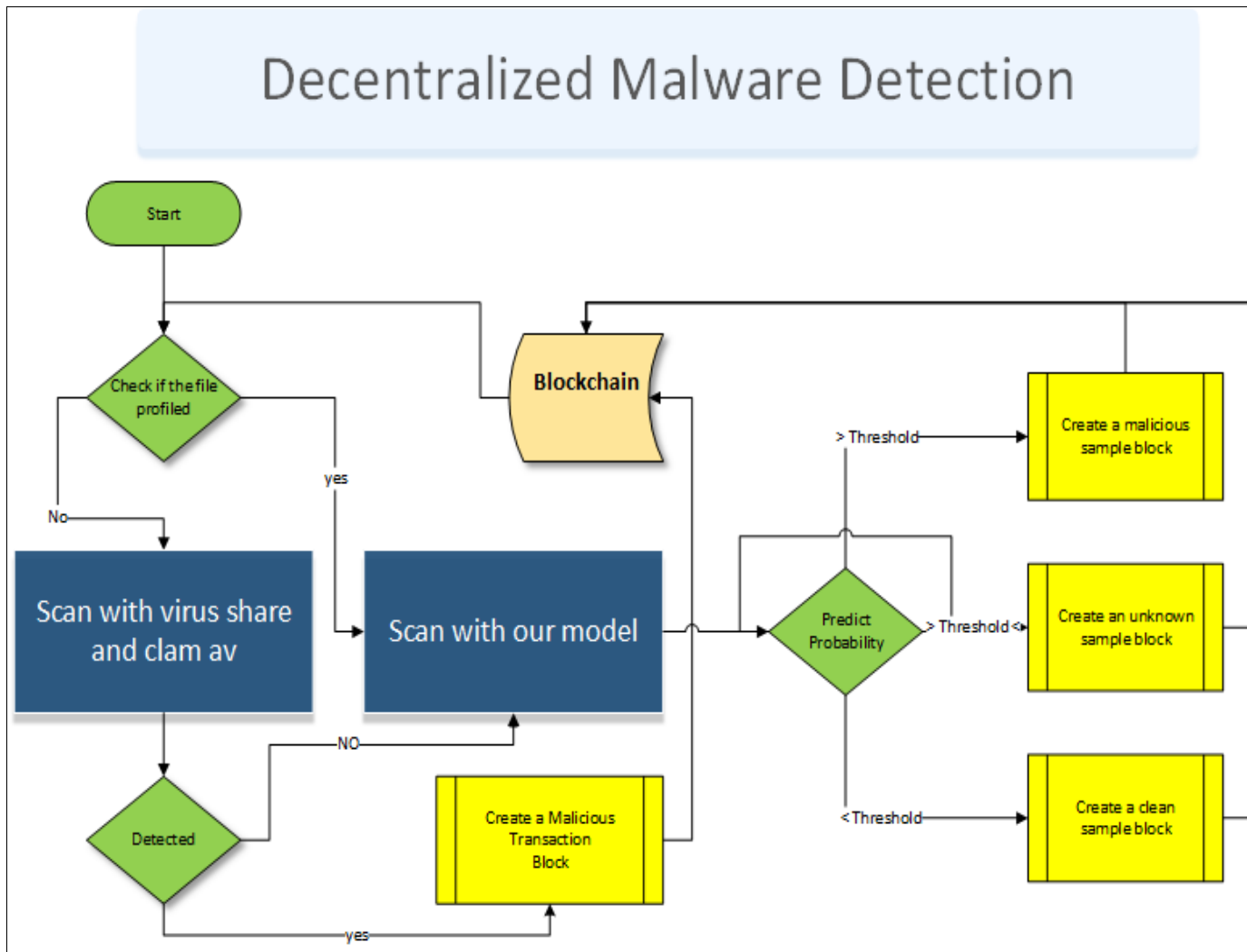


Figure 4.11 Proposed fraework for Decentralized Malware Detection.

Let's understand how the framework works. When a file which is to be tested enters the system, we have to profile the file. We check it using already existing antivirus solutions like Clam AV and Yara, which use VirusShare database. All these technologies are open source. If the file is detected by the antiviruses as malicious, we straight away label it as a malware and log it in the database as not-clean. However, if the file is claimed to be clean by the antiviruses, we do not completely believe them and run it through our model.

Now after scanning about one lakh files from a balanced dataset of malicious and clean files, the model (through machine learning algorithms), automatically finds out the feature set which distinguishes between clean and malicious files. We calculate a threshold value for maliciousness in a file in the form of probability. Now we calculate this probability in our test

sample. If it is more than the threshold value, we label it as malicious. If it is less than the threshold value, we label it as clean. If it is equal to the threshold value, we label it as unknown and make it run through the model again for reanalysis. The results are stored in the form of an immutable ledger called Blockchain.

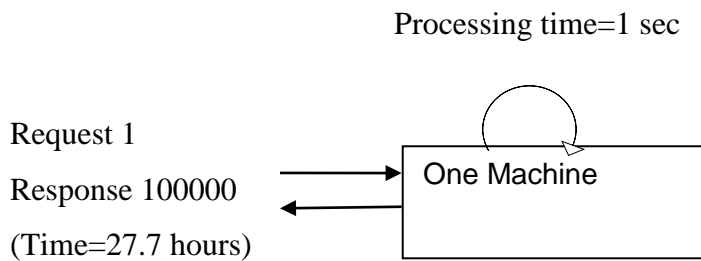
Our framework uses various machine learning algorithms of clustering and classification like KNN, Random Forest, N-Gram, Tree classifier etc. Also for the training phase a huge dataset is required for the model to become viable and give correct results. The model is iterative so that it improves as more and more samples enter the system even during testing phase. Thus a great amount of computing power and data storage is required to run this model. Therefore, running on a single node is not an effective solution.

We use Amazon AWS Cloud services to run the framework in a distributed environment. Apache Spark framework is used for distributing the load between participating nodes. Blockchain not only maintains details about clean and malicious files on individual nodes but is also used for rewarding users according to the amount of computing power they donate for the detection model. This reward is in the form of tokens mined by the users.

4.5.3 Centralized versus Distributive Computing

There is a huge performance benefit that can be achieved using distributed computing [58]. Let's take an example. Say we have 10,000 requests to be served. The processing time per request is say 1 second. Therefore, after the first response, the last response will come after $100000/3600 = 27.7$ hours. However, if we have 1000 nodes working in parallel, the same response is going to come in $100000/6000 = 1.6$ minutes. This is a great performance benefit.

Centralized System



Distributive System

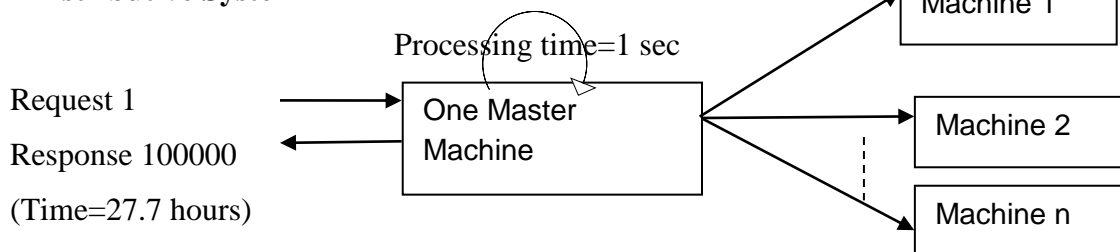


Figure 4.12 Comparison of performance between a centralized and a distributive system

4.5.4 Role of Apache Spark in PosDeF

Apache Spark is a framework used for cluster or distributed computing. It is an open source framework [59]. Since our model will run on all networked nodes, Spark will make them run parallel and will also give fault-tolerance. Therefore, even in the case of failure of any node, the system is not going to breakdown and will continue operating properly. Spark gives to its users speed, easy usage and complex analytics.

For distributive computing, we had many technologies like Hadoop, MapReduce and Storm. Now Hadoop normally runs in batch mode and cannot handle real time data [60]. Apache Spark, however gives us real-time data analysis in a distributive environment. The other advantage of Spark is that it gives a unified framework for a variety of datasets like text dataset, graph dataset etc. It also supports data from both real time and batch sources [61].

Spark also enables one to write applications in a variety of languages like Java, Scala and Python. Also, in addition to Map Reduce, it also lets a user use SQL queries, streaming data, machine learning and graph data processing. In an experiment done by the company 'DataBricks' [62] in October 2014, a new world record was set for sorting 100 TB of data by Apache Spark on 207 EC2 virtual machines in 23 minutes. Apache Spark runs everywhere on Hadoop Mesos, Kubernetes, standalone or in the cloud. It can work with diverse data sources like HDFS, Cassandra, HBase and S3 [63].

In our model, after testing it on a single node, we need to test it on thousands of nodes because we will have petabytes of data. For our testing purpose, we have leveraged AWS M3 RAM medium instance that contains 3.75 GB, 1 core CPU and 410 GB of SSD. We have stored our data in HDFS which is designed for large storage only.

4.5.4.1 AWS M3- M3 is an instance of AWS. M3 is a second generation, general-purpose EC2 instance type[64]. We are using a medium M3 instance.

4.5.4.2 M3 medium- It has a RAM of 3.75 GiB with an instant storage (SSD) of 1X4 GB cache. M3 instances have high frequency Intel Xeon E5-2670(Sandy Bridge or Ivy Bridge) processors. M3 medium is a general purpose instance family. It has only one vCPU.

4.5.4.3 Amazon EC2- EC2 is an Elastic Compute Cloud. It is a web service that gives secure, resizable compute capacity on the cloud. It is designed to make web-scale cloud computing easier for developers. New service instances can be added or removed within minutes to scale both up and down as one's computing requirements change.

Amazon EC2 gives users a wide choice of instance types to suit their needs. Instance types give various combinations of CPU, memory, storage and networking capacity to give one the right mix of resources to choose from. Each instance type includes one or more instance sizes to scale your resources as per your workload[65].

4.5.4.4 SSD- Solid State Drives. These deliver high random IO performance. SSD is used as memory for persistent storage of data. SSD uses interfaces compatible with traditional IO and Hard Disk Drives which help in simple replacement in many applications. In our model, 410 GB of SSD is used.

4.5.4.5 HDFS- Hadoop Distributed File System. This is a fault-tolerant system and is designed to be deployed on low cost hardware. It is specifically suitable for large data sets. Since our model will handle data in petabytes, HDFS is suitable for it. It can scale to hundreds of nodes in a single cluster. Hadoop is actually a data warehousing system and it needs MapReduce to process the data.

Spark is designed to write and read data from and to HDFS as well as from other storage systems like HBase and Amazon's S3.

4.5.4.6 HDFS Architecture- HDFS has a master-slave architecture [66]. An HDFS cluster has one NameNode which is a master server managing the file system namespace. It also regulates access to files by the clients. Also there are a number of DataNodes which are one per node in the cluster which store data. A file is split into one or more blocks and these are stored in DataNodes. The NameNode manages opening, closing and renaming of files and directories. It also tells about the mapping of blocks to DataNodes. The DataNodes fulfill the request to read and write from the clients. The DataNodes are also involved in block creation, deletion and block replication.

4.5.4.7 Non-Computational Data Locality- We need processing and data storage in the same node. With databases, we had computational and data locality. What that means is that we had data in one machine and processing in the other. So normally, data is stored on multiple machines but processing is done on the server. This results in huge amount of data transfer. Thus traditional MapReduce is not successful for efficient data sharing. Basically, Hadoop is used for data storage. It is not successful for analysis. However, Spark increases speed of analysis in real time.

4.5.4.8 Data and Processing on each machine- In a Hadoop cluster, data which is in the form of HDFS and the MapReduce system, both are present in each machine in

the cluster. This has two advantages; first it introduces redundancy in the system. Therefore, fault-tolerance increases. Also since data and data processing software resides on the same machine, therefore information retrieval speed increases.

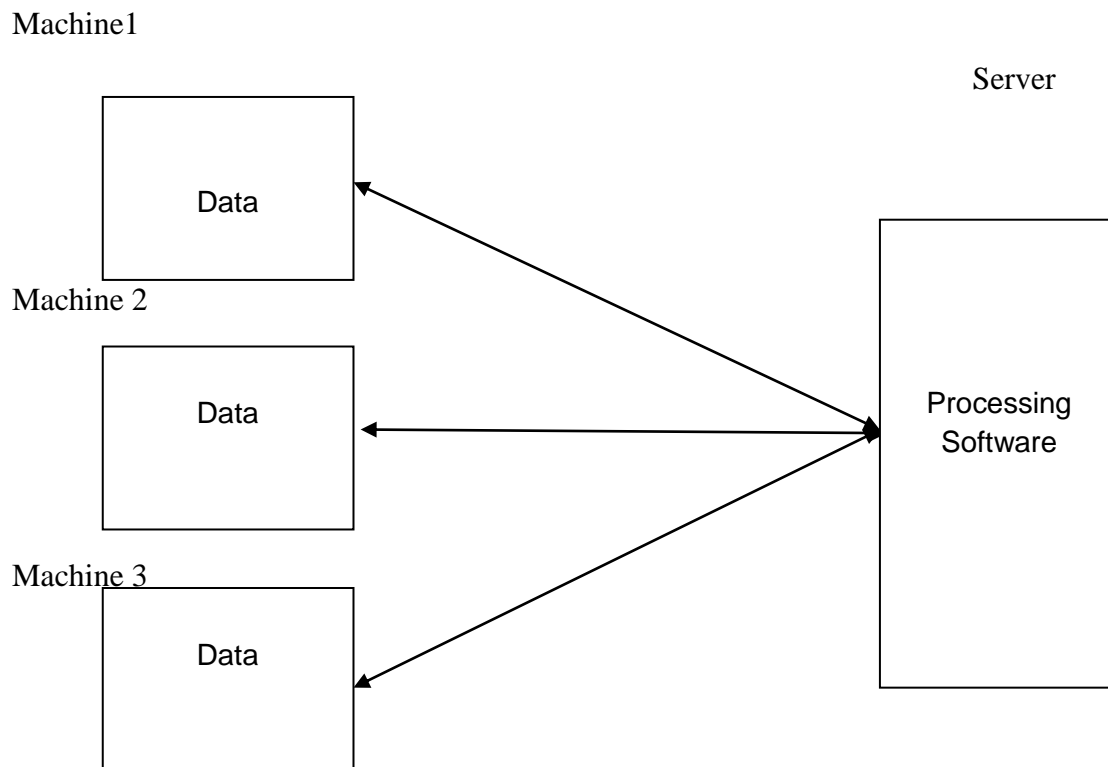


Fig 4.13 Traditional Database Processing Structure

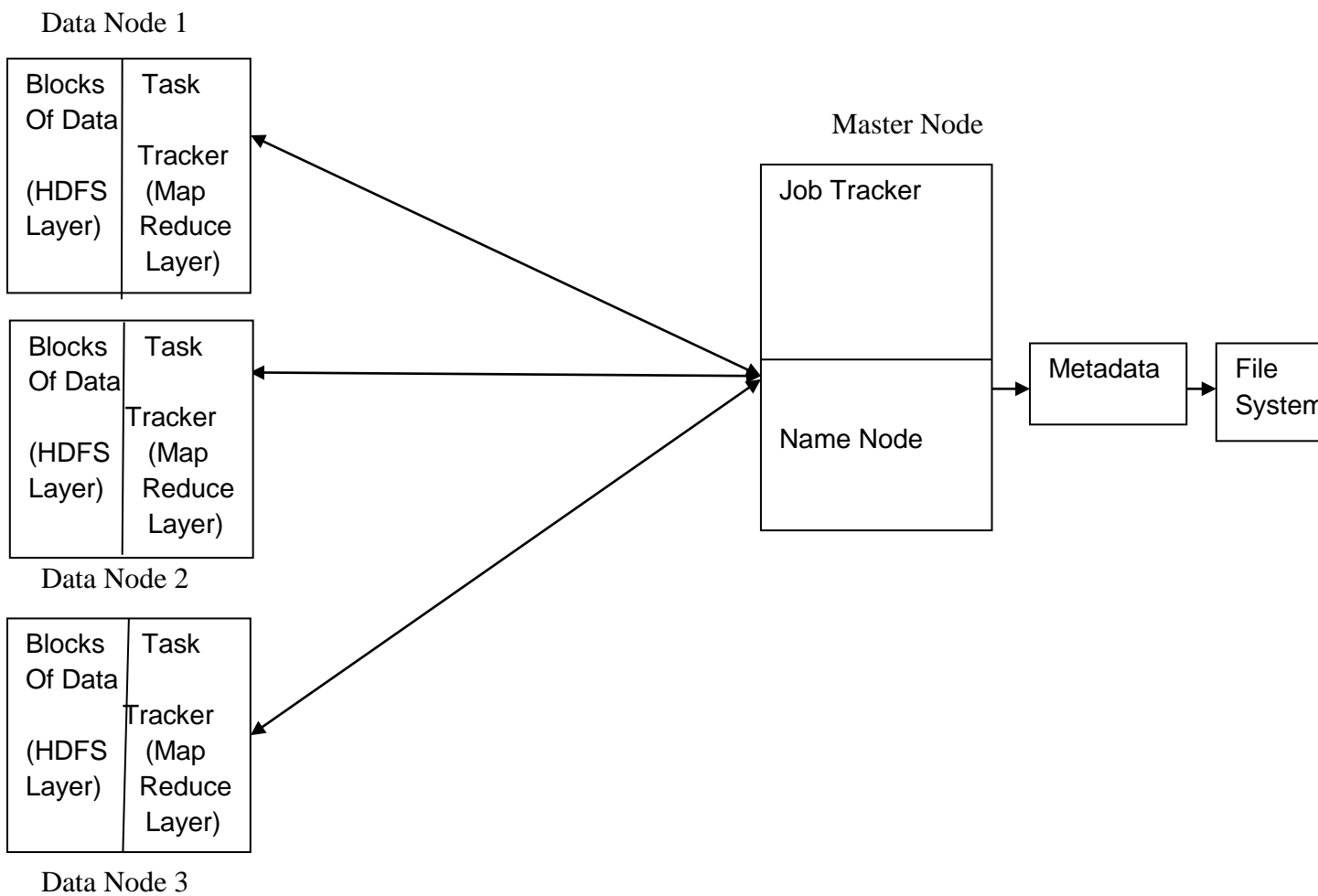


Figure 4.14 Hadoop Cluster

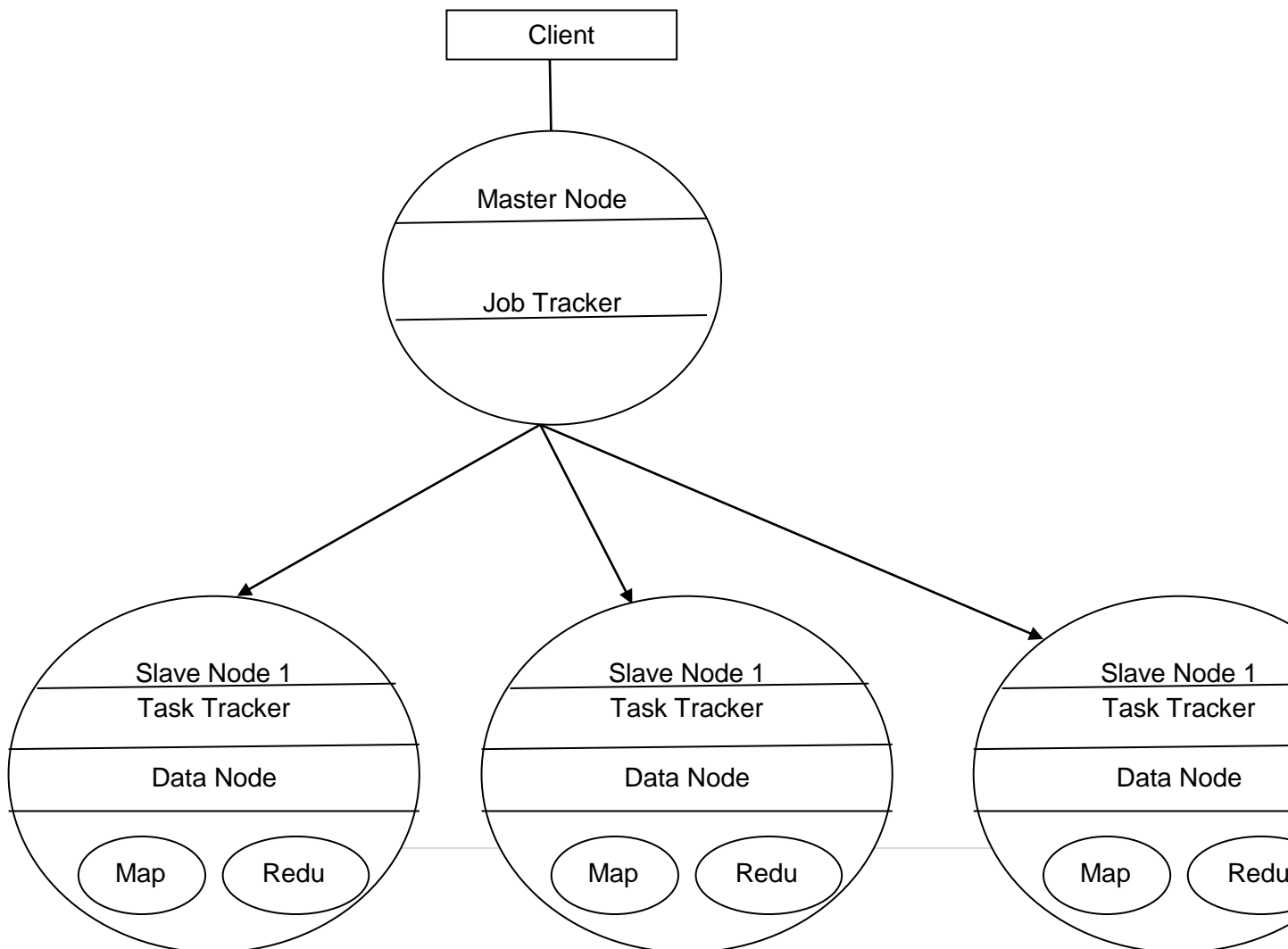
4.5.5 Working of the Hadoop Cluster

The Hadoop cluster works in a modular fashion and the work is divided between master and slave nodes. Detailed working is as follows:

- 1) Start.
- 2) Client requests for a file.
- 3) JobTracker receives the request.
- 4) NameNode which stores metadata tells which DataNodes store the blocks that make up that file.
- 5) Client directly reads the blocks from the individual DataNodes.
- 6) JobTracker schedules the Map and Reduce tasks on the appropriate TaskTrackers in a rack-aware manner.
- 7) JobTracker monitors for any failing tasks that need to be rescheduled on a different TaskTracker.
- 8) TaskTracker spawns JVMs to run Map and Reduce tasks and report back to the TaskTracker.

Stop after the process completes.

Figure 4.15 Hadoop Master-Slave Architecture



CHAPTER 5 FINDINGS AND CONCLUSIONS

5.1 Execution for a clean sample on the Framework.....	155
5.1.1 Static Analysis.....	155
5.1.2 Dynamic Analysis.....	155
5.1.3 Snort Analysis.....	161
5.2 Execution for a malicious sample on the Framework.....	162
5.2.1 Static Analysis.....	162
5.2.2 Dynamic Analysis.....	163
5.2.3 Snort Analysis.....	167
5.2.4 Sandbox Evasion Detection.....	170
5.3 Final Results.....	170
5.3.1 Results obtained from the Framework after all four steps.....	171
5.3.2 Comparison with the existing Centralized systems.....	175
5.3.2.1 Working of the Distributed Framework.....	177
5.3.2.2 Blockchain for Records and Rewards.....	177
5.3.2.3 Using Ethereum for maintaining Blockchains.....	180
5.4 Conclusions.....	182
5.5 Limitations of the Study.....	182
5.6 Further Research Potential.....	183

CHAPTER 5

FINDINGS AND CONCLUSIONS

After successfully training the model, we will test the model with about 40 files. Below first we will show the screenshots of actual execution first with a clean file and then with a malicious file. Then we show an excel document of the complete results with a dataset of 40 files.

5.1 Execution for a clean sample on the Framework:

Say we have a file named '1.exe'.

5.1.1 Static Analysis

First we do static analysis for the file. When we get the result, the static analysis gives a probability of only 26% that this file is a malicious file.

On the command prompt we can write:

```
$ ./checkpe.py ../finalresult/clean/1.exe
```

The result that we get is:

```
[0.26 0.74]
```

5.1.2 Dynamic Analysis

This is the dynamic analysis for the file. Dynamic analysis or behavior analysis is done using 'Cuckoo Sandbox'. After setting up Cuckoo on Ubuntu host we will start submitting samples. Submitting of sample can be done using simple command:

```
$ cuckoo submit /path/to/binary
```

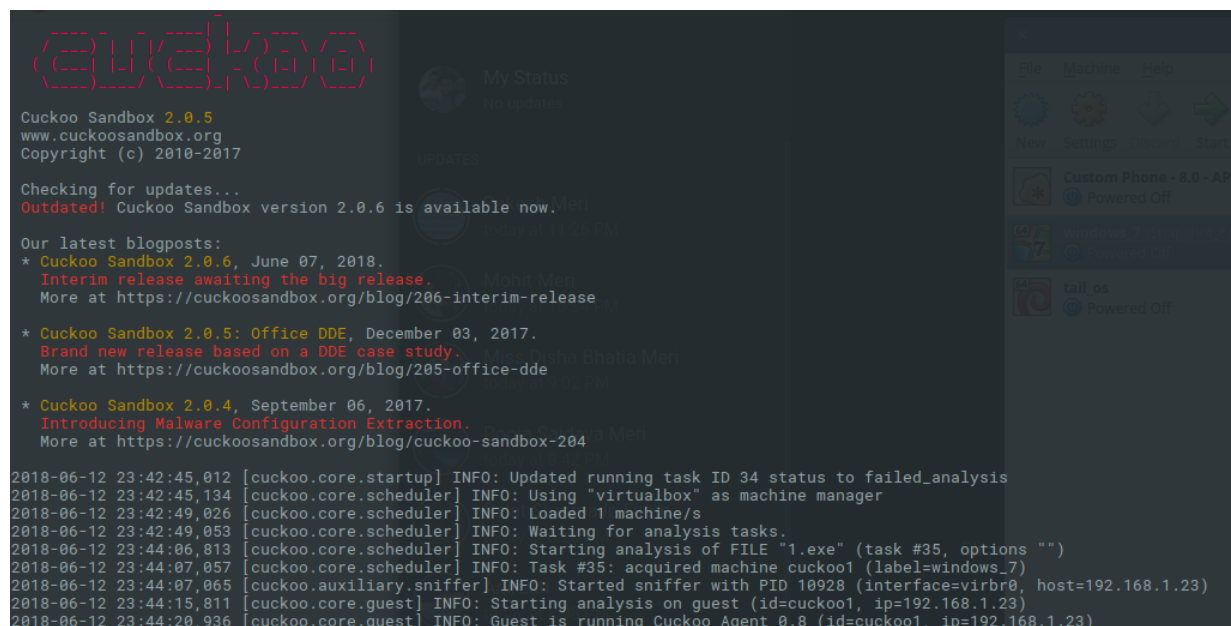
Because it would be cumbersome to go and submit every file manually here is a small python script that makes the work very simple.

```
Import os
from subprocess import call
directory_in_str="/home/varnit/Downloads/clean_files";
directory = os.fsencode(directory_in_str)
for file in os.listdir(directory):
    filename = os.fsdecode(file)
    call(["cuckoo", "submit", "/home/varnit/Downloads/clean_files/"+filename])
```

Figure 5.1 Python Script for submitting files to Cuckoo

Above script will fetch every other file into clean directory and submit it to cuckoo. There is a similar directory named malicious and we can run the same script for that as well.

This is how Cuckoo runs and analyzes the file.



```
Cuckoo Sandbox 2.0.5
www.cuckoosandbox.org
Copyright (c) 2010-2017

Checking for updates...
Outdated! Cuckoo Sandbox version 2.0.6 is available now.

Our latest blogposts:
* Cuckoo Sandbox 2.0.6, June 07, 2018.
  Interim release awaiting the big release.
  More at https://cuckoosandbox.org/blog/206-interim-release

* Cuckoo Sandbox 2.0.5: Office DDE, December 03, 2017.
  Brand new release based on a DDE case study.
  More at https://cuckoosandbox.org/blog/205-office-dde

* Cuckoo Sandbox 2.0.4, September 06, 2017.
  Introducing Malware Configuration Extraction.
  More at https://cuckoosandbox.org/blog/cuckoo-sandbox-204

2018-06-12 23:42:45,012 [cuckoo.core.startup] INFO: Updated running task ID 34 status to failed_analysis
2018-06-12 23:42:45,134 [cuckoo.core.scheduler] INFO: Using "virtualbox" as machine manager
2018-06-12 23:42:49,026 [cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2018-06-12 23:42:49,053 [cuckoo.core.scheduler] INFO: Waiting for analysis tasks.
2018-06-12 23:44:06,813 [cuckoo.core.scheduler] INFO: Starting analysis of FILE "1.exe" (task #35, options "")
2018-06-12 23:44:07,057 [cuckoo.core.scheduler] INFO: Task #35: acquired machine cuckoo1 (label=windows_7)
2018-06-12 23:44:07,065 [cuckoo.auxiliary.sniffer] INFO: Started sniffer with PID 10928 (interface=virbr0, host=192.168.1.23)
2018-06-12 23:44:15,811 [cuckoo.core.guest] INFO: Starting analysis on guest (id=cuckoo1, ip=192.168.1.23)
2018-06-12 23:44:20,936 [cuckoo.core.guest] INFO: Guest is running Cuckoo Agent 0.8 (id=cuckoo1, ip=192.168.1.23)
```

Figure 5.2 Analysis of files by Cuckoo

This screenshot shows Cuckoo processing file in the sandbox which is actually an Oracle VM VirtualBox here.

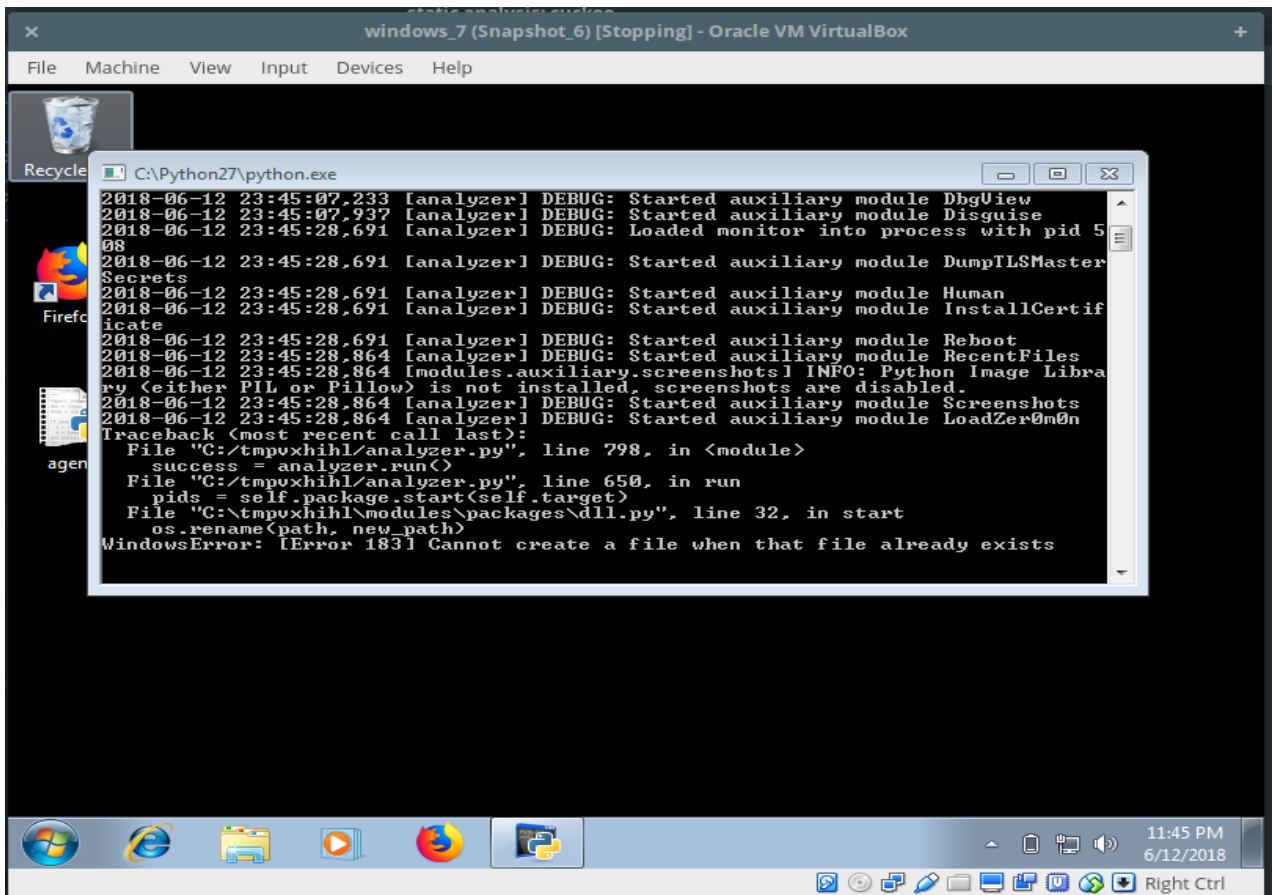


Figure 5.3 Cuckoo processing the file in a virtual sandbox

This is the JSON report generated for the file.

```
varnit@varnit-HP-EliteBook-840-G1:~/cuckoo/storage/analyses/latest$ cat reports/report.json
{
  "info": {
    "added": 1528847107.751837,
    "started": 1528847108.296863,
    "duration": 25,
    "ended": 1528847133.625968,
    "owner": null,
    "score": 0.0,
    "id": 36,
    "category": "file",
    "git": {
      "head": "59d32361c1636b2b3802a1746f480a7768f6384f",
      "fetch_head": "59d32361c1636b2b3802a1746f480a7768f6384f"
    },
    "monitor": "e19c4b4b529be2e90b3c5a3dfaad96f71c4fd54b",
    "package": "",
    "route": "none",
    "custom": null,
    "machine": {
      "status": "stopped",
      "name": "cuckoo1",
      "label": "windows_7",
      "manager": "VirtualBox",
      "started_on": "2018-06-12 23:45:10",
      "shutdown_on": "2018-06-12 23:45:32"
    },
    "platform": null,
    "version": "2.0.5",
    "options": ""
  },
  "signature": []
}
```

Figure 5.4 A JSON report for the file analyzed by Cuckoo

Now, this JSON file is human readable but it's a lengthy file and it will be very difficult for a machine learning algorithm to process it. A man called Konrad Rieck at the University of Göttingen created a method name Malheur which has created a new format called mist format. MIST format represents JSON files very efficiently. They have used it to classify malwares into different families i.e. multiclass classification but it's equally good for binary classification ie. malware or benign sample.

Now we need to convert this JSON format into MIST format to make it more understandable for machine learning algorithms.

```
# process 2768 thread 2772 #
06 06 | 000000e0
06 07 | 00344ce5 73330000 00000000 071ca90c
06 02 | 00000000 00344ce5 73330000 0084fb10 03337ae7
06 02 | 00000000 00344ce5 73330000 0084fb10 03337af1
06 02 | 00000000 00344ce5 73330000 0084fb10 0b33379e
06 06 | 000000f4
06 06 | 000000f0
06 06 | 00000044
06 08 | 040490bc 754d0000
06 02 | 00000000 04c59405 754d0000 772e9d35 0de60062
06 08 | 040490bc 754d0000
06 02 | 00000000 04c59405 754d0000 772e9d35 0de60062
06 06 | 000000ac
06 06 | 000000b0
06 06 | 000000a8
06 06 | 000000a4
06 06 | 0000009c
06 06 | 000000a0
06 06 | 00000098
06 06 | 00000078
06 06 | 00000070
06 06 | 00000064
06 06 | 00000068
06 06 | 0000006c
06 06 | 00000074
06 06 | 00000060
06 06 | 0000005c
09 12 | 0000005c 00020019 00b80535
06 06 | 0000005c
06 06 | 00000038
06 06 | 00000024
06 06 | 00000034
```

Figure 5.5. Conversion of JSON report to MIST format

Now though this MIST format is better than JSON format, however, it is still not appropriate for the machine learning algorithms. Therefore, the MIST format has to be converted into

sparse matrix format ('transformed_mat') for feeding into the machine learning algorithms. The 31 stored elements here show the 31 unique NGram tokens. We use 'vectorizer' which is an object provided by the Scikit-Learn library. The vectorizer object converts the MIST format file into sparse matrix. It splits the MIST file into unique tokens. It then assigns weight to each token proportional to the frequency in which it occurs in the MIST file. Then it creates a sparse matrix in which each row represents a document and each column represents a token.

A sample ngram sparse matrix which is made of 16 files is of dimension -

```
x.shape  
  
(16, 7255)
```

Figure 5.6 N Gram Sparse Matrix

As visible in the above screenshot our matrix is of 16 by 7255 dimensions.

```
x.toarray()  
  
array([[ 345,  0, 150, ...,  0,  33,  33],  
       [1693,  0, 309, ...,  0,  16,  16],  
       [ 162,  0,  0, ...,  0,  1,  1],  
       ...,  
       [ 872,  2,  54, ...,  0,  11,  11],  
       [1326,  2,  83, ...,  0,  11,  11],  
       [ 726,  2,  2, ...,  0,  11,  11]], dtype=int64)
```

Figure 5.7 A 16 X 7255 Dimension Sparse Matrix

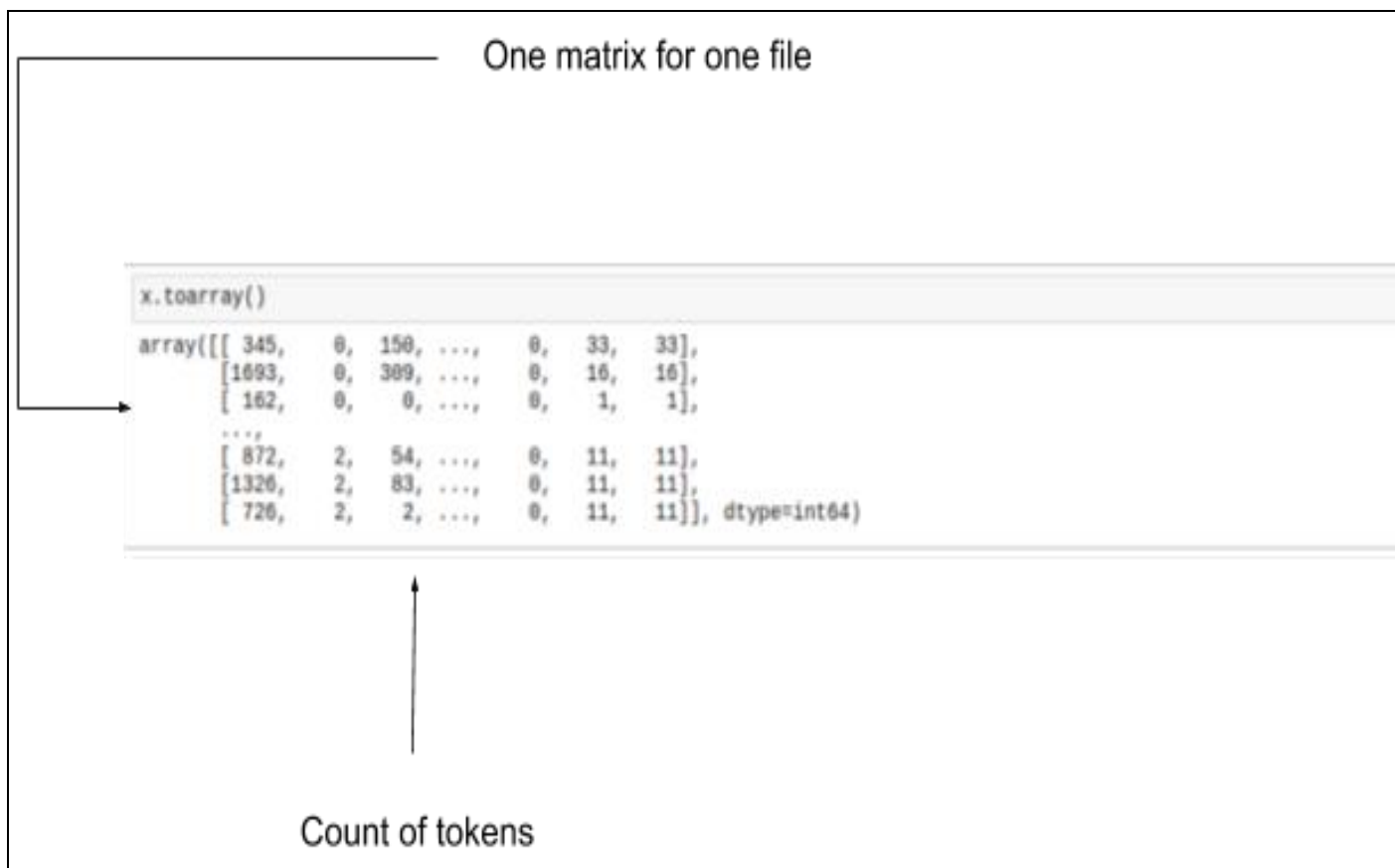


Figure 5.8 Explanation of a sample Sparse Matrix.

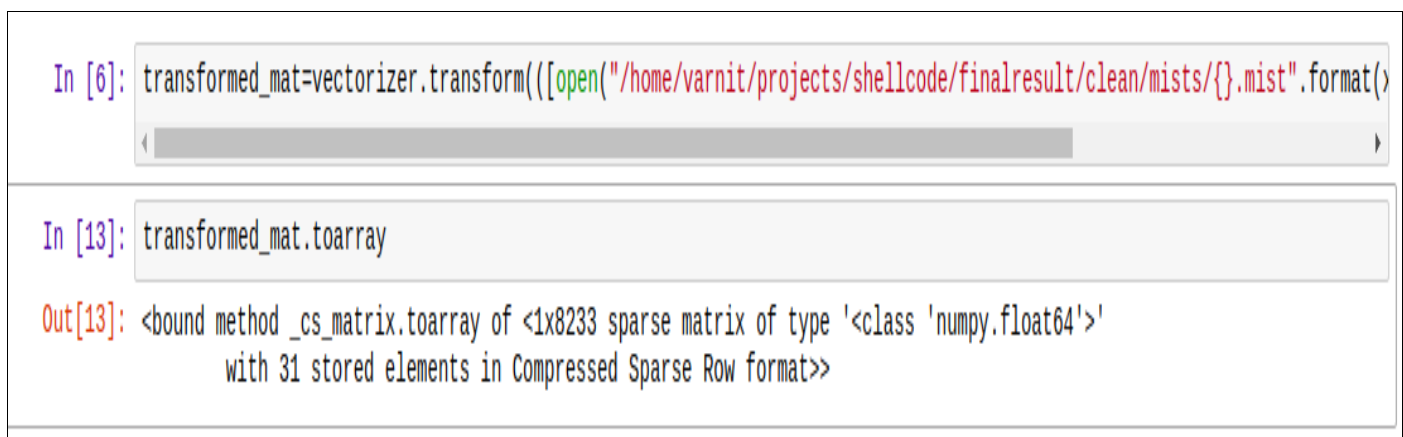


Figure 5.9 Commands to show the formation of Sparse Matrix for our data.

Various algorithms like KNN, Perceptron, Random Forest, Passive-Aggressive, L2 Penalty etc. are compared. Out of these, KNN showed the best accuracy.

The KNN (K Nearest Neighbor) machine learning algorithm is applied on the sparse matrix(transformed_mat) to predict probability of maliciousness. The result below shows that there is only 20% chance of the file being a malware.

```
In [14]: from sklearn.neighbors import KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=10)
         knn.fit(x,y)

Out[14]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                             metric_params=None, n_jobs=1, n_neighbors=10, p=2,
                             weights='uniform')

In [15]: knn.predict_proba(transformed_mat)

Out[15]: array([[0.2, 0.8]])
```

Figure 5.10 KNN algorithm used to predict probability on the transformed matrix.

5.1.3 Snort Analysis

A snort analysis done on the file gives its network characteristics. After submission to Cuckoo, a dump.pcap file is generated which gets converted into N Gram sparse matrix.

This is the screenshot of dump.pcap which is generated by Cuckoo.

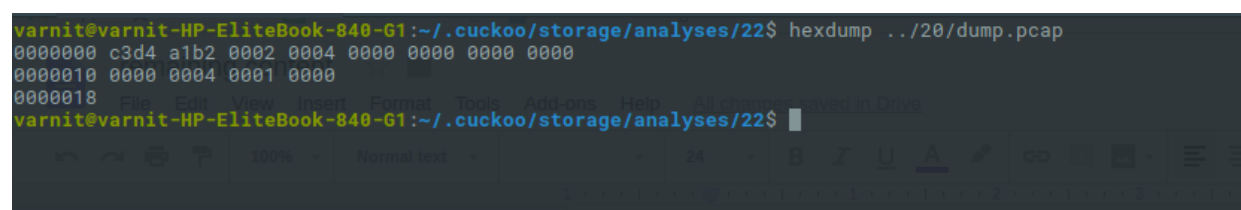


Figure 5.11 Snapshot of dump.pcap file

The sparse matrix is called 'transformed_mat'. Random Forest algorithm is used for predicting the probability of maliciousness of the file. It predicts that there is only 35% chance of the file being malicious. There is 65% chance of it being clean according to Snort analysis.


```
In [8]: from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier(n_estimators=100)
rf.fit(x,y);

In [9]: rf
Out[9]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
max_depth=None, max_features='auto', max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None,
min_samples_leaf=1, min_samples_split=2,
min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
oob_score=False, random_state=None, verbose=0,
warm_start=False)

In [11]: transformed_mat=vectorizer.transform([[open("/home/varnit/projects/shellcode/finalresult/clean/snort/{}.txt".format(x)
<
In [12]: transformed_mat
Out[12]: <1x204 sparse matrix of type '<class 'numpy.float64'>'
with 3 stored elements in Compressed Sparse Row format>

In [ ]: rf.predict_proba(transformed_mat)

In [14]:
array([0.35,0.65])

In [ ]:
```

Figure 5.12 Predicting probability of maliciousness using Random Forest Algorithm

The sample does not evade the sandbox. Therefore there is no sandbox evasion screenshot.

5.2 Execution for a malicious sample:

Now, when we run the model with a malicious file, it is easy to see a stark comparison.

5.2.1 Static Analysis

Firstly let's start with static analysis. In the training phase we will test various algorithms and again we split the set of data into 80% training and 20% testing. The graph below shows the result obtained from that. So we have found that Random Forest is the winner in classifying malwares based on static analysis.

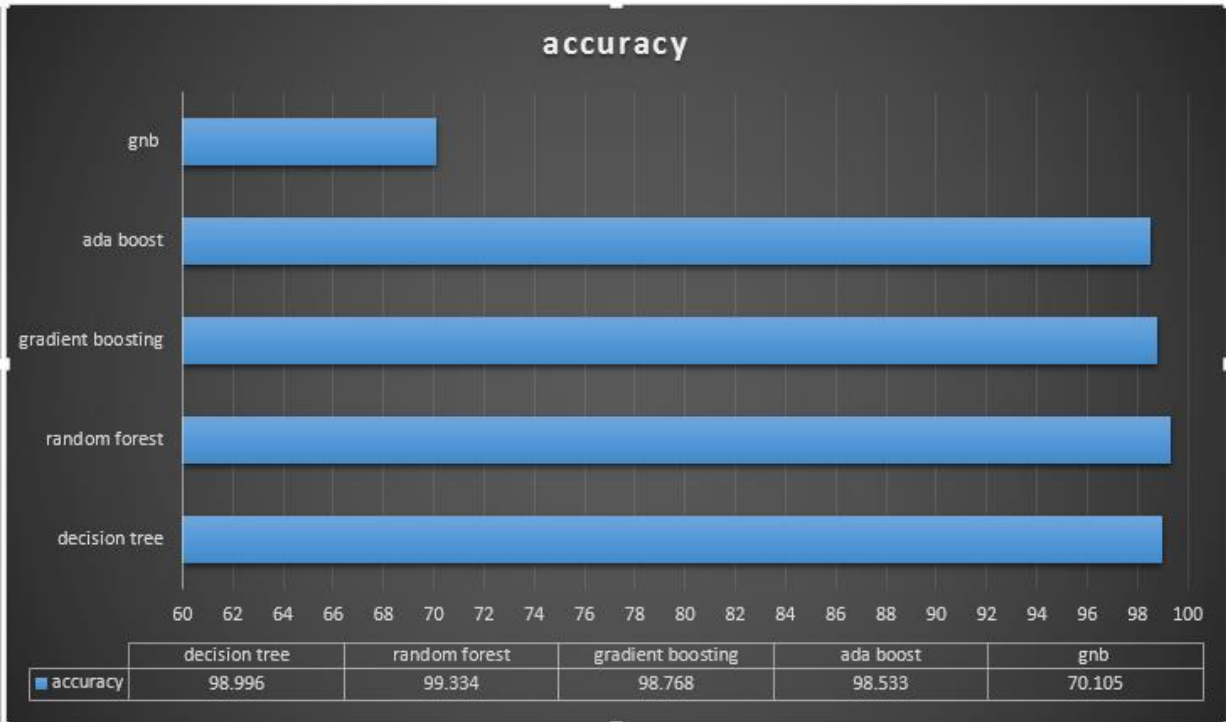


Figure 5.13 Comparison of various machine learning algorithms to show that Random Forest is the best

Below one can clearly see that static analysis gives a 1 probability that the file is malicious.

```
varnit@varnit-HP-EliteBook-840-G1:~/projects/shellcode/static analysis$ ./checkpe.py ../finalresult/malicious/6.exe
[1. 0.]
```

Figure 5.14 Result of a file showing malicious behavior

5.2.2 Dynamic Analysis

This is the dynamic analysis for the sample. Dynamic analysis or behavior analysis is done using ‘Cuckoo Sandbox’. This is how the sample is submitted to cuckoo for analysis.

```
varnit@varnit-HP-EliteBook-840-G1:~/projects/shellcode/finalresult/malicious$ cuckoo submit 6.exe
Success: File "/home/varnit/projects/shellcode/finalresult/malicious/6.exe" added as task with ID #37
```

Figure 5.15 Same file submitted to Cuckoo for dynamic analysis.

The sample runs in a sandbox which is actually an Oracle VM VirtualBox.

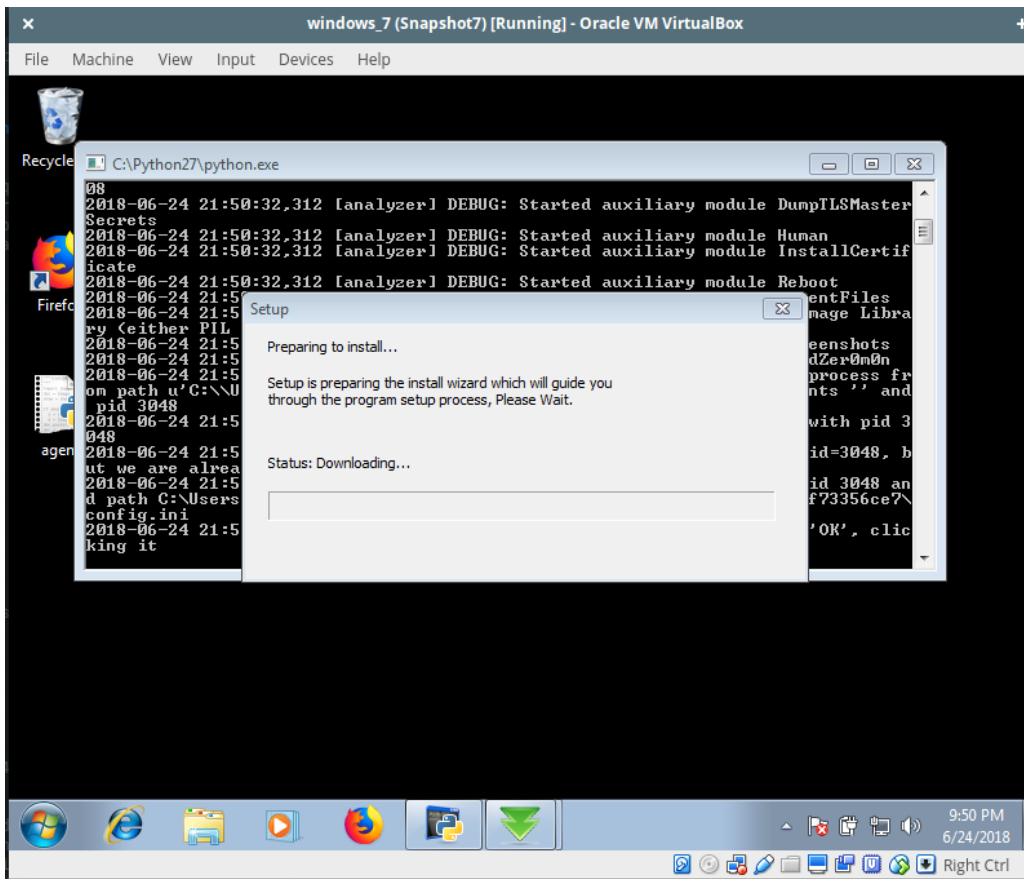


Figure 5.16 Sample running in a VM virtual box.

JSON report is generated for the sample.

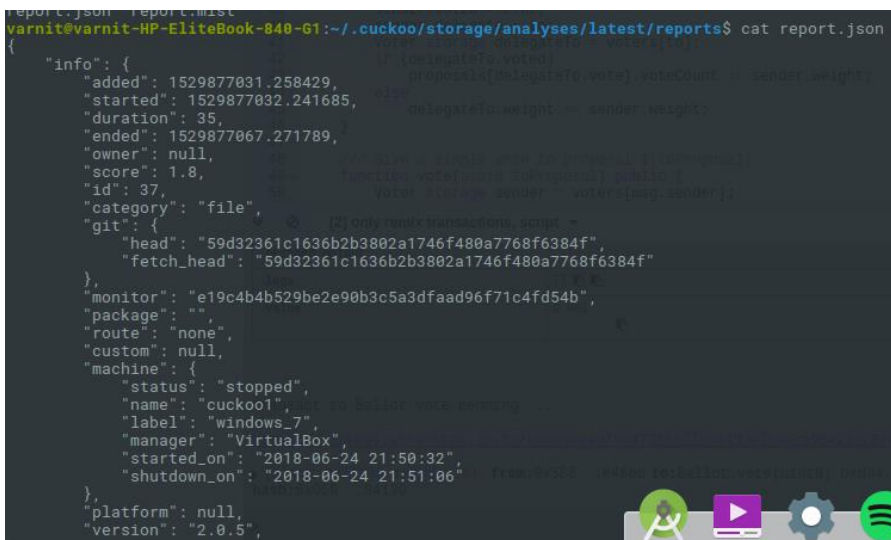


Figure 5.17 A JSON report generated for the sample

Now, this is the sample JSON file extracted from a malicious sample. We have omitted most of the file and only included those portions through which we can get some idea of how malicious behaviour can be predicted by the help of a JSON report.

Process Tree -> In the process tree it is visible that it's calling Internet Explorer. Now a malware may be trying to download some malicious payload in to the system or trying to inject itself into system through various processes like java applets, flash player or plain javascript.

antisandbox_sleep -> A process attempted to delay the analysis task.

antivm_virtualpc -> Tries to detect VirtualPC.

antisandbox_unhook ->Tries to unhook Windows functions monitored by Cuckoo.

DLL-loaded -> its visible through the JSON file that the malware is trying to load some dll which look suspicious.

DNSAPI.dll-> This API is used to manipulate dns cache in windows. A malware can hijack this file to redirect users to any advertisement or it may start downloading payloads.

sspicli .dll-> This is a security support library for windows and is used to play with certificates. This library can be hijacked to manipulate security certificates in Windows.

Rpcrt4.dll -> This is remote procedure call library which is used to call remote procedures i.e. functions that are deployed on remote computer. A malware can use this file to contact its master.

cryptbase.dll-> this is the core cryptography library for Windows. A malware can use it to encrypt files and later ask for ransoms or for some other purposes.

Strings -> This portion contains the extracted strings during the execution phase of a file. In our case we can see some suspicious strings for example tor. Tor urls are mostly used for anonymous communication and are definitely suspicious.

The JSON report is converted into MIST format using a cuckoo2mist converter function in python.

```
varnit@varnit-HP-EliteBook-840-G1:~/projects/shellcode/Cuckoo2Mist-master/Cuckoo2Mist$ python cuckoo2mist.py -i /home/varnit/.cuckoo/storage/anal
Starting Cuckoo2Mist converter.
```

Figure 5.18 JSON report converted to MIST format using cuckoo2mist converter

This is the screenshot for the MIST report.

```
varnit@varnit-HP-EliteBook-840-G1:~/projects/shellcode/Cuckoo2Mist-master/Cuckoo2Mist$ cat report.mist
# process 3048 thread 2548 #
06 08 | 0879df9c 76a40000
06 02 | 00000000 0c94b872 76a40000 76a54f2b 03983583
06 02 | 00000000 0c94b872 76a40000 76a5359f 0d39d8d5
06 02 | 00000000 0c94b872 76a40000 76a51252 0cd15a65
06 02 | 00000000 0c94b872 76a40000 76a54208 0cd15925
06 02 | 00000000 0c94b872 76a40000 76a54d28 052dde88
06 02 | 00000000 0c94b872 76a40000 76ad4195 055c3c37
06 02 | 00000000 0c94b872 76a40000 76a5d31f 0d0cf405
06 02 | 00000000 0c94b872 76a40000 76a6ee7e 0707f4f2
06 02 | 00000000 0c94b872 76a40000 7758441c 0b120e72
06 02 | 00000000 0c94b872 76a40000 775ac50e 044f76e3
06 02 | 00000000 0c94b872 76a40000 775ac381 0fe498d2
06 02 | 00000000 0c94b872 76a40000 76a6f088 03700774
06 02 | 00000000 0c94b872 76a40000 775905d7 0bb168d4
06 02 | 00000000 0c94b872 76a40000 775aca24 01fe31b4
06 02 | 00000000 0c94b872 76a40000 77560b8c 02339373
06 02 | 00000000 0c94b872 76a40000 7761fde8 0f4fd013
06 02 | 00000000 0c94b872 76a40000 775b1e1d 09498b22
06 02 | 00000000 0c94b872 76a40000 76ad4761 0bccf9fe
06 02 | 00000000 0c94b872 76a40000 76accd11 0f2ae5a7
06 02 | 00000000 0c94b872 76a40000 01284000 00a51753
06 02 | 00000000 0c94b872 76a40000 76ad424f 0f9a43f8
06 02 | 00000000 0c94b872 76a40000 76ad46b1 0fe8ab08
06 02 | 00000000 0c94b872 76a40000 76ae6676 0c3b4cd8
06 02 | 00000000 0c94b872 76a40000 76ad4751 00d14e28
06 02 | 00000000 0c94b872 76a40000 76ae65f1 0d43dcd8
06 02 | 00000000 0c94b872 76a40000 76ad47c1 064773b5
06 02 | 00000000 0c94b872 76a40000 76ad47e1 06a1df25
06 02 | 00000000 0c94b872 76a40000 76ad47f1 092dca18
06 02 | 00000000 0c94b872 76a40000 012840b0 0d98c174
06 06 | 000000a8
09 12 | 000000a8 00020019 06d333c5
06 06 | 000000a8
09 12 | 000000a8 00020019 00935c65
06 06 | 000000a8
05 03 | 00000be8 00040000 00000004 ffffffff 00002000 004d0000
05 03 | 00000be8 00001000 00000004 ffffffff 00001000 004d0000
05 03 | 00000be8 00005000 00000004 ffffffff 00001000 004d1000
05 03 | 00000be8 00005000 00000004 ffffffff 00001000 004d6000
09 12 | 000000a8 00020019 077f9aa5
09 12 | 000000ac 00020019 071f78b3
09 12 | 000000b0 00020019 02ec67e3
05 03 | 00000be8 001a0000 00000004 ffffffff 00002000 00a60000
05 03 | 00000be8 00001000 00000004 ffffffff 00001000 00bc0000
06 07 | 0cfaec45 73ef0000 00000000 02cd643c
06 02 | 00000000 0cfaec45 73ef0000 73f04571 0bd62c3b
06 04 |
06 08 | 056484ec 75570000
```

Figure 5.19 MIST report for the sample.

For prediction of maliciousness of the sample, the best algorithm has to be found out. We compare all algorithms and according to the highest score, we find KNN to be the best.

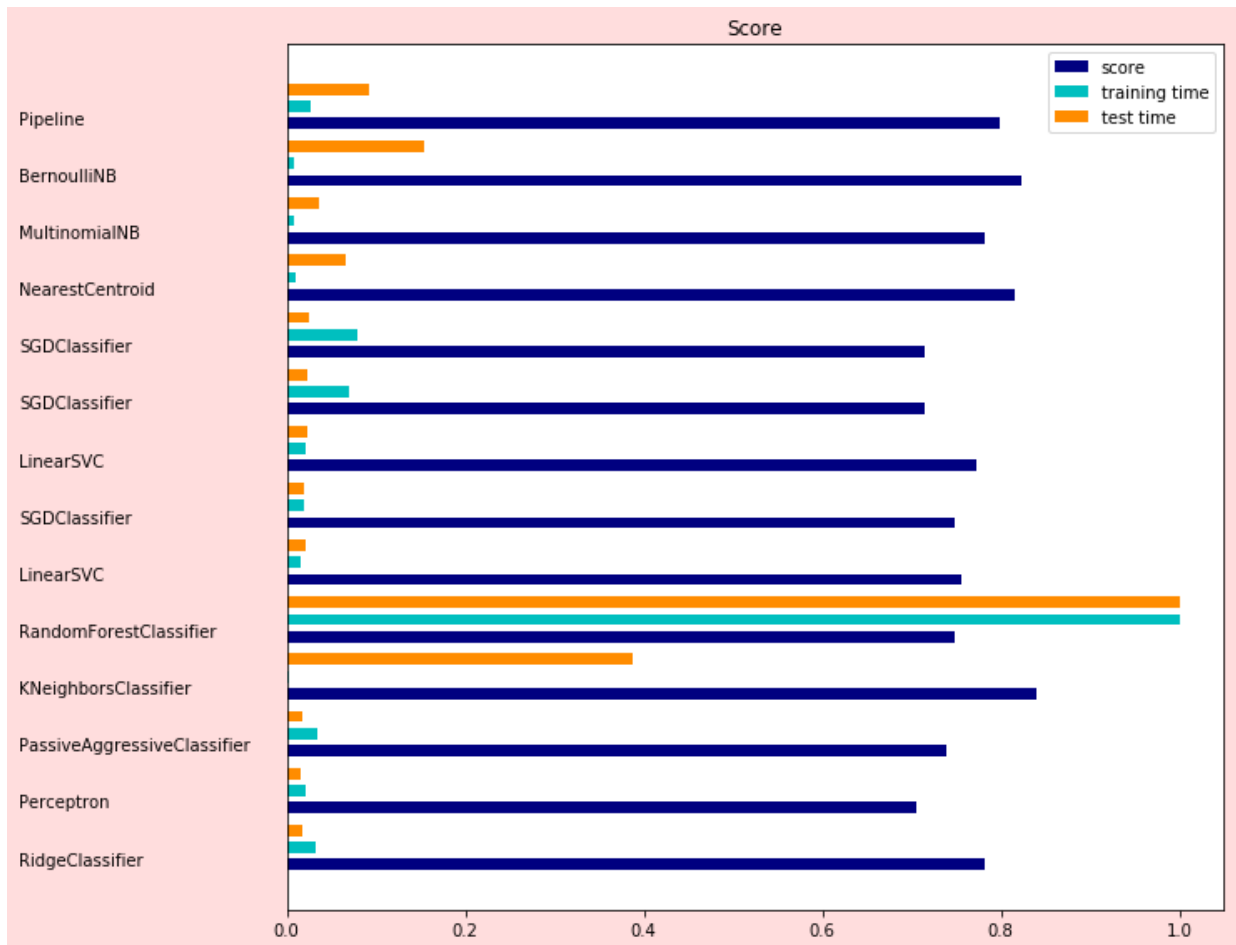


Figure 5.20 Graph comparing all algorithms and finding that KNN is best suited for analysis

This MIST report is converted into an N Gram sparse matrix using `vectorizer.transform()` function of python. Then it is converted into an array. KNN classifier algorithm is applied to this array to predict behavior maliciousness of this sample. It is predicted that there is 60% chance that this sample is malicious and 40% chance that it is clean.

```
In [45]: knn.predict_proba(transformed_mat)
Out[45]: [0.6, 0.4]
```

Figure 5.21 KNN Algorithm predicting maliciousness of the sample from transformed matrix

5.2.3 Snort Analysis

Cuckoo generates a `dump.pcap` file. Snort converts this pcap file into a text file. A screenshot of that file is shown below.

```

varnit@varnit-HP-EliteBook-840-G1:~/cuckoo/storage/analyses/latest$ sudo snort -c /etc/snort/snort.conf -r dump.pcap
Running in IDS mode
[snort] - snort(1)

--- Initializing Snort ---
Initializing Output Plugins!      RandomForestClassifier bootstrap=true, class_weight=None, criterion=gini
Initializing Preprocessors!      max_depth=None, max_features=auto, max_leaf_nodes=None
Initializing Plug-ins!           min_samples_leaf=1, min_samples_split=None
Parsing Rules file "/etc/snort/snort.conf"
PortVar 'HTTP_PORTS' defined : [ 80:81 311 383 591 593 901 1220 1414 1741 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:7145 7510 7777 7779 8000 8008 8014 8028 8080 8085 8088
8090 8118 8123 8180:8181 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444 41080 50002 55555 ]
PortVar 'SHELLCODE_PORTS' defined : [ 0:79 81:65535 ]
PortVar 'ORACLE_PORTS' defined : [ 1024:65535 ]
PortVar 'SSH_PORTS' defined : [ 22 ]
PortVar 'FTP_PORTS' defined : [ 21 2100 3535 ]
PortVar 'SIP_PORTS' defined : [ 5060:5061 5600 ]
PortVar 'FILE_DATA_PORTS' defined : [ 80:81 110 143 311 383 591 593 901 1220 1414 1741 1830 2301 2381 2809 3037 3128 3702 4343 4848 5250 6988 7000:7001 7144:7145 7510 7777 7779 8000 8008 8014 8028 8
080 8085 8088 8090 8118 8123 8180:8181 8243 8280 8300 8800 8888 8899 9000 9060 9080 9090:9091 9443 9999 11371 34443:34444 41080 50002 55555 ]
PortVar 'GTP_PORTS' defined : [ 2123 2152 3386 ]
Detection:
  Search-Method = AC-Full-Q
  Split Any/Any group = enabled
  Search-Method-Optimizations = enabled
  Maximum pattern length = 20
  Tagged Packet Limit: 256
Loading dynamic engine /usr/local/lib/snort_dynamicengine/libsf_engine.so... done
Loading all dynamic detection libs from /usr/local/lib/snort_dynamicrules...

```

Figure 5.22 A dump.pcap file generated by Cuckoo

We find Random Forest to be the best in this case by comparing it with all other algorithms. So KNN is being used for behavior analysis while Random Forest for Snort analysis in this case. We have calculated which algorithm scores best. The choice of algorithm may vary according to the data. Though the graph below shows that a lot of time is taken for training and testing for Random Forest, but the score is good and our main aim is to reduce the number of false positives and false negatives.

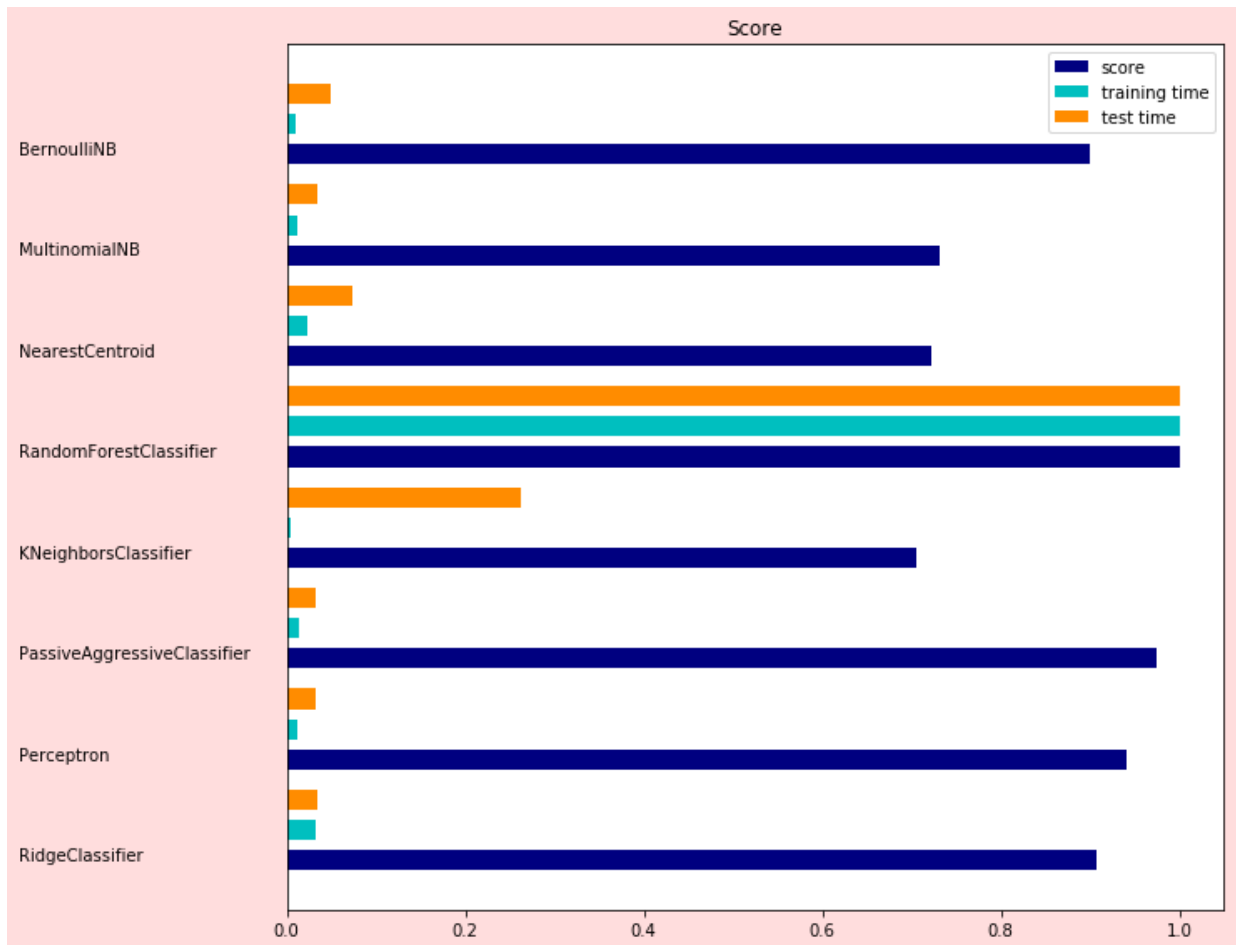


Figure 5.23 Graph comparing all algorithms and showing Random Forest to be the best for analysis

A text report generated by Snort is converted into a sparse matrix called 'transformed_mat'. Random Forest classifier is used for predicting the probability of network maliciousness of this sample. The Random Forest classifier predicts the sample to be 74% malicious and 26% clean.

```

In [ ]: rf.predict_proba(transformed_mat)

In [5]: |
         array([0.74,0.26])

In [ ]:

```

Figure 5.24 Random Forest algorithm predicts maliciousness of the sample to be 74%

5.2.4 Sandbox Evasion Detection

The screenshot below shows that this sample tries to evade the sandbox. Therefore this definitely is a malware.

```
{
  "markcount": 1,
  "families": [],
  "description": "A process attempted to delay the analysis task.",
  "severity": 3,
  "marks": [
    {
      "type": "generic",
      "description": "9.exe tried to sleep 2005199973 seconds, actually delayed analysis time by 2005199973 seconds"
    }
  ],
  "references": [],
  "name": "antisandbox_sleep"
},
{
  "markcount": 1,
  "families": [],
  "description": "Checks the CPU name from registry, possibly for anti-virtualization",
  "severity": 3,
  "marks": [
    {
      "category": "registry",
      "ioc": "HKEY_LOCAL_MACHINE\\HARDWARE\\DESCRIPTION\\System\\CentralProcessor\\0\\ProcessorNameString",
      "type": "ioc",
      "description": null
    }
  ],
  "references": [],
  "name": "antivm_generic_cpu"
},
}
```

Figure 5.25 Evasion of sandbox by the sample

5.3 Final Results







For the final testing we have taken 40 samples. Some of them are considered clean and some malicious by standard anti-virus software. For every malicious file that we find, we quarantine it after logging. Following are the benefits of our model as compared to standard models present in the market today:

- Our model has been successfully able to detect all malware with 100% accuracy.
- Our model is free as compared to standard antivirus software and Intrusion Detection Prevention Systems which are costing as high as thousands of dollars.

- Our model is non-proprietary and non-central. Therefore it is free from bias. It is also free from illegitimate profit making where companies would float their self-created malware and then propose paid solutions for them.
- Our model has an instant update facility. It is updated at all times. The moment a new malware hits the Internet, its entire signature and behavior reaches the model and it updates its database accordingly.
- It is a fast model as compared to other models in the system. Its complexity is log n. the work is distributed and divided among all nodes. Thus it is never resource-heavy and always fast no matter how heavy the analysis of the sample is.
- In our model everyone benefits. Both consumers and companies. Consumers get efficient anti malware facilities for free and companies get latest malware signatures which were previously undetected.

5.3.1 Results obtained from the PosDeF after all four steps

After the final calculation, we have also categorized the malware according to the level of severity. Informational means it has very less malware behavior and is not a big threat. Low means the severity is more than Informational but still the threat is not great. High means it poses a great threat to the system. Finally, Critical means the highest level of threat and this type of malware must be dealt with immediately.

<u>Level of Severity</u>	<u>Combined Probability</u>	<u>Color</u>
• Clean	<0.5000	
• Informational	0.5000-0.5500	
• Low	0.5500-0.6000	
• Medium	0.6000-0.6500	
• High	0.6500-0.7000	
• Critical	>0.70	

ffdf8eb73c7e506d8e1006cf573bd76e	1.00	0.6	0.72	0	Malicious	0.58	
fff8783b7567821cec8838d075d247e1	1.00	0.5	0.66	0	Malicious	0.54	
fff71367aec8f4985fb1071aad9bb677	1.00	0.5	0.77	0	Malicious	0.5675	
ffaf901cce614413547e4ff5a3ad105d	0.96	0.3	0.73	1	Malicious	0.7475	
ffac33bb85018d70153a36bf39ff1406	1.00	0.4	0.75	1	Malicious	0.7875	
ffaf901cce614413547e4ff5a3ad105d	0.96	0.3	0.7	1	Malicious	0.74	
ffb142b184585cb95354997516f050e4	0.54	0.4	0.8	1	Malicious	0.685	
4087f52a17eb28592a7dc0d8440a980e	0.26	0	0.26	0	Clean	0.13	
f7b53b4bd50c13d17f5c54f82cde7836	0	0.2	0.25	0	Clean	0.1125	
81418288d97ad8fddee1a91538a85a6b	0	0	0.25	0	Clean	0.0625	
ba4e1a60bd20ca7978c76d79f19e37f0	0.22	0.3	0.27	0	Clean	0.1975	
fe2b659d941440294ab90559acf69f11	0.22	0.2	0.25	0	Clean	0.1675	
3da66ef520d45081dcffdaecd3de17c8	0	0	0.22	0	Clean	0.055	
0f498e1e332f1c1fbf32b558805ed0d5	0.36	0.1	0.25	0	Clean	0.1775	
ad61f7afe913b2642650504df2	0	0.2	0.26	0	Clean	0.115	

83aa63								
44daf0a410ab80e7cab7c12ede5ffb34	0.28	0.2	0.25	0	Clean	0.1825		
b2f75222e51d1e896951787ae9de8bb6	0.23	0.1	0.14	0	Clean	0.1175		
b82466f58fd7776c135890485fd119a1	0.78	0.7	0.62	0	Malicious	0.525		
73c5a538151abc30d7d4c164fca14131	0.69	0.4	0.34	1	Malicious	0.6075		
e26ce823eb40720c2ec9c5334d578369	0.96	0.7	0.77	0	Malicious	0.6075		
03fff11a415b7026c1fc01e45d771c93	0.87	0.5	0.79	0	Malicious	0.54		
5effd66560365ded439d209f4d493ece	1.00	0.6	0.72	0	Malicious	0.58		
f428a574e9f26745fb70ee3128daf876	0.92	0.4	0.66	0	Clean	0.495		
03b9ab193fb9c0a4d57365cb080bc88b	0.96	0.5	0.77	0	Malicious	0.5575		
a14df5a55c2f3b760c37953f36d5f459	0.79	0.5	0.73	0	Clean	0.505		
6d7cbfd1a6f527df4546da1f6887a339	1.00	0.3	0.75	1	Malicious	0.7625		
718e0121fb212ea4448e67648def9fb9	1.00	0.6	0.7	1	Malicious	0.825		
ca80332eaa27a9a97327b4e78ade9574	0.77	0.62	0.70	1	Malicious	0.7725		
a44f2d1a832aa9aba551d552c69f44da7b02ff7a126c3a6eb4b1	0.62	0.78	0.77	0	Malicious	0.5425		

22a32265c34b							
c9cfdc43448980fcd17066bc00							
b73baa	0.61	0.53	0.71	1	Malicious	0.7125	

Table 5.1 : Final Results obtained by testing sample files

5.3.2 Comparison with the existing Centralized systems

In the existing frameworks commonly used today, centralised technologies are used. These technologies are allow and inefficient. We have used decentralised technologies like Apache Spark on Amazon AWS Cloud for implementation of the model. The graph below shows the result of training on Spark. It shows that as the number of nodes increase, execution time decreases. From 140 when running our program on a single node and reduces to 40 seconds when run on 3 or 4 nodes.

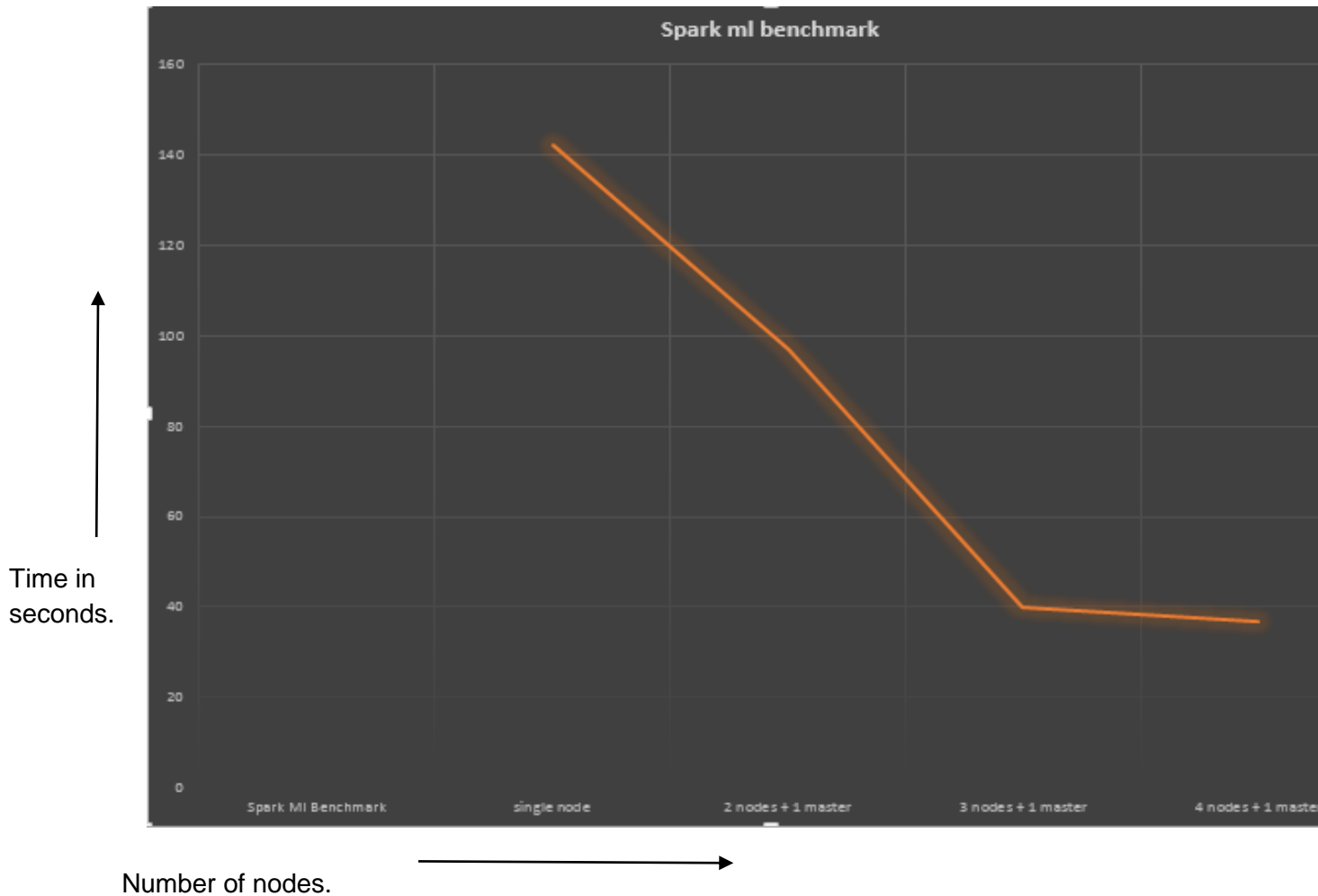


Figure 5.26 Graph showing training on Apache Spark.

The next graph shows that the distribution is successfully done on all four nodes and all of these are used by the program for distributed execution. Here we have taken 4 machines with IP Addresses:

- 172.31.29.19
- 172.31.23.172
- 172.31.28.58
- 172.31.25.122

We have tested all four nodes for an aggregated load of one hour. The X-axis shows time and the Y-axis shows distribution of load.

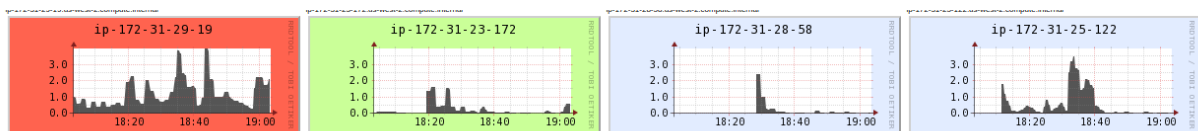
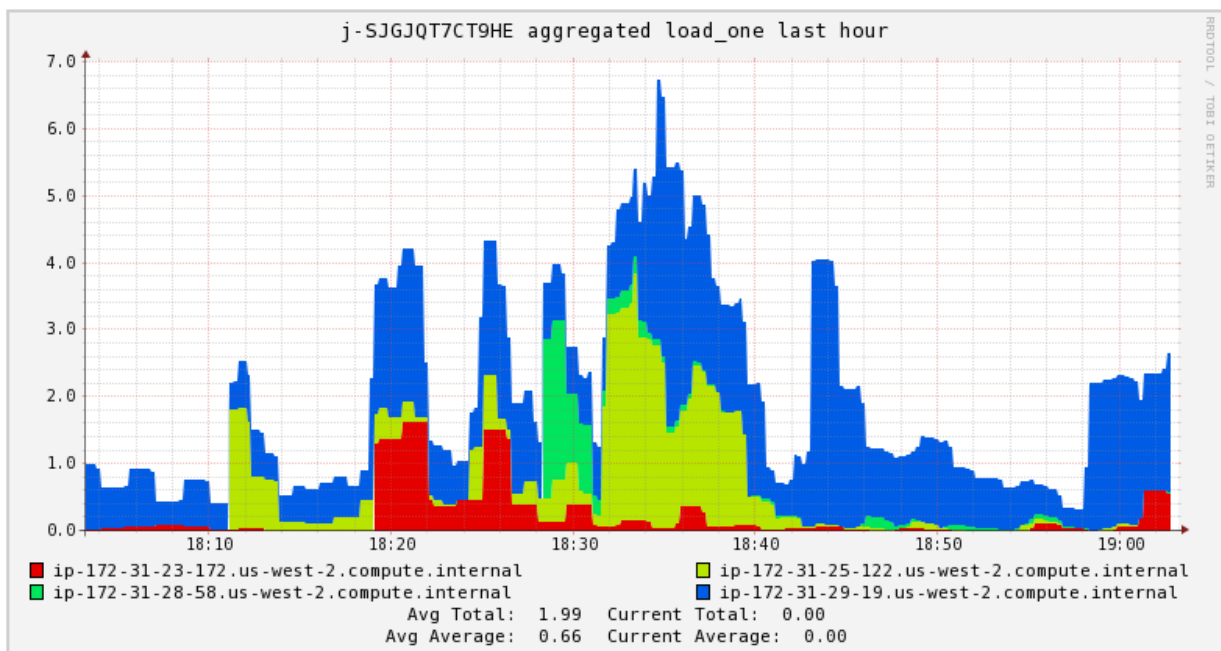


Figure 5.27 Distribution of load on 4 nodes aggregated for 1 hour.

When we launch an instance of AWS, it is given a hostname. Our Amazon EC2 private DNS name looks like this:

ip-173-31-23-172.us-west-2.compute.internal.

ip-173-31-23-172 is IPV4 address which is the internal domain.

Compute is the service and west is the region. This name is for a Linux instance.

5.3.2.1 Working of the Distributed Framework

- 1) We create a sample Amazon EMR cluster by using the AWS Management Console.
- 2) After going to the AWS Management Console, we select Elastic MapReduce Service. Amazon EWS gives a managed Hadoop framework. Thus it becomes easy, fast and cost-effective to prove huge volumes of data across dynamic, scalable Amazon EC2 instances. It also lets you run other distributed frameworks like Apache Spark.
- 3) Then we select the number of nodes required to run our model. Here we select first one, then two, then three and then four nodes to find out the effectiveness of distribution.
- 4) We also select the tools required for processing i.e. 'Apache Spark' and 'Hadoop'. Spark for distributed data processing and Hadoop for in-node data processing.
- 5) After that, an Apache Zeppelin Notebook is exposed to us. Here we write our program and execute it in a distributed manner using Hadoop and Apache Spark. Zeppelin is a web based notebook for interactive data exploration. It is used for data ingestion, discovery, analytics, visualization and collaboration.

5.3.2.2 Blockchain for Records and Rewards

Blockchain is a data structure that distributes trust among many nodes instead of one. It's like a single link list that lives on everybody's computer (miner) on the network. It is a huge chain of transactions. It's more like a distributive ledger. It grows continuously as more and more transactions are added to the network [67]. A blockchain is an immutable list of transactions that have ever occurred in the network that all the nodes have to store a copy of. Each block contains address to the next block. It also has the complete record of the transaction including timestamp and transactional data.

Rewards: are a way to recognize nodes and their users who give their CPU speed and execution power and their storage for the running of this system. The rewards are in the form of recognitions. They can take any form. Say a person contributes his resources to our framework. Now he becomes instrumental in discovering a new malware in the process. Then his contributions can be recognized by putting his name in CVE database along with the name of the malware to appreciate that he helped in discovering that malware.

The blockchain is verifiable. The transaction is represented as a 25 character hash. This hash is a string of letters and numbers. All the miners in the network will validate the transaction for it to be added to the blockchain. All these miners have to vote on the validity of each transaction. Before they can vote, they have to provide a proof that they have computed this random mathematical problem. This is the proof of their computational work. This is called 'Proof of work' algorithm.



Figure 5.28 A part Blockchain of five blocks generated by Implemented model.

1. **Nonce** - Nonce is a 32-bit (4-byte) field in Bitcoin network. It is used to decide how hard it would be to mine a new block for the transaction to proceed.
2. **Hash**- Hash is an alphanumeric value that is calculated for the data present in each block. This is the digital fingerprint of this data. It is because of the hashes why Blockchain can be trusted. If the data is modified in some way, its hash value changes and is rejected by other blocks in the chain.

3. **Md5sum**- This is the message digest of the file being profiled. It is a calculated checksum. The Hash field gives hash value for the entire block whereas Md5sum gives the hash value of the file present in the block.
4. **Previous Hash**- Is used for maintaining the integrity of the chain. This contains hash of the previous block.
5. **Status**- Tells whether the tested sample is malicious, clean or unknown.
6. **Timestamp**- is the Unix timestamp format. It stores the time when the block was created. This is there in case we want to scan the available files again and change the status.

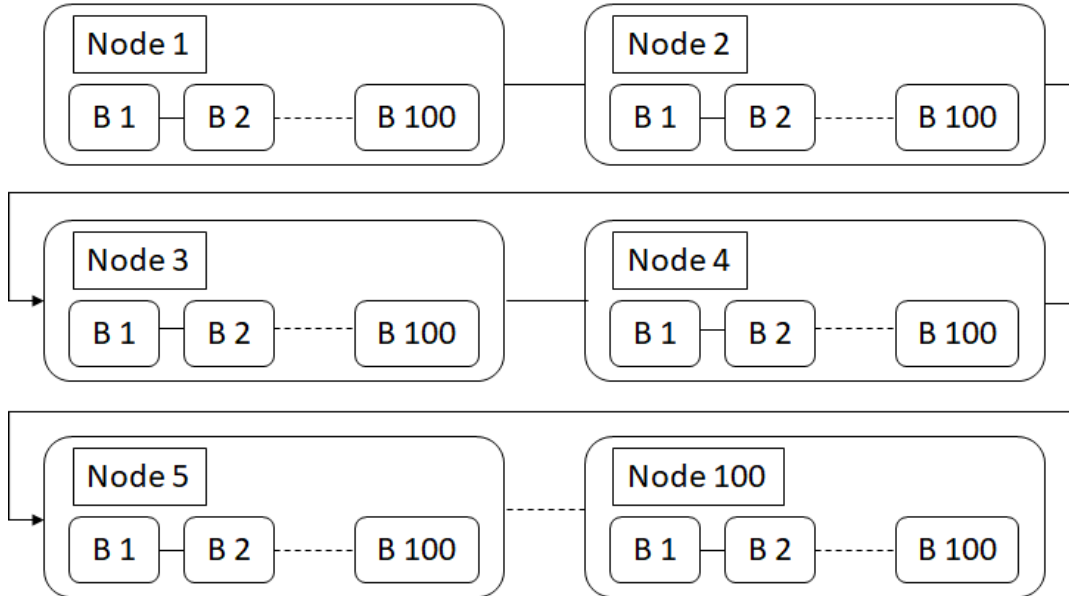
5.3.2.3 Using Ethereum for maintaining Blockchains

Initially Bitcoins were used as a payment system which is digitized and has no central regulatory system. It is a peer to peer currency. Ethereum is an infrastructure in addition to being a payment system. It provides Smart Contracts and even Crowdsourcing. It provides developers with expanded functionality because it is written in a Turing Complete language. A Turing complete language enables you to do anything with it provided you have enough time and computing power.

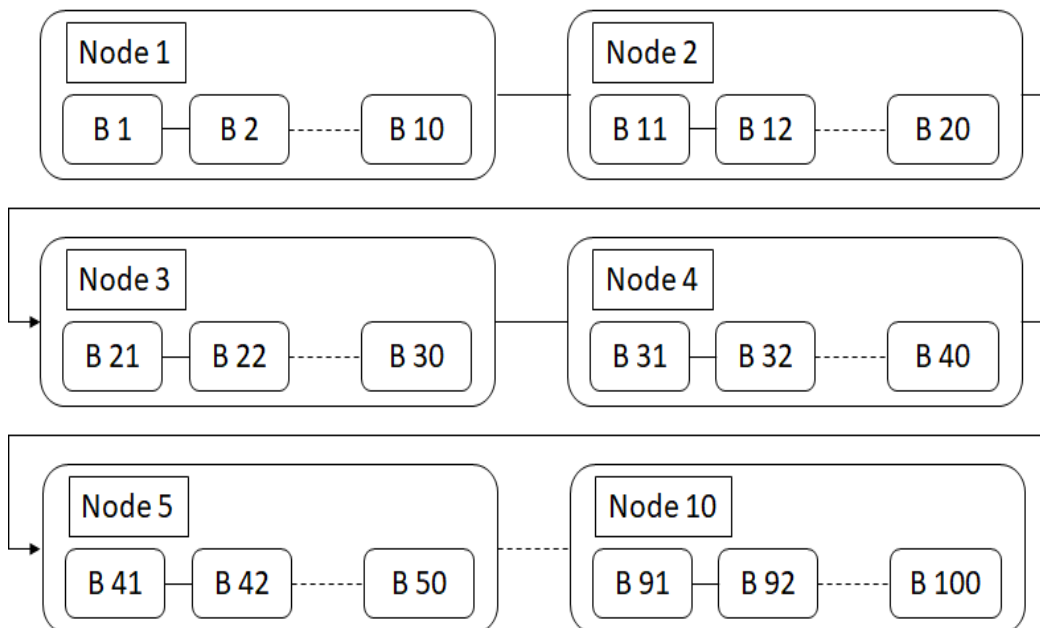
The database in both Bitcoin and Ethereum is stored in every node. Now this increases the storage requirement of every node participating in the detection mechanism. Therefore we build the Ethereum blockchain on top of our Hadoop Cluster in an indexed manner.



Ethereum alone without Hadoop



Ethereum on top of Hadoop



Index	Node
1 - 10	N1
11 - 20	N2
21 - 30	N3
31 - 40	N4
41 - 50	N5
51 - 60	N6
61 - 70	N7
71 - 80	N8
81 - 90	N9
91 - 100	N10

Figure 5.29 Comparison of Ethereum Blockchain database architecture without Indexing and with Indexing on top of Hadoop.

5.4 Conclusions

- Polymorphic shellcodes can be detected by the above mentioned framework. This is an organized manner through which most of the polymorphic shellcodes can be correctly identified and the risks can be mitigated. Our model was accurately able to detect all the malwares. In line 38 of the final results table, the sample was expected to be clean but our model showed it to be malicious of a mild kind. We ran it on NoDistribute and no antivirus was able to declare it as malicious. This can be a case when the existing antivirus solutions were not able to detect the malware because of several reasons, maybe it was of a mutating kind or it evaded the sandbox. Whatever may be the case, our model was able to prove that it was indeed a malicious sample with maliciousness probability of 0.505.
- Similarly on line number 31, the present models, predicted the probability of the sample to be clean. But our model declared it as malicious with maliciousness probability of 0.525. This can be the case that the sample was new therefore the existing solutions could not detect its maliciousness. Since our model takes a 360⁰ approach to detection therefore our model was successful in detecting the malware.
- In line 41, 42 and 43, we have details of advance malware that we had created on our own using Metasploit, AdmMutate, Clet and other techniques and inserted them in legitimate code using Shellter. In NoDistribute, only 1 out of 36 popular antimalware solutions were able to detect them but our framework detected them accurately.
- This Machine learning framework is used for the detection of all types of malwares. The model is iterative, thus self- improving. It is made efficient and scalable using Amazon AWS cloud technologies. The load distribution is carried out by the framework of Apache Spark which uses Hadoop and HDFS architecture. Immutable dataset is stored in a secure manner through Blockchain. All nodes are rewarded for providing their computing power through tokens in the Blockchain. The complete model is in the execution stage and is running fine in a lab environment. It can be scaled easily to the real world implementation by increasing the number of nodes in the cloud.

5.5 Limitations of the Study

There are some limitations to this study.

- Distributive environment- The framework has to work in a distributive environment so as to be resource efficient. In a centralized environment, the framework can become slow and ineffective and the identification of malicious shellcode may take lot of time.
- High data requirement- More the data for the training of the model, more accurate will the model be. Thus one more limitation is the amount of dataset we use for training and testing purposes as there is a chance of false positives wherein a benign code may be identified as malicious and vice-versa because of paucity of enough data. Also deep learning will be more accurate if it has more iterations and that can happen with a large dataset.
- Dependence on ancillary technologies- the framework depends on Ethereum blockchain and cloud technologies. The Ethereum blockchain can be replaced with any other blockchain and Amazon cloud also can be replaced by other cloud platforms. But the overall efficiency of the framework is somewhat dependant on these ancillary technologies also. So when they improve, our framework improves too.

5.6 Further Research Potential

There is a lot of scope for further research in this subject. There may be more efficient ways to detect the polymorphic shellcodes. Those ways may take less time and may have very less chance of false positives. We tried the Neural Network approach also because of cheap GPUs. This approach works very well in a distributive environment. However, the score that this algorithm gave was less than the other algorithms. Therefore, we had to leave this approach. In future, when this algorithm is developed further, we may get a better score and use this algorithm. The model will improve as dataset increases. More the data means better trained the machine. Therefore as we get more samples for training our machine, the accuracy of the machine will start increasing further. Also when more users get attached in the block chain, more and more people will be giving their data for analysis, thus enhancing the dataset. Also, since all the systems participating in the blockchain are voluntarily giving their CPU services also thus as more block increase, the efficiency of the blockchain and thus the entire

system will also shows an increase. Cloud technology is used to carry out distributed services. As Cloud services become more efficient and cheap, the system improves further.

REFERENCES

- [1]Bazrafshan, Z., Hashemi, H., Fard, S. M. H., & Hamzeh, A. (2013, May). “A survey on heuristic malware detection techniques”. In Information and Knowledge Technology (IKT), 2013 5th Conference on (pp. 113-120). IEEE.
- [2] Mathur, K., & Hiranwal, S. (2013). “A survey on techniques in detection and analyzing malware executables”. International Journal of Advanced Research in Computer Science and Software Engineering, 3(4).
- [3] Carson, M. (2016). “Evaluating the Ability of Anti-Malware to Overcome Code Obfuscation”. Selected Computing Research Papers, 9.
- [4]Adeyinka, O. (2008, May). “Internet attack methods and internet security technology”. In Modeling & Simulation, 2008. AICMS 08. Second Asia International Conference on (pp. 77-82). IEEE.
- [5]Martin, R. A. (2001). “Managing vulnerabilities in networked systems”. Computer, (11), 32-38.
- [6]Mell, P., & Grance, T. (2002). “Use of the common vulnerabilities and exposures (cve) vulnerability naming scheme” (No. NIST-SP-800-51). National Institute of Standards and Technology Gaithersburg MD Computer Security Div.
- [7] Tsyganok, K., Tumoyan, E., Babenko, L., & Anikeev, M. (2012, October). “Classification of polymorphic and metamorphic malware samples based on their behavior”. In Proceedings of the Fifth International Conference on Security of Information and Networks (pp. 111-116). ACM.
- [8]Baysa, D., Low, R. M., & Stamp, M. (2013). “Structural entropy and metamorphic malware”. Journal of computer virology and hacking techniques, 9(4), 179-192.
- [9] Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). “Automatic analysis of malware behavior using machine learning”. Journal of Computer Security, 19(4), 639-668.

- [10] Santos, I., Devesa, J., Brezo, F., Nieves, J., & Bringas, P. G. (2013). "Opem: A static-dynamic approach for machine-learning-based malware detection". In International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions(pp. 271-280). Springer, Berlin, Heidelberg.
- [11] Oktavianto D, Muhandianto I. "Cuckoo malware analysis". (2013 Oct 16) Packt Publishing Ltd.
- [12] Blasing, T., Batyuk, L., Schmidt, A. D., Camtepe, S. A., & Albayrak, S. (2010, October). "An android application sandbox system for suspicious software detection". In 2010 5th International Conference on Malicious and Unwanted Software (MALWARE 2010) (pp. 55-62). IEEE.
- [13] Prayudi, Y., & Riadi, I. (2015). "Implementation of malware analysis using static and dynamic analysis method". International Journal of Computer Applications, 117(6).
- [14] Zolkipli, M. F., & Jantan, A. (2011, March). "An approach for malware behavior identification and classification". In Computer Research and Development (ICCRD), 2011 3rd International Conference on (Vol. 1, pp. 191-194). IEEE.
- [15] Bhardwaj, A., Avasthi, V., Sastry, H., & Subrahmanyam, G. V. B. (2016). "Ransomware digital extortion: a rising new age threat". Indian Journal of Science and Technology, 9(14), 1-5.
- [16] Shinde, R., Van der Veecken, P., Van Schooten, S., & van den Berg, J. (2016, December). "Ransomware: Studying transfer and mitigation". In Computing, Analytics and Security Trends (CAST), International Conference on (pp. 90-95). IEEE.
- [17] Dwivedi, K., & Dubey, S. K. (2014, September). "Analytical review on Hadoop Distributed file system". In Confluence The Next Generation Information Technology Summit (Confluence), 2014 5th International Conference- (pp. 174-181). IEEE.
- [18] Correia, R. C., Spadon, G., Eler, D. M., Olivete, C., & Garcia, R. E. (2018). "Teaching Distributed Systems Using Hadoop". In Information Technology-New Generations (pp. 355-362). Springer, Cham.
- [19] Jangra, A., & Bala, R. (2011). "Spectrum of cloud computing architecture: Adoption and avoidance issues". International Journal of Computing and Business Research, 2(2), 5.
- [20] Sareen, P. (2013). "Cloud computing: types, architecture, applications, concerns, virtualization and role of it governance in cloud". International Journal of Advanced Research in Computer Science and Software Engineering, 3(3).

- [21] MacDonald, T. J., Allen, D. W., & Potts, J. (2016). "Blockchains and the boundaries of self-organized economies: Predictions for the future of banking". In *Banking Beyond Banks and Money* (pp. 279-296). Springer, Cham.
- [22] Davidson, S., De Filippi, P., & Potts, J. (2018). "Blockchains and the economic institutions of capitalism". *Journal of Institutional Economics*, 1-20.
- [23] Tschorsch, F., & Scheuermann, B. (2016). "Bitcoin and beyond: A technical survey on decentralized digital currencies". *IEEE Communications Surveys & Tutorials*, 18(3), 2084-2123.
- [24] Böhme, R., Christin, N., Edelman, B., & Moore, T. (2015). "Bitcoin: Economics, technology, and governance". *Journal of Economic Perspectives*, 29(2), 213-38.
- [20]Kong, Deguang, et al. "SAS: semantics aware signature generation for polymorphic worm detection." *International Journal of Information Security* 10.5 (2011): 269-283.
- [21]Loh, Peter KK, and Brian WY Loh. "Cells—A novel IOT security approach." *Region 10 Conference (TENCON)*, 2016 IEEE. IEEE, 2016.
- [22]Chouchane, Radhouane, et al. "Detecting machine-morphed malware variants via engine attribution." *Journal of Computer Virology and Hacking Techniques* 9.3 (2013): 137-157.
- [23]Austin, Thomas H., et al. "Exploring hidden Markov models for virus analysis: a semantic approach." *System Sciences (HICSS)*, 2013 46th Hawaii International Conference on. IEEE, 2013.
- [24]Rad, Babak Bashari, Maslin Masrom, and Suhaimi Ibrahim. "Camouflage in malware: from encryption to metamorphism." *International Journal of Computer Science and Network Security* 12.8 (2012): 74-83.
- [25] Denzin, N. K., & Lincoln, Y. S. (1994). "Handbook of qualitative research". Sage publications, inc.
- [26] Nasrabadi, N. M. (2007). "Pattern recognition and machine learning". *Journal of electronic imaging*, 16(4), 049901.
- [27] Hastie, T., Tibshirani, R., & Friedman, J. (2009). "Unsupervised learning". In *The elements of statistical learning* (pp. 485-585). Springer, New York, NY.

- [28] Cavnar, W. B., & Trenkle, J. M. (1994, April). N-gram-based text categorization. In *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*(Vol. 161175).
- [29] “x86 Instruction Sequence”. Article. Accessed: 24 July 2017. Available via: http://x86.renejeschke.de/html/file_module_x86_id_217.html
- [30] Akritidis, P., Markatos, E. P., Polychronakis, M., & Anagnostakis, K. (2005, May). Stride: Polymorphic sled detection through instruction sequence analysis. In *IFIP International Information Security Conference* (pp. 375-391). Springer, Boston, MA.
- [31] Haugsness K. ”IDFAQ: What is polymorphic shell code and what can it do?”Article, SANS. Accessed: 10 July 2017. Available via: <https://www2.sans.org/security-resources/idfaq/what-is-polymorphic-shell-code-and-what-can-it-do/2/19>
- [32]Johansson K., ”Re:pen testing & obfuscated shellcode(more neat stuff)”, Article, Accessed: 9 July 2017, Available via: <http://seclists.org/pen-test/2004/Feb/69>
- [33] CourseHero, “Trampolining despite the fact that nop sledding makes”, Article, Accessed: 26 June 2017, Available via: <https://www.coursehero.com/file/p2o51ar/Trampolining-Despite-the-fact-that-NOP-sledding-makes-stack-based-buffer>
- [34] Polychronakis, M., Anagnostakis, K. G., & Markatos, E. P. (2007, September). Emulation-based detection of non-self-contained polymorphic shellcode. In *International Workshop on Recent Advances in Intrusion Detection* (pp. 87-106). Springer, Berlin, Heidelberg.
- [35] Wang, S. J., & Kao, D. Y. (2007). Internet forensics on the basis of evidence gathering with Peep attacks. *Computer Standards & Interfaces*, 29(4), 423-429.
- [36] Fuchsberger, A. (2005). Intrusion detection systems and intrusion prevention systems. *Information Security Technical Report*, 10(3), 134-139.
- [37] Christodorescu, M., & Jha, S. (2006). *Static analysis of executables to detect malicious patterns*. Wisconsin Univ-Madison dept of Computer Sciences.
- [38] Song, Y., Locasto, M. E., Stavrou, A., Keromytis, A. D., & Stolfo, S. J. (2010). On the infeasibility of modeling polymorphic shellcode. *Machine learning*, 81(2), 179-205.

- [39] Li, X., Loh, P. K., & Tan, F. (2011, September). Mechanisms of polymorphic and metamorphic viruses. In *2011 European intelligence and security informatics conference* (pp. 149-154). IEEE.
- [40] Borello, J. M., & Mé, L. (2008). Code obfuscation techniques for metamorphic viruses. *Journal in Computer Virology*, 4(3), 211-220.
- [41] Crandall, J. R., Su, Z., Wu, S. F., & Chong, F. T. (2005, November). On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *Proceedings of the 12th ACM conference on Computer and communications security* (pp. 235-248). ACM.
- [42] Nakashima, E., & Timberg, C. (2017). NSA officials worried about the day its potent hacking tool would get loose. Then it did. *Washington Post*.
- [43] Syverson, P., Dingleline, R., & Mathewson, N. (2004). Tor: The second generation onion router. In *Usenix Security*.
- [44] Popoola, S. I., Ojewande, S. O., Sweetwilliams, F. O., John, S. N., & Atayero, A. A. (2017). Ransomware: Current Trend, Challenges, and Research Directions.
- [45] Du, W., & Zhan, Z. (2002, December). Building decision tree classifier on private data. In *Proceedings of the IEEE international conference on Privacy, security and data mining-Volume 14* (pp. 1-8). Australian Computer Society, Inc..
- [46] Sung, A. H., Xu, J., Chavez, P., & Mukkamala, S. (2004, December). Static analyzer of vicious executables (save). In *20th Annual Computer Security Applications Conference* (pp. 326-334). IEEE.
- [47] Ferrand, O. (2013). How to detect the Cuckoo Sandbox and hardening it?. In *EICAR Annual Conf., Hannover, Germany*.
- [48] Qiao, Y., Yang, Y., He, J., Tang, C., & Liu, Z. (2014). CBM: free, automatic malware analysis framework using API call sequences. In *Knowledge engineering and management* (pp. 225-236). Springer, Berlin, Heidelberg.
- [49] Dolan-Gavitt, B., Leek, T., Zhivich, M., Giffin, J., & Lee, W. (2011, May). Virtuoso: Narrowing the semantic gap in virtual machine introspection. In *2011 IEEE Symposium on Security and Privacy* (pp. 297-312). IEEE.

- [50] Hejazi, S. M., Talhi, C., & Debbabi, M. (2009). Extraction of forensically sensitive information from windows physical memory. *digital investigation*, 6, S121-S131.
- [51] Qiao, Y., Yang, Y., Ji, L., & He, J. (2013, July). Analyzing malware by abstracting the frequent itemsets in API call sequences. In *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 265-270). IEEE.
- [52] Chandramohan, M., Tan, H. B. K., & Shar, L. K. (2012, November). Scalable malware clustering through coarse-grained behavior modeling. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (p. 27). ACM.
- [53] Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of Computer Security*, 19(4), 639-668.
- [54] Jain, S., & Meena, Y. K. (2011, August). Byte level n-gram analysis for malware detection. In *International Conference on Information Processing* (pp. 51-59). Springer, Berlin, Heidelberg.
- [55] Trinius, P., Willems, C., Holz, T., & Rieck, K. (2009). A malware instruction set for behavior-based analysis.
- [56] Roesch, M. (1999, November). Snort: Lightweight intrusion detection for networks. In *Lisa* (Vol. 99, No. 1, pp. 229-238).
- [57] Kaur, N., Bindal, A. K., & PhD, A. (2016). A complete dynamic malware analysis. *International Journal of Computer Applications*, 135(4), 20-25.
- [58] Thain, D., Tannenbaum, T., & Livny, M. (2005). Distributed computing in practice: the Condor experience. *Concurrency and computation: practice and experience*, 17(2-4), 323-356.
- [59] Zaharia, M., Xin, R. S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Ghodsi, A. (2016). Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11), 56-65.
- [60] Lam, C. (2010). *Hadoop in action*. Manning Publications Co..
- [61] Shanahan, J. G., & Dai, L. (2015, August). Large scale distributed data science using apache spark. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 2323-2324). ACM.

- [62] Armbrust, M., Bateman, D., Xin, R., & Zaharia, M. (2016, June). Introduction to spark 2.0 for database researchers. In *Proceedings of the 2016 International Conference on Management of Data* (pp. 2193-2194). ACM.
- [63] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., & Khan, S. U. (2015). The rise of “big data” on cloud computing: Review and open research issues. *Information systems*, 47, 98-115.
- [64] Ostermann, S., Iosup, A., Yigitbasi, N., Prodan, R., Fahringer, T., & Epema, D. (2009, October). A performance analysis of EC2 cloud computing services for scientific computing. In *International Conference on Cloud Computing* (pp. 115-131). Springer, Berlin, Heidelberg.
- [65] Juve, G., Deelman, E., Vahi, K., Mehta, G., Berriman, B., Berman, B. P., & Maechling, P. (2009, December). Scientific workflow applications on Amazon EC2. In *2009 5th IEEE International Conference on e-science workshops* (pp. 59-66). IEEE.
- [66] Luo, Y., Luo, S., Guan, J., & Zhou, S. (2013). A RAM Cloud Storage System based on HDFS: Architecture, implementation and evaluation. *Journal of Systems and Software*, 86(3), 744-750.
- [67] Zyskind, G., & Nathan, O. (2015, May). Decentralizing privacy: Using blockchain to protect personal data. In *2015 IEEE Security and Privacy Workshops* (pp. 180-184). IEEE.

APPENDIX 1: PUBLICATION DETAILS

Sr. No.	Journal Name	Title of the Paper	Conference Name	Indexing
1.	Advances in Intelligent Systems and Computing (AISC), Computational Intelligence: Theories, Applications and Future Directions, Springer Verlag, Singapore , ISBN-10: 981131134X , ISBN-13: 978-9811311345, 17 September 2018	'Behavioural Analysis of Recent Ransomwares and Prediction of Future Attacks by Polymorphic and Metamorphic Ransomware' (DOI: 10.1007/978-981-13-1135-2_6)	International Conference on Computational Intelligence: Theories, Applications and Future Directions December 6th - 8th, 2017, Indian Institute of Technology Kanpur , India,	<ul style="list-style-type: none"> • SCOPUS • ISI Proceedings • DBLP-Ulrich's • EI-Copendex • Zentralblatt Math • MetaPress • Springerlink • Google Scholar
2.	Journal of Advance Research in Dynamical & Control Systems(JARDCS), U.S.A , Vol. 10, 12-Special Issue, 2018, ISSN 1943-023X,	'Employing Decentralized Technologies for Machine-Learning Detection of Advanced Malware' (*Best Paper Award and First prize in the Track)	International Conference, 'TeLMISR 18' 21, 22 May 2018, Vivekananda School of Information Technology, Delhi,	<ul style="list-style-type: none"> • SCOPUS • Elsevier • MathSciNet • Computers and Applied Sciences
3.	MERI-Journal of Management & IT (UGC approved journal) , Volume 11, ISSN: 0974-2093, October-2017	'PMASCE- Polymorphic and Metamorphic Shellcode Creation Engine' (DOI:10.25089/MERI/2017/v11/i1/164011)		<ul style="list-style-type: none"> • J-Gate • Root Indexing • i-Scholar
4.	MERI-Journal of Management & IT (UGC approved journal), Volume 10, ISSN: 0974-2093, April-2017.	'Wannacry Malware Analysis' (DOI : 10.25089/MERI/2017/v10/i2/151167)		<ul style="list-style-type: none"> • J-Gate • Root Indexing • i-Scholar
5.	ABS Journal of Management, Volume 5, Issue 1, ISSN: 2319-684X, January-2017.	'Harmful Effects of Cybercrime in Business and Economic Sustainability'	International Conference on Technological Revolution for International	

			Business and Economic Sustainability (TRIBES- 2017), ABS Business School, Noida,	
6.	MERI Conference Proceedings, ISBN:978-93-84871-07-9, 22-23 February- 2017.	'Digital Economy- Some Advancements in the Modes of Digital Payments'	International Conference on Digital Economy: Challenges and Opportunities, MERI College	
7.	MERI Conference Proceedings, ISBN:978-93-84871-07-9, 22-23 February- 2017.	'Digital Economy: Dieasy India(Making Village as a Smart Village)'	International Conference on Digital Economy: Challenges and Opportunities, MERI College	
8.	MERI Conference Proceedings, ISBN:978-93-84871-07-9, 22-23 February- 2017.	'Machine Learning-A Game Changer in Digital Economy'	International Conference on Digital Economy: Challenges and Opportunities, MERI College	
9.	NCDM Conference Proceedings, ISBN:978-93-85777-37-0 February-2016.	'Cloud Security is Important for Health Information Systems'-	Third National Conference on Data Mining and ICT (NCDM 15-16), Allana Institute of Management Sciences, Camp, Pune	

Was a **reviewer** at ICIM'19 (International Conference on Intelligent Machines) sponsored by Springer held on 15th and 16th March 2019 , Baba Farid Group of Institutions, India.

APPENDIX 2 : COMPANY LETTERS AND MAIL CONVERSATIONS

1. ISACA CPE Letter for speaking in the **03rd April 2015** session on the topic **“Cyber Crimes and Cyber Forensics”**.
2. ISACA CPE Letter for speaking in the **09th July 2016** session on the topic **“Polymorphic Shellcode Detection”**.
3. ISACA CPE Letter for speaking in the **10th June 2017** session on the topic **“Advanced Malware and Ransomware, Case Study: WannaCry, Live Demo”**.
4. ISACA Letter for speaking in the **19th March 2019** , ‘SheLeadsTech’ event on the topic **“AI and ML- New Innovations in Cybersecurity”** .
5. Letter from the Company **“Reinova Biz Consultant”** for implementation of the framework.
6. Mail conversations with **“virusshare.com”** for data acquisition for malicious files.
7. Reviewer Certificate for International Conference.

ISACA CPE Letter for speaking in the 03rd April 2015 session on the topic “Cyber Crimes and Cyber Forensics”.



238, ANARKALI COMPLEX
JHANDEWALAN, NEW DELHI – 110055, INDIA
TEL: +91 11 49429733
www.isacanewdelhi.org.in
www.isaca.org

To Whomsoever It May Concern

This is to certify that, **Ms. Navneet Kaur Popli** of MERI College was an invited speaker at ISACA, New Delhi Chapter CPE event on 03rd April 2015.

Topic of Session - “**Cyber Crimes and Cyber Forensics**”

ISACA New Delhi Chapter, thank you for sharing your valuable time and knowledge with our members.

For ISACA New Delhi Chapter

A handwritten signature in blue ink, appearing to read "G L Manchanda".

G L Manchanda
Vice-President

ISACA CPE Letter for speaking in the 09th July 2016 session on the topic “Polymorphic Shellcode Detection”.



238, ANARKALI COMPLEX
JHANDEWALAN, NEW DELHI – 110055, INDIA
TEL: +91 11 49429733
www.isacanewdelhi.org.in
www.isaca.org

To Whomsoever It May Concern

This is to certify that, *Ms. Navneet Kaur Popli* of MERI College was an invited speaker at ISACA, New Delhi Chapter CPE event on 09th July 2016.

Topic of Session - **“Polymorphic Shellcode Detection”**

ISACA New Delhi Chapter, thank you for sharing your valuable time and knowledge with our members.

For ISACA New Delhi Chapter

A handwritten signature in blue ink, appearing to read "G L Manchanda".

G L Manchanda
Vice-President

ISACA CPE Letter for speaking in the 10th June 2017 session on the topic “Advanced Malware and Ransomware, Case Study: WannaCry, Live Demo”.



238, ANARKALI COMPLEX
JHANDEWALAN, NEW DELHI – 110055, INDIA
TEL: +91 11 49429733
www.isacanewdelhi.org.in
www.isaca.org

To Whomsoever It May Concern

This is to certify that, **Ms. Navneet Kaur Popli** of MERI College was an invited speaker at ISACA, New Delhi Chapter CPE event on 10th June 2017.

Topic of Session “**Advanced Malware and Ransomware, Case Study: WannaCry, Live Demo**”

ISACA New Delhi Chapter, thank you for sharing your valuable time and knowledge with our members.

For ISACA New Delhi Chapter

A handwritten signature in blue ink that reads "G L Manchanda".

G L Manchanda
Vice-President

ISACA Letter for speaking in the 19th March 2019 ,
‘SheLeadsTech’ event on the topic “AI and ML- New
Innovations in Cybersecurity” .



238, ANARKALI COMPLEX
JHANDEWALAN, NEW DELHI – 110055, INDIA
TEL: +91 11 49429733
www.isacanewdelhi.org.in
www.isaca.org

To Whomsoever It May Concern

This is to certify that, *Ms. Navneet Kaur Popli* of MERI College was an invited speaker at ISACA, New Delhi Chapter ‘SheLeadsTech’ event on 19th March 2019 held at C.D. Deshmukh Auditorium, New Delhi.

Topic of Session - “**AI and ML- New Innovations in Cybersecurity**”

ISACA New Delhi Chapter, thank you for sharing your valuable time and knowledge with our members.

For ISACA New Delhi Chapter

A handwritten signature in blue ink, appearing to read "G L Manchanda".

G L Manchanda
Chairman-Academic Affairs

Letter from the Company “Reinova Biz Consultant” for implementation of the framework.

info@reinova.com
www.reinova.com



Dear Navneet,

This is in reference to your research work titled, “Framework to Detect and Mitigate Untraced Polymorphic Shell code”. Our team has gone through your work and we find it quite innovative and worthwhile providing complete security to our organization at low cost. We are ready to integrate your framework in our security systems after relevant tests. All the technical and financial aspects for the same will be finalized in our further sittings.

Regards,



Avinash Verma
Head Business
Reinova Biz Consultant

A-13, 4th floor, Amar Saket, Near Natraj Hotel,
Swargate, Pune - 411042, Maharashtra, India. Phone : +9120 - 24482266

Mail conversations with “virusshare.com” for data acquisition for malicious files(1/3).

7/6/2019

Gmail - Request for dataset



navneet popli <navneetkaurpopli@gmail.com>

Request for dataset

navneet popli <navneetkaurpopli@gmail.com>
To: admin@virusshare.com

Mon, Dec 11, 2017 at 8:05 PM

Dear Sir/Mam,

First of all, I would like to thank you , virus share community to maintain such a nice dataset of malware

I'm Mrs. Navneet Popli and I'm doing research on detection of malware using various machine learning techniques as my Ph.D. thesis. To carry forward my research I need the dataset of virus share so that I can study the behavior of various malware.

It is requested to please provide me the credentials of virusshare.

thankyou

Navneet Kaur Popli

Mail conversations with “virusshare.com” for data acquisition for malicious files(2/3).

7/6/2019

Gmail - My credentials



navneet popli <navneetkaurpopli@gmail.com>

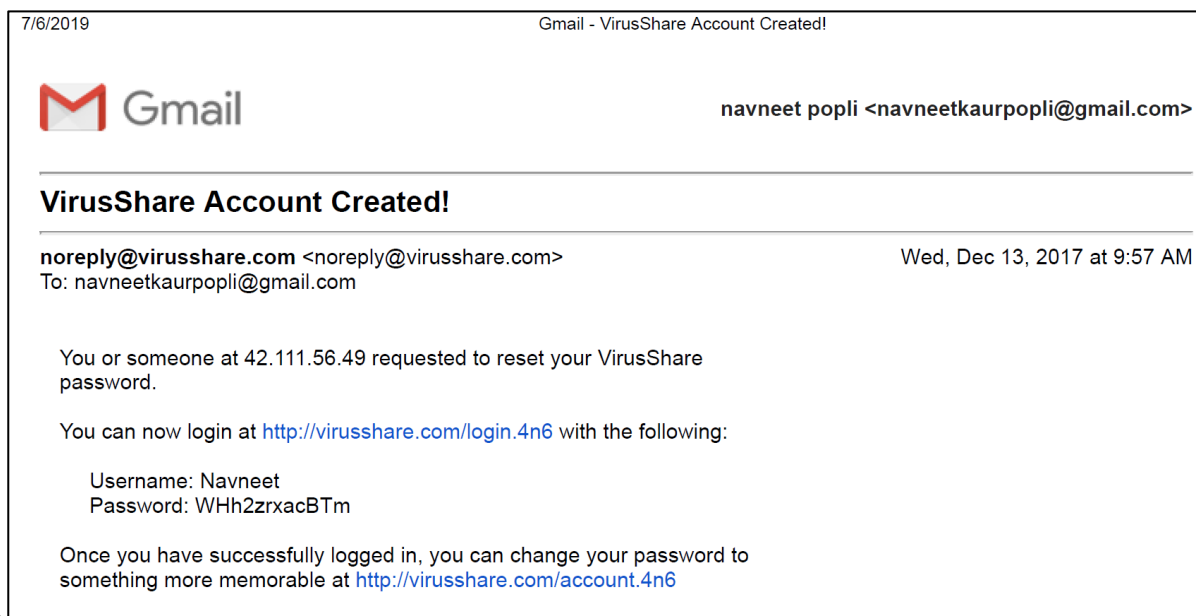
My credentials

navneet popli <navneetkaurpopli@gmail.com>
To: admin@virusshare.com

Mon, Dec 11, 2017 at 8:07 PM

I m Mrs. Navneet Popli
Management Education and Research Institute
GGSIP University
Janak Puri
Delhi
India

Mail conversations with “virusshare.com” for data acquisition for malicious files(3/3)



Reviewer Certificate for International Conference

Sr No : ICIM19/R/078

BABA FARID COLLEGE OF ENGINEERING & TECHNOLOGY **INTERNATIONAL CONFERENCE**

Technical Collaborations



ON **INTELLIGENT MACHINES (ICIM'19)**

15th & 16th March, 2019

Sponsored by



Council of Scientific & Industrial Research



Maharaja Ranjit Singh
Punjab Technical University, Bathinda



Indian Society for Technical Education

Technical Collaborations



Reviewer Certificate

Navneet Kaur Popli

This certificate is awarded to Prof./Dr./Mr./Ms.....

We hereby notify that the person has served as a reviewer of International Conference on Intelligent Machines (ICIM'19) held at Baba Farid College of Engineering & Technology, Bathinda, Punjab, India on 15th & 16th March, 2019. We highly appreciate his/her efforts and contribution to the quality of the work that we have published.


Convener

BABA FARID
GROUP OF INSTITUTIONS

Bathinda (Punjab), www.babafaridgroup.com
Helpline: 1800-180-2002 (Toll-Free)

APPENDIX 3

Screenshots of NoDistribute malware detection of self created malwares

Search or scan a URL, IP address, domain, or file hash

2 / 67

2 engines detected this file

SHA-256 6ab0263f989a1af2e82b7cb94a0a0e3725f39ccd7375622495f081c6b42565d4
File name ImageLine_Keygen.exe
File size 450.88 KB
Last analysis 2018-11-16 10:18:15 UTC
Community score +6

Detection Details Relations Behavior Community

AegisLab	⚠ Virus.Win32.Generic.m2FO	AhnLab-V3	⚠ Unwanted/Win32.Keygen.C1032663
Ad-Aware	✓ Clean	Alibaba	✓ Clean
ALYac	✓ Clean	Arcabit	✓ Clean
Avast	✓ Clean	Avast Mobile Security	✓ Clean
AVG	✓ Clean	Avira	✓ Clean
Babable	✓ Clean	Baidu	✓ Clean
BitDefender	✓ Clean	Bkav	✓ Clean
CMC	✓ Clean	Cylance	✓ Clean
DrWeb	✓ Clean	eGambit	✓ Clean
Emsisoft	✓ Clean	eScan	✓ Clean
F-Prot	✓ Clean	F-Secure	✓ Clean
Jiangmin	✓ Clean	Kaspersky	✓ Clean
Kingsoft	✓ Clean	NANO-Antivirus	✓ Clean
Palo Alto Networks	✓ Clean	Qihoo-360	✓ Clean
Rising	✓ Clean	TACHYON	✓ Clean
Tencent	✓ Clean	TheHacker	✓ Clean
Trustlook	✓ Clean	VIPRE	✓ Clean
ViRobot	✓ Clean	ZoneAlarm	✓ Clean
Zoner	✓ Clean	Symantec Mobile Insight	🚫 Unable to process file type



2 / 67

2 engines detected this file

SHA-256 144afc704c2757dce47fc4114f62ffec82ab2e5cea2144beb8a8e2754e442215
 File name Patch Winrar 64bit Universal Patch.exe
 File size 465.46 KB
 Last analysis 2018-11-11 02:04:18 UTC
 Community score +32



Detection	Details	Relations	Behavior	Community
AhnLab-V3	Unwanted/Win32.HackTool.C1690841			
Ad-Aware	Clean			
Alibaba	Clean			
Antiy-AVL	Clean			
Avast	Clean			
AVG	Clean			
Babable	Clean			
BitDefender	Clean			
CMC	Clean			
Cybereason	Clean			
DrWeb	Clean			
Endgame	Clean			
F-Prot	Clean			
Kingsoft	Clean			
MAX	Clean			
Qihoo-360	Clean			
SUPERAntiSpyware	Clean			
Tencent	Clean			
TotalDefense	Clean			
ViRobot	Clean			
Zoner	Clean			
Bkav	W32.eHeur.Malware09			
AegisLab	Clean			
ALYac	Clean			
Arcabit	Clean			
Avast Mobile Security	Clean			
Avira	Clean			
Baidu	Clean			
ClamAV	Clean			
CrowdStrike Falcon	Clean			
Cylance	Clean			
Emsisoft	Clean			
eScan	Clean			
F-Secure	Clean			
Malwarebytes	Clean			
Microsoft	Clean			
Rising	Clean			
TACHYON	Clean			
TheHacker	Clean			
Trustlook	Clean			
Zillya	Clean			
Symantec Mobile Insight	Unable to process file type			



3 / 65

3 engines detected this file

SHA-256 eb0fcea13c7ecc14b4e809cb6054cec36e7626fbc52ea9a0057613d5a086b8
 File name drw.exe
 File size 32.06 MB
 Last analysis 2018-11-15 16:43:37 UTC
 Community score +25



Detection	Details	Relations	Community
Cyren	W32/GenBl.7164BB44!Olympus	Microsoft	PUA:Win32/FusionCore
VBA32	Trojan.Zpevdo	Ad-Aware	Clean
AegisLab	Clean	AhnLab-V3	Clean
Alibaba	Clean	ALYac	Clean
Antiy-AVL	Clean	Arcabit	Clean
Avast	Clean	Avast Mobile Security	Clean
AVG	Clean	Avira	Clean
Babable	Clean	Baidu	Clean
BitDefender	Clean	Bkav	Clean
CAT-QuickHeal	Clean	ClamAV	Clean
CMC	Clean	CrowdStrike Falcon	Clean
Cybereason	Clean	Cylance	Clean
DrWeb	Clean	Emsisoft	Clean
Endgame	Clean	eScan	Clean
ESET-NOD32	Clean	F-Prot	Clean
F-Secure	Clean	Fortinet	Clean
GData	Clean	Ikarus	Clean
Jiangmin	Clean	K7AntiVirus	Clean
K7GW	Clean	Kaspersky	Clean
Kingsoft	Clean	Malwarebytes	Clean
MAX	Clean	McAfee	Clean
McAfee-GW-Edition	Clean	NANO-Antivirus	Clean
Palo Alto Networks	Clean	Panda	Clean
Qihoo-360	Clean	Rising	Clean
SentinelOne	Clean	Sophos AV	Clean
Sophos ML	Clean	SUPERAntiSpyware	Clean
Symantec	Clean	TACHYON	Clean
Tencent	Clean	TheHacker	Clean
TrendMicro	Clean	TrendMicro-HouseCall	Clean
Trustlook	Clean	ViRobot	Clean
Webroot	Clean	Yandex	Clean
Zillya	Clean	ZoneAlarm	Clean
Zoner	Clean	eGambit	Timeout
Symantec Mobile Insight	Unable to process file type		



Search or scan a URL, IP address, domain, or file hash



Sign in



1 / 65

1 engines detected this file

SHA-256 6e9401a3dfdba948013240e1c8cef03a5f50a5e7bfa347fa68b07b9b5ea9b7e2
 File name Keygen.exe
 File size 234 KB
 Last analysis 2018-11-11 23:29:53 UTC
 Community score +45

Detection Details Relations Behavior Community 3

Zoner	PUA.Malicious	Alibaba	Clean
Avast Mobile Security	Clean	Avira	Clean
Babable	Clean	Baidu	Clean
Bkav	Clean	CMC	Clean
F-Prot	Clean	Jiangmin	Clean
Kaspersky	Clean	Kingsoft	Clean
Malwarebytes	Clean	Microsoft	Clean
Palo Alto Networks	Clean	Qihoo-360	Clean
SUPERAntiSpyware	Clean	TACHYON	Clean
Tencent	Clean	TheHacker	Clean
TrendMicro	Clean	TrendMicro-HouseCall	Clean



Search or scan a URL, IP address, domain, or file hash



Sign in



1 / 67

1 engines detected this file

SHA-256 7a0ebd5cb9e34a535c0cd3575c1bd8231099869972c9d07e4a21df06c5b3316a
 File name xf-adsk2018_x64v3.exe
 File size 851.5 KB
 Last analysis 2018-11-15 07:24:20 UTC
 Community score -5

Detection Details Relations Community 1

Zoner	Trojan.Application	AegisLab	Clean
Alibaba	Clean	ALYac	Clean
Avast Mobile Security	Clean	Avira	Clean
Babable	Clean	Baidu	Clean
Bkav	Clean	CMC	Clean
DrWeb	Clean	eGambit	Clean
F-Prot	Clean	Kaspersky	Clean
Kingsoft	Clean	MAX	Clean
NANO-Antivirus	Clean	Qihoo-360	Clean
Rising	Clean	SUPERAntiSpyware	Clean
TACHYON	Clean	Tencent	Clean
TheHacker	Clean	Trustlook	Clean
VBA32	Clean	ViRobot	Clean



7 engines detected this file

SHA-256 a7551b209c18c504ea9d8caa89bea2e0b9415911b850676714bc752a073302f5
 File name 64bit Patch build 6.exe
 File size 1.94 MB
 Last analysis 2018-07-31 19:26:22 UTC

7 / 66



Detection	Details	Relations	Behavior	Community
Comodo	CloudScanner.Trojan.Gen			Panda
Rising	Malware.Undefined!8.C (CLOUD)			PUP/Patcher
Sophos ML	heuristic			SentinelOne
Webroot	W32.Hack.Tool			static engine - malicious
AegisLab	Clean			TrendMicro-HouseCall
ALYac	Clean			Suspicious_GEN.F47V0613
Arcabit	Clean			Ad-Aware
Avast Mobile Security	Clean			Clean
Avira	Clean			AhnLab-V3
Babable	Clean			Clean
BitDefender	Clean			Antiy-AVL
CAT-QuickHeal	Clean			Clean
CMC	Clean			Avast
Cybereason	Clean			Clean
Cyren	Clean			AVG
eGambit	Clean			Clean
Endgame	Clean			AVware
ESET-NOD32	Clean			Clean
F-Secure	Clean			Baidu
GData	Clean			Clean
Jiangmin	Clean			Bkav
K7GW	Clean			Clean
Kingsoft	Clean			ClamAV
MAX	Clean			Clean
McAfee-GW-Edition	Clean			CrowdStrike Falcon
NANO-Antivirus	Clean			Clean
Qihoo-360	Clean			Cylance
SUPERAntiSpyware	Clean			Clean
TACHYON	Clean			DrWeb
TheHacker	Clean			Clean
VBA32	Clean			Emsisoft
ViRobot	Clean			Clean
ZoneAlarm	Clean			eScan
Alibaba	Unable to process file type			Clean
Trustlook	Unable to process file type			F-Prot
				Clean
				Fortinet
				Clean
				Ikarus
				Clean
				K7AntiVirus
				Clean
				Kaspersky
				Clean
				Malwarebytes
				Clean
				McAfee
				Clean
				Microsoft
				Clean
				Palo Alto Networks
				Clean
				Sophos AV
				Clean
				Symantec
				Clean
				Tencent
				Clean
				TrendMicro
				Clean
				VIPRE
				Clean
				Yandex
				Clean
				Zoner
				Clean
				Symantec Mobile Insight
				Unable to process file type

Search or scan a URL, IP address, domain, or file hash

No engines detected this file

SHA-256: 9be7470ec58e079d86b72f23a92e4785276fe50df49e607e3046e6be7a8daf22
 File name: amtlib
 File size: 2.15 MB
 Last analysis: 2018-10-05 20:05:28 UTC
 Community score: +46

0 / 66

Detection | Details | Relations | Community

Ad-Aware	Clean	AegisLab	Clean
AhnLab-V3	Clean	Alibaba	Clean
ALYac	Clean	Antiy-AVL	Clean
Arcabit	Clean	Avast	Clean
Avast Mobile Security	Clean	AVG	Clean
Avira	Clean	Avira	Clean

Search or scan a URL, IP address, domain, or file hash

4 engines detected this file

SHA-256: d2bef451a44457ef4b1da38982f568e1e75402fbd2fedc6eaa5f761cd6a5e751
 File name: WAT Fix.exe
 File size: 686.41 KB
 Last analysis: 2018-11-15 14:34:23 UTC
 Community score: +48

4 / 63

Detection | Details | Relations | Behavior | Community

Webroot	W32.Dropper.Gen	Yandex	Trojan.DR.Dunik!UzRc3m8wN34
ZoneAlarm	HackTool.Win32.KMSAutc.ad	Zoner	Trojan.Dunik
Alibaba	Clean	Avast Mobile Security	Clean
Babable	Clean	Baidu	Clean
Bkav	Clean	ClamAV	Clean
CMC	Clean	eGambit	Clean
Fortinet	Clean	NANO-Antivirus	Clean
Palo Alto Networks	Clean	SentinelOne	Clean
TACHYON	Clean	TheHacker	Clean



7 engines detected this file



7 / 65

SHA-256 6b216cbf7e4d01d49e3ccb2b48aa7758cbecef064dbff1ecbff2e674bbb6208a
 File name DTCommonRes.dll
 File size 5.48 MB
 Last analysis 2018-10-25 16:23:05 UTC
 Community score +90

Detection	Details	Relations	Community
Bkav	W32.eHeur.Virus02		ClamAV Win.Trojan.Sality-46407
Endgame	malicious (high confidence)		ESET-NOD32 a variant of Win32/HackTool.Crack.DM
Kingsoft	Win32.Sality.G.122880		TheHacker Trojan/HackTool.Crack.dm
Zillya	Trojan.Crack.Win32.1		Ad-Aware Clean
AegisLab	Clean		AhnLab-V3 Clean
Alibaba	Clean		ALYac Clean
Antiy-AVL	Clean		Arcabit Clean
Avast	Clean		Avast Mobile Security Clean
AVG	Clean		Avira Clean
Babable	Clean		Baidu Clean
BitDefender	Clean		CAT-QuickHeal Clean
CMC	Clean		CrowdStrike Falcon Clean
Cylance	Clean		Cyren Clean
DrWeb	Clean		eGambit Clean
Emsisoft	Clean		eScan Clean
F-Prot	Clean		F-Secure Clean
Fortinet	Clean		GData Clean
Ikarus	Clean		Jiangmin Clean
K7AntiVirus	Clean		K7GW Clean
Kaspersky	Clean		Malwarebytes Clean
MAX	Clean		McAfee Clean
McAfee-GW-Edition	Clean		Microsoft Clean
NANO-Antivirus	Clean		Palo Alto Networks Clean
Panda	Clean		Qihoo-360 Clean
Rising	Clean		SentinelOne Clean
Sophos AV	Clean		Sophos ML Clean
SUPERAntiSpyware	Clean		Symantec Clean
TACHYON	Clean		Tencent Clean
TrendMicro	Clean		TrendMicro-HouseCall Clean
Trustlook	Clean		VBA32 Clean
ViRobot	Clean		Webroot Clean
Yandex	Clean		ZoneAlarm Clean
Zoner	Clean		Cybereason Unable to process file type
Symantec Mobile Insight	Unable to process file type		



2 / 65

2 engines detected this file

SHA-256 3db13c769f4bab69ab8b0ab26059a3072fc4f3f426920f051eacf24cc38f5667
 File name KMSpico_setup.exe
 File size 2.72 MB
 Last analysis 2018-11-09 03:53:22 UTC
 Community score +32

- Detection
- Details
- Relations
- Behavior
- Community

Webroot	W32.Hacktool.Kms	Yandex	Riskware.HackTool!TCbRy015uVg
Ad-Aware	Clean	AegisLab	Clean
Alibaba	Clean	ALYac	Clean
Avast Mobile Security	Clean	Avira	Clean
Babable	Clean	Baidu	Clean
Bkav	Clean	CMC	Clean
CrowdStrike Falcon	Clean	Cylance	Clean
eScan	Clean	Fortinet	Clean
Jiangmin	Clean	Kaspersky	Clean
Kingsoft	Clean	Malwarebytes	Clean
MAX	Clean	Palo Alto Networks	Clean
SUPERAntiSpyware	Clean	TACHYON	Clean
Tencent	Clean	TheHacker	Clean
TrendMicro-HouseCall	Clean	Trustlook	Clean
VBA32	Clean	ViRobot	Clean
Zillya	Clean	ZoneAlarm	Clean



Search or scan a URL, IP address, domain, or file hash



Sign in



4 engines detected this file

SHA-256 99f0875ce316761fe9dde48b1313486ba59e257f2db08d8040bee5b07067010c
 File name cr-piriform.exe
 File size 440 KB
 Last analysis 2018-11-13 09:09:41 UTC
 Community score +39

4 / 65

Detection	Details	Relations	Behavior	Community
Ad-Aware	Trojan.GenericKD.12615974		ALYac	Trojan.GenericKD.12615974
Yandex	PUPAgent!		Zillya	Trojan.GenericKD.Win32.42016
AegisLab	Clean		AhnLab-V3	Clean
Alibaba	Clean		Avast Mobile Security	Clean
Avira	Clean		Babable	Clean
Baidu	Clean		Bkav	Clean
CMC	Clean		CrowdStrike Falcon	Clean
Cylance	Clean		DrWeb	Clean
Emsisoft	Clean		F-Prot	Clean
Jiangmin	Clean		Kaspersky	Clean
Kingsoft	Clean		Malwarebytes	Clean
NANO-Antivirus	Clean		Panda	Clean
Qihoo-360	Clean		Rising	Clean
SUPERAntiSpyware	Clean		TACHYON	Clean
Tencent	Clean		TheHacker	Clean
TotalDefense	Clean		VIRobot	Clean
Webroot	Clean		ZoneAlarm	Clean
Zoner	Clean		Symantec Mobile Insight	Unable to process file type



Search or scan a URL, IP address, domain, or file hash



Sign in



7 engines detected this file

SHA-256 2f2aba1e074f5f4baa08b524875461889f8f04d4ffc43972ac212e286022ab94
 File name Windows Loader.exe
 File size 3.83 MB
 Last analysis 2018-11-16 15:01:55 UTC
 Community score +1162

7 / 67

Detection	Details	Relations	Behavior	Community
Ad-Aware	Application.Hacktool.PR		AegisLab	Hacktool.Win32.KMSAuto.3tc
Antiy-AVL	Worm/Win32.AutoRun		Arcabit	Application.Hacktool.PR
Avira	SPR/WinLoader.40210		BitDefender	Application.Hacktool.PR
ZoneAlarm	HackTool.Win32.KMSAuto.bu		AhnLab-V3	Clean
Alibaba	Clean		ALYac	Clean
Avast	Clean		Avast Mobile Security	Clean
AVG	Clean		Babable	Clean
Baidu	Clean		Bkav	Clean
CMC	Clean		CrowdStrike Falcon	Clean
DrWeb	Clean		eGambit	Clean
F-Secure	Clean		Fortinet	Clean
Jiangmin	Clean		Kingsoft	Clean
NANO-Antivirus	Clean		Palo Alto Networks	Clean
Qihoo-360	Clean		Rising	Clean



One engine detected this file



SHA-256 a44f2d1a832aa9aba551d552c69f44da7b02ff7a126c3a6eb4b122a32265c34b
 File name branding.dll
 File size 51.71 KB
 Last analysis 2018-09-19 08:55:26 UTC

1 / 66

Detection	Details	Relations	Community
TheHacker	Trojan/MMM.avy		
Ad-Aware	Clean		
AegisLab	Clean	AhnLab-V3	Clean
ALYac	Clean	Antiy-AVL	Clean
Arcabit	Clean	Avast	Clean
Avast Mobile Security	Clean	AVG	Clean
Avira	Clean	AVware	Clean
Babable	Clean	Baidu	Clean
BitDefender	Clean	Bkav	Clean
CAT-QuickHeal	Clean	ClamAV	Clean
CMC	Clean	Comodo	Clean
CrowdStrike Falcon	Clean	Cylance	Clean
Cyren	Clean	DrWeb	Clean
Emsisoft	Clean	Endgame	Clean
eScan	Clean	ESET-NOD32	Clean
F-Prot	Clean	F-Secure	Clean
Fortinet	Clean	GData	Clean
Ikarus	Clean	Jiangmin	Clean
K7AntiVirus	Clean	K7GW	Clean
Kaspersky	Clean	Kingsoft	Clean
Malwarebytes	Clean	MAX	Clean
McAfee	Clean	McAfee-GW-Edition	Clean
Microsoft	Clean	NANO-Antivirus	Clean
Palo Alto Networks	Clean	Panda	Clean
Qihoo-360	Clean	Rising	Clean
SentinelOne	Clean	Sophos AV	Clean
Sophos ML	Clean	SUPERAntiSpyware	Clean
Symantec	Clean	TACHYON	Clean
Tencent	Clean	TotalDefense	Clean
TrendMicro	Clean	TrendMicro-HouseCall	Clean
VBA32	Clean	VIPRE	Clean
ViRobot	Clean	Webroot	Clean
Yandex	Clean	Zillya	Clean
ZoneAlarm	Clean	Zoner	Clean



4 engines detected this file

SHA-256: bffc3007747caf507636806b055a5db82b0d05e01a0a0d63da2a1439b02f1581
 File name: ccleaner-professional_5-44-6575_fr_433011.exe
 File size: 15.26 MB
 Last analysis: 2018-11-03 18:43:03 UTC
 Community score: -4

4 / 64

Detection	Details	Relations	Behavior	Community
ESET-NOD32	Win32/Bundled.Toolbar.Google.D potentially unsafe			K7AntiVirus Unwanted-Program (004b953d1)
K7GW	Unwanted-Program (004b953d1)			Yandex Trojan.Droma!OipNvZuV2DQ
Ad-Aware	Clean			AegisLab Clean
AhnLab-V3	Clean			Alibaba Clean
ALYac	Clean			Antiy-AVL Clean
Arcabit	Clean			Avast Clean
Avast Mobile Security	Clean			AVG Clean
Avira	Clean			Babable Clean
Baidu	Clean			BitDefender Clean
Bkav	Clean			CAT-QuickHeal Clean
ClamAV	Clean			CMC Clean
CrowdStrike Falcon	Clean			Cybereason Clean
Cylance	Clean			Cyren Clean
DrWeb	Clean			eGambit Clean
Emsisoft	Clean			Endgame Clean
eScan	Clean			F-Prot Clean
F-Secure	Clean			Fortinet Clean
GData	Clean			Ikarus Clean
Jiangmin	Clean			Kaspersky Clean
Kingsoft	Clean			Malwarebytes Clean
MAX	Clean			McAfee-GW-Edition Clean
Microsoft	Clean			NANO-Antivirus Clean
Palo Alto Networks	Clean			Panda Clean
Qihoo-360	Clean			Rising Clean
SentinelOne	Clean			Sophos AV Clean
Sophos ML	Clean			SUPERAntiSpyware Clean
Symantec	Clean			TACHYON Clean



2 engines detected this file



2 / 66

SHA-256 4a3e883249c4e6514987a0b21433548f7bda8bf419b9e9896792ecd8929cb8f4
 File name BRD.dll
 File size 105 KB
 Last analysis 2018-10-24 11:04:05 UTC
 Community score +18

- Detection
- Details
- Relations
- Community 6

AegisLab	Troj.W32.Agent.toYH	Antiy-AVL	Trojan/Win32.TSGeneric
Ad-Aware	Clean	AhnLab-V3	Clean
Alibaba	Clean	ALYac	Clean
Arcabit	Clean	Avast	Clean
Avast Mobile Security	Clean	AVG	Clean
Avira	Clean	Babable	Clean
Baidu	Clean	BitDefender	Clean
ClamAV	Clean	CMC	Clean
Comodo	Clean	DrWeb	Clean
eGambit	Clean	Emsisoft	Clean
eScan	Clean	F-Prot	Clean
GData	Clean	Kingssoft	Clean
Malwarebytes	Clean	McAfee	Clean
McAfee-GW-Edition	Clean	Microsoft	Clean
NANO-Antivirus	Clean	Palo Alto Networks	Clean
Panda	Clean	Qihoo-360	Clean
Rising	Clean	SUPERAntiSpyware	Clean
Symantec	Clean	TACHYON	Clean
Tencent	Clean	TheHacker	Clean
TotalDefense	Clean	Trustlook	Clean
VIPRE	Clean	ViRobot	Clean
Webroot	Clean	Zoner	Clean

APPENDIX 3 : PROGRAMMING CODES USED FOR COMPLETE

EXECUTION

Static analysis code

Training code

```
import pandas as pd
import numpy as np
import pickle
import sklearn.ensemble as ske
from sklearn import cross_validation, tree, linear_model
from sklearn.feature_selection import SelectFromModel
from sklearn.externals import joblib
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix

data = pd.read_csv('data.csv', sep='|')
X = data.drop(['Name', 'md5', 'legitimate'], axis=1).values
y = data['legitimate'].values

print('Researching important feature based on %i total features\n' % X.shape[1])

# Feature selection using Trees Classifier
fsel = ske.ExtraTreesClassifier().fit(X, y)
model = SelectFromModel(fsel, prefit=True)
X_new = model.transform(X)
nb_features = X_new.shape[1]

X_train, X_test, y_train, y_test = cross_validation.train_test_split(X_new, y, test_size=0.2)

features = []

print('%i features identified as important:' % nb_features)

indices = np.argsort(fsel.feature_importances_)[::-1][:nb_features]
for f in range(nb_features):
    print("%d. feature %s (%f)" % (f + 1, data.columns[2+indices[f]], fsel.feature_importances_[indices[f]]))

# XXX : take care of the feature order
for f in sorted(np.argsort(fsel.feature_importances_)[::-1][:nb_features]):
    features.append(data.columns[2+f])
```



```

#Algorithm comparison
algorithms = {
    "DecisionTree": tree.DecisionTreeClassifier(max_depth=10),
    "RandomForest": ske.RandomForestClassifier(n_estimators=50),
    "GradientBoosting": ske.GradientBoostingClassifier(n_estimators=50),
    "AdaBoost": ske.AdaBoostClassifier(n_estimators=100),
    "GNB": GaussianNB()
}

results = {}
print("\nNow testing algorithms")
for algo in algorithms:
    clf = algorithms[algo]
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    print("%s : %f %%" % (algo, score*100))
    results[algo] = score

winner = max(results, key=results.get)
print('\nWinner algorithm is %s with a %f %% success' % (winner, results[winner]*100))

# Save the algorithm and the feature list for later predictions
print('Saving algorithm and feature list in classifier directory...')
joblib.dump(algorithms[winner], 'classifier/classifier.pkl')
open('classifier/features.pkl', 'wb').write(pickle.dumps(features))
print('Saved')

# Identify false and true positive rates
clf = algorithms[winner]
res = clf.predict(X_test)
mt = confusion_matrix(y_test, res)
print("confusion matrix %s" % mt)
print("False positive rate : %f %%" % ((mt[0][1] / float(sum(mt[0]))) * 100))
print('False negative rate : %f %%' % ((mt[1][0] / float(sum(mt[1])) * 100))

```

Static analysis prediction code

```

#!/usr/bin/python2
import pefile
import os
import array
import math
import pickle
from sklearn.externals import joblib
import sys
import argparse

def get_entropy(data):
    if len(data) == 0:
        return 0.0

```

```

occurrences = array.array('L', [0]*256)
for x in data:
    occurrences[x if isinstance(x, int) else ord(x)] += 1

entropy = 0
for x in occurrences:
    if x:
        p_x = float(x) / len(data)
        entropy -= p_x*math.log(p_x, 2)

return entropy

def get_resources(pe):
    """Extract resources :
    [entropy, size]"""
    resources = []
    if hasattr(pe, 'DIRECTORY_ENTRY_RESOURCE'):
        try:
            for resource_type in pe.DIRECTORY_ENTRY_RESOURCE.entries:
                if hasattr(resource_type, 'directory'):
                    for resource_id in resource_type.directory.entries:
                        if hasattr(resource_id, 'directory'):
                            for resource_lang in resource_id.directory.entries:
                                data = pe.get_data(resource_lang.data.struct.OffsetToData,
resource_lang.data.struct.Size)
                                size = resource_lang.data.struct.Size
                                entropy = get_entropy(data)

                                resources.append([entropy, size])
        except Exception as e:
            return resources
    return resources

def get_version_info(pe):
    """Return version infos"""
    res = {}
    for fileinfo in pe.FileInfo:
        if fileinfo.Key == 'StringFileInfo':
            for st in fileinfo.StringTable:
                for entry in st.entries.items():
                    res[entry[0]] = entry[1]
        if fileinfo.Key == 'VarFileInfo':
            for var in fileinfo.Var:
                res[var.entry.items()[0][0]] = var.entry.items()[0][1]
    if hasattr(pe, 'VS_FIXEDFILEINFO'):
        res['flags'] = pe.VS_FIXEDFILEINFO.FileFlags
        res['os'] = pe.VS_FIXEDFILEINFO.FileOS
        res['type'] = pe.VS_FIXEDFILEINFO.FileType
        res['file_version'] = pe.VS_FIXEDFILEINFO.FileVersionLS
        res['product_version'] = pe.VS_FIXEDFILEINFO.ProductVersionLS
        res['signature'] = pe.VS_FIXEDFILEINFO.Signature
        res['struct_version'] = pe.VS_FIXEDFILEINFO.StrucVersion

```

```
return res
```

```
def extract_infos(fpath):
    res = {}
    pe = pefile.PE(fpath)
    res['Machine'] = pe.FILE_HEADER.Machine
    res['SizeOfOptionalHeader'] = pe.FILE_HEADER.SizeOfOptionalHeader
    res['Characteristics'] = pe.FILE_HEADER.Characteristics
    res['MajorLinkerVersion'] = pe.OPTIONAL_HEADER.MajorLinkerVersion
    res['MinorLinkerVersion'] = pe.OPTIONAL_HEADER.MinorLinkerVersion
    res['SizeOfCode'] = pe.OPTIONAL_HEADER.SizeOfCode
    res['SizeOfInitializedData'] = pe.OPTIONAL_HEADER.SizeOfInitializedData
    res['SizeOfUninitializedData'] = pe.OPTIONAL_HEADER.SizeOfUninitializedData
    res['AddressOfEntryPoint'] = pe.OPTIONAL_HEADER.AddressOfEntryPoint
    res['BaseOfCode'] = pe.OPTIONAL_HEADER.BaseOfCode
    try:
        res['BaseOfData'] = pe.OPTIONAL_HEADER.BaseOfData
    except AttributeError:
        res['BaseOfData'] = 0
    res['ImageBase'] = pe.OPTIONAL_HEADER.ImageBase
    res['SectionAlignment'] = pe.OPTIONAL_HEADER.SectionAlignment
    res['FileAlignment'] = pe.OPTIONAL_HEADER.FileAlignment
    res['MajorOperatingSystemVersion'] = pe.OPTIONAL_HEADER.MajorOperatingSystemVersion
    res['MinorOperatingSystemVersion'] = pe.OPTIONAL_HEADER.MinorOperatingSystemVersion
    res['MajorImageVersion'] = pe.OPTIONAL_HEADER.MajorImageVersion
    res['MinorImageVersion'] = pe.OPTIONAL_HEADER.MinorImageVersion
    res['MajorSubsystemVersion'] = pe.OPTIONAL_HEADER.MajorSubsystemVersion
    res['MinorSubsystemVersion'] = pe.OPTIONAL_HEADER.MinorSubsystemVersion
    res['SizeOfImage'] = pe.OPTIONAL_HEADER.SizeOfImage
    res['SizeOfHeaders'] = pe.OPTIONAL_HEADER.SizeOfHeaders
    res['Checksum'] = pe.OPTIONAL_HEADER.CheckSum
    res['Subsystem'] = pe.OPTIONAL_HEADER.Subsystem
    res['DllCharacteristics'] = pe.OPTIONAL_HEADER.DllCharacteristics
    res['SizeOfStackReserve'] = pe.OPTIONAL_HEADER.SizeOfStackReserve
    res['SizeOfStackCommit'] = pe.OPTIONAL_HEADER.SizeOfStackCommit
    res['SizeOfHeapReserve'] = pe.OPTIONAL_HEADER.SizeOfHeapReserve
    res['SizeOfHeapCommit'] = pe.OPTIONAL_HEADER.SizeOfHeapCommit
    res['LoaderFlags'] = pe.OPTIONAL_HEADER.LoaderFlags
    res['NumberOfRvaAndSizes'] = pe.OPTIONAL_HEADER.NumberOfRvaAndSizes

    # Sections
    res['SectionsNb'] = len(pe.sections)
    entropy = map(lambda x:x.get_entropy(), pe.sections)
    res['SectionsMeanEntropy'] = sum(entropy)/float(len(entropy))
    res['SectionsMinEntropy'] = min(entropy)
    res['SectionsMaxEntropy'] = max(entropy)
    raw_sizes = map(lambda x:x.SizeOfRawData, pe.sections)
    res['SectionsMeanRawsize'] = sum(raw_sizes)/float(len(raw_sizes))
    res['SectionsMinRawsize'] = min(raw_sizes)
    res['SectionsMaxRawsize'] = max(raw_sizes)
    virtual_sizes = map(lambda x:x.Misc_VirtualSize, pe.sections)
    res['SectionsMeanVirtualsize'] = sum(virtual_sizes)/float(len(virtual_sizes))
    res['SectionsMinVirtualsize'] = min(virtual_sizes)
```

```

res['SectionMaxVirtualSize'] = max(virtual_sizes)

#Imports
try:
    res['ImportsNbDLL'] = len(pe.DIRECTORY_ENTRY_IMPORT)
    imports = sum([x.imports for x in pe.DIRECTORY_ENTRY_IMPORT], [])
    res['ImportsNb'] = len(imports)
    res['ImportsNbOrdinal'] = len(filter(lambda x:x.name is None, imports))
except AttributeError:
    res['ImportsNbDLL'] = 0
    res['ImportsNb'] = 0
    res['ImportsNbOrdinal'] = 0

#Exports
try:
    res['ExportNb'] = len(pe.DIRECTORY_ENTRY_EXPORT.symbols)
except AttributeError:
    # No export
    res['ExportNb'] = 0

#Resources
resources= get_resources(pe)
res['ResourcesNb'] = len(resources)
if len(resources)> 0:
    entropy = map(lambda x:x[0], resources)
    res['ResourcesMeanEntropy'] = sum(entropy)/float(len(entropy))
    res['ResourcesMinEntropy'] = min(entropy)
    res['ResourcesMaxEntropy'] = max(entropy)
    sizes = map(lambda x:x[1], resources)
    res['ResourcesMeanSize'] = sum(sizes)/float(len(sizes))
    res['ResourcesMinSize'] = min(sizes)
    res['ResourcesMaxSize'] = max(sizes)
else:
    res['ResourcesNb'] = 0
    res['ResourcesMeanEntropy'] = 0
    res['ResourcesMinEntropy'] = 0
    res['ResourcesMaxEntropy'] = 0
    res['ResourcesMeanSize'] = 0
    res['ResourcesMinSize'] = 0
    res['ResourcesMaxSize'] = 0

# Load configuration size
try:
    res['LoadConfigurationSize'] = pe.DIRECTORY_ENTRY_LOAD_CONFIG.struct.Size
except AttributeError:
    res['LoadConfigurationSize'] = 0

# Version configuration size
try:
    version_infos = get_version_info(pe)
    res['VersionInformationSize'] = len(version_infos.keys())
except AttributeError:
    res['VersionInformationSize'] = 0

```

```

return res

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Detect malicious files')
    parser.add_argument('FILE', help='File to be tested')
    args = parser.parse_args()
    # Load classifier
    clf = joblib.load(os.path.join(
        os.path.dirname(os.path.realpath(__file__)),
        'classifier/classifier.pkl'
    ))
    features = pickle.loads(open(os.path.join(
        os.path.dirname(os.path.realpath(__file__)),
        'classifier/features.pkl',
        'r').read()
    ))

    data = extract_infos(args.FILE)

    pe_features = map(lambda x:data[x], features)

    res= clf.predict_proba([pe_features])[0]

    print( res)

```

Behaviour analysis code

```

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer =TfidfVectorizer(min_df=5,
                             max_df = 0.8,
                             sublinear_tf=True,
                             use_idf=True,decode_error='ignore')

files = [open("reports_mix/report ({}).mist".format(x),'r') for x in range(1,200)]
corpus = []
for x in range (0,197):
    {
        corpus.append(files[x].read())
    }

x = vectorizer.fit_transform(corpus)

y=[]
for i in range(1,100):
    y.append(0);
for i in range(101,199):

```

```

y.append(1);

from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.6, random_state=42)
from sklearn.neighbors import KNeighborsClassifier
import logging
import numpy as np
from optparse import OptionParser
import sys
from time import time
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import RidgeClassifier
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils.extmath import density
from sklearn import metrics

def benchmark(clf):
    print('_' * 80)
    print("Training: ")
    print(clf)
    t0 = time()
    clf.fit(X_train, y_train)
    train_time = time() - t0
    print("train time: %0.3fs" % train_time)

    t0 = time()
    pred = clf.predict(X_test)
    test_time = time() - t0
    print("test time: %0.3fs" % test_time)

    score = metrics.accuracy_score(y_test, pred)
    print("accuracy: %0.3f" % score)

    if hasattr(clf, 'coef_'):
        print("dimensionality: %d" % clf.coef_.shape[1])
        print("density: %f" % density(clf.coef_))

```

```

print("classification report:")
print(metrics.classification_report(y_test, pred
    ))

print("confusion matrix:")
print(metrics.confusion_matrix(y_test, pred))

print()
clf_descr = str(clf).split('(')[0]
return clf_descr, score, train_time, test_time

results = []
for clf, name in (
    (RidgeClassifier(tol=1e-2, solver="lsqr"), "Ridge Classifier"),
    (Perceptron(n_iter=50), "Perceptron"),
    (PassiveAggressiveClassifier(n_iter=50), "Passive-Aggressive"),
    (KNeighborsClassifier(n_neighbors=10), "kNN"),
    (RandomForestClassifier(n_estimators=100), "Random forest")):
    print('=' * 80)
    print(name)
    results.append(benchmark(clf))

for penalty in ["l2", "l1"]:
    print('=' * 80)
    print("%s penalty" % penalty.upper())
    # Train Liblinear model
    results.append(benchmark(LinearSVC(penalty=penalty, dual=False,
        tol=1e-3)))

    # Train SGD model
    results.append(benchmark(SGDClassifier(alpha=.0001, n_iter=50,
        penalty=penalty)))

# Train SGD with Elastic Net penalty
print('=' * 80)
print("Elastic-Net penalty")
results.append(benchmark(SGDClassifier(alpha=.0001, n_iter=50,
    penalty="elasticnet")))

# Train NearestCentroid without threshold
print('=' * 80)
print("NearestCentroid (aka Rocchio classifier)")
results.append(benchmark(NearestCentroid()))

# Train sparse Naive Bayes classifiers
print('=' * 80)
print("Naive Bayes")
results.append(benchmark(MultinomialNB(alpha=.01)))

```

```

results.append(benchmark(BernoulliNB(alpha=.01)))

print('=' * 80)
print("LinearSVC with L1-based feature selection")
# The smaller C, the stronger the regularization.
# The more regularization, the more sparsity.
results.append(benchmark(Pipeline([
    ('feature_selection', SelectFromModel(LinearSVC(penalty="l1", dual=False,
                                                    tol=1e-3))),
    ('classification', LinearSVC(penalty="l2"))]))))

# make some plots

indices = np.arange(len(results))

results = [[x[i] for x in results] for i in range(4)]

clf_names, score, training_time, test_time = results
training_time = np.array(training_time) / np.max(training_time)
test_time = np.array(test_time) / np.max(test_time)

plt.figure(figsize=(12, 8))
plt.title("Score")
plt.barh(indices, score, .2, label="score", color='navy')
plt.barh(indices + .3, training_time, .2, label="training time",
          color='c')
plt.barh(indices + .6, test_time, .2, label="test time", color='darkorange')
plt.yticks(())
plt.legend(loc='best')
plt.subplots_adjust(left=.25)
plt.subplots_adjust(top=.95)
plt.subplots_adjust(bottom=.05)

for i, c in zip(indices, clf_names):
    plt.text(-.3, i, c)

plt.show()

```

Snort analysis code

```

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df=5,
                             max_df = 0.8,
                             sublinear_tf=True,
                             use_idf=True, decode_error='ignore')

files = [open("/home/varnit/projects/shellcode/snort_mix_reports/{}.txt".format(x), 'r') for x in range(1,200)]
corpus = []

```



```

for x in range (0,197):
    {
        corpus.append(files[x].read())
    }

```

```

x = vectorizer.fit_transform(corpus)
y=[]
for i in range(1,100):
    y.append(0);

```

```

for i in range(101,199):
    y.append(1);

```

```

from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.6, random_state=42)

```

```

import logging
import numpy as np
from optparse import OptionParser
import sys
from time import time
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import HashingVectorizer
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.linear_model import RidgeClassifier
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import PassiveAggressiveClassifier
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neighbors import NearestCentroid
from sklearn.ensemble import RandomForestClassifier
from sklearn.utils.extmath import density
from sklearn import metrics

```

```

def benchmark(clf):
    print('_' * 80)
    print("Training: ")
    print(clf)
    t0 = time()
    clf.fit(X_train, y_train)
    train_time = time() - t0
    print("train time: %0.3fs" % train_time)

```

```

t0 = time()

```

```

pred = clf.predict(X_test)
test_time = time() - t0
print("test time: %0.3fs" % test_time)

score = metrics.accuracy_score(y_test, pred)
print("accuracy: %0.3f" % score)

if hasattr(clf, 'coef_'):
    print("dimensionality: %d" % clf.coef_.shape[1])
    print("density: %f" % density(clf.coef_))

print("classification report:")
print(metrics.classification_report(y_test, pred
    ))

print("confusion matrix:")
print(metrics.confusion_matrix(y_test, pred))

print()
clf_descr = str(clf).split('(')[0]
return clf_descr, score, train_time, test_time

```

```

results = []
for clf, name in (
    (RidgeClassifier(tol=1e-2, solver="lsqr"), "Ridge Classifier"),
    (Perceptron(n_iter=50), "Perceptron"),
    (PassiveAggressiveClassifier(n_iter=50), "Passive-Aggressive"),
    (KNeighborsClassifier(n_neighbors=10), "kNN"),
    (RandomForestClassifier(n_estimators=100), "Random forest")):
    print('=' * 80)
    print(name)
    results.append(benchmark(clf))

#for penalty in ["l2", "l1"]:
#    print('=' * 80)
#    print("%s penalty" % penalty.upper())
#    # Train Liblinear model
#    results.append(benchmark(LinearSVC(penalty=penalty, dual=False,
#    tol=1e-3)))

# Train SGD model
# results.append(benchmark(SGDClassifier(alpha=.0001, n_iter=50,
#    penalty=penalty)))

# Train SGD with Elastic Net penalty

```

```

#print('=' * 80)
#print("Elastic-Net penalty")
#results.append(benchmark(SGDClassifier(alpha=.0001, n_iter=50,
#                                  penalty="elasticnet")))

# Train NearestCentroid without threshold
print('=' * 80)
print("NearestCentroid (aka Rocchio classifier)")
results.append(benchmark(NearestCentroid()))

# Train sparse Naive Bayes classifiers
print('=' * 80)
print("Naive Bayes")
results.append(benchmark(MultinomialNB(alpha=.01)))
results.append(benchmark(BernoulliNB(alpha=.01)))

print('=' * 80)
#print("LinearSVC with L1-based feature selection")
# The smaller C, the stronger the regularization.
# The more regularization, the more sparsity.
#results.append(benchmark(Pipeline([
# ('feature_selection', SelectFromModel(LinearSVC(penalty="l1", dual=False,
#                                               tol=1e-3))),
# ('classification', LinearSVC(penalty="l2"))])))

# make some plots

indices = np.arange(len(results))

results = [[x[i] for x in results] for i in range(4)]

clf_names, score, training_time, test_time = results
training_time = np.array(training_time) / np.max(training_time)
test_time = np.array(test_time) / np.max(test_time)

plt.figure(figsize=(12, 8))
plt.title("Score")
plt.barh(indices, score, .2, label="score", color='navy')
plt.barh(indices + .3, training_time, .2, label="training time",
         color='c')
plt.barh(indices + .6, test_time, .2, label="test time", color='darkorange')
plt.yticks(())
plt.legend(loc='best')
plt.subplots_adjust(left=.25)
plt.subplots_adjust(top=.95)
plt.subplots_adjust(bottom=.05)

for i, c in zip(indices, clf_names):
    plt.text(-.3, i, c)

plt.show()

```