Efficient Zero-day Attacks Detection Techniques

A thesis submitted
for the award of the degree of
DOCTOR OF PHILOSOPHY

by

Ratinder Kaur
(900903003)

under the guidance of

Dr. Maninder Singh
Associate Professor
Computer Science and Engineering Department
Thapar University, Patiala -147004

Computer Science and Engineering Department
Thapar University, Patiala – 147004, INDIA

February, 2016

Dedicated to:

The Almighty God

For the good health and wellbeing.

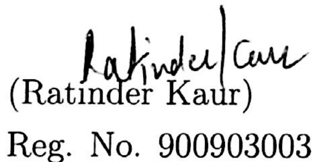
My parents and my brother

Who always inspire, motivate and encourage me to do better and better!

Certificate

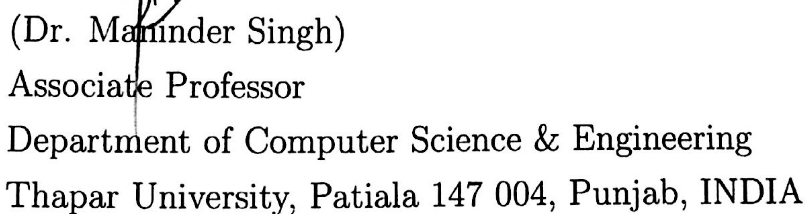
I hereby certify that the work which is being presented in this thesis entitled *Efficient Zero-day Attacks Detection Techniques*, for the award of degree of Doctor of Philosophy submitted to the Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Maninder Singh and refers other researchers works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.



(Ratinder Kaur)
Reg. No. 900903003

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.



(Dr. Maninder Singh)
Associate Professor
Department of Computer Science & Engineering
Thapar University, Patiala 147 004, Punjab, INDIA

Scanned by CamScanner

Acknowledgements

I would like to gratefully acknowledge various people who have been journeyed with me in recent years as I have worked on this Ph.D. thesis. Foremost, I would like to express my sincere gratitude to my supervisor Dr. Maninder Singh, Associate Professor, Computer Science and Engineering Department for his continuous support, patience, motivation, enthusiasm, and immense knowledge. He is a great teacher, best advisor, a strong mentor, a supporting guide and above all a very nice and kind human being. I would like to express my special appreciation and thanks to him for encouraging my research and for allowing me to grow as a researcher. Your advice for my career and other enlightening talks while working together have been priceless.

I also take the opportunity to thank Dr. Deepak Garg, Associate Professor and Head, Computer Science and Engineering Department, Thapar University, Patiala, for providing me with all the necessary facilities for the research.

I would also like to express my sincere thanks to Doctoral Committee Members Dr. Seema Bawa and Dr. A.K. Verma for their insightful comments and hard questions which incented me to widen my research from various perspectives.

I would like to thank faculty and staff members of Computer Science and Engineering Department of Thapar University, Patiala for their help and support.

I want to thank Tata Consultancy Services (TCS), India for their financial support granted through TCS Research Fellowship Scheme.

In particular, I am grateful to Dr. Inderpreet Chopra for all the stimulating discussions, brilliant comments and suggestions. I am extremely thankful and indebted to him for sharing expertise and sincere encouragement extended to me. My heartfelt thanks to Jyotsna Sharma for her guidance and for always being there as an elder sister. I would like to express special thanks to Dr. Shashi Bhanwar for enlightening me the first glance of research. My special regards to Dr. Amit Bhardwaj and Dr. Sanmeet Bhatia who were always ready to give their timely help whenever required.

A special thanks to all my friends for their understanding and encouragement. Nidhi Jain Kansal and Seemu Sharma, thank you for your much needed moral and emotional support and for all the fun we have had in these years. A special mention of thanks to Gurpal Singh, Karamjeet Kaur, Avleen Malhi, Tarunpreet Kaur. I cannot list all names here, but friends, you are always on my mind.

I am also grateful to numerous others who have directly or indirectly contributed towards carrying out the research in all aspects.

My deepest gratitude for my parents for their unconditional love, support, dedication and the many years of support during my studies that

provided the foundation for this work. Thank you for trusting me and giving me all facilities and freedom to work on my research. Thanks brother, your strong comments always motivates me to remain hungry to chase my dreams. Thanks to my another half of family, members at Nanighar, your prayers for me was what sustained me thus far.

Above all, I am thankful to Almighty God, for all his blessings that came to me abundantly, unexpectedly and delightfully.

Ratinder Kaur

October 2015

Abstract

Root cause of any security loophole on all kinds of networks lies within a developer's realm. As security is not seriously considered in software development effort, more and more vulnerabilities are getting discovered every single day. Software vulnerabilities are of two types: known and unknown. Known vulnerabilities are the one which has been identified and fixed. On the other hand, unknown vulnerabilities are the one for which there is no prior knowledge of the flaw and therefore, no patch or fix is available for it. These are also known as zero-day vulnerabilities, which are extremely dangerous and unpredictable. Zero-day vulnerabilities provide a backdoor into any operating system or application and represents a serious threat. A cyber attack that targets or exploits zero-day vulnerability(ies) is known as zero-day attack.

The major contribution of the thesis is a system called RADAR. RADAR stands for Real-time Zero-day Attack Detection Analysis and Reporting system. RADAR uses a hybrid approach and is capable of detecting zero-day attacks. It does so by identifying benign traffic based on important traffic features and creating a baseline to seek unknown deviations. RADAR also implements a stub to analyze zero-day binary in parallel. The analysis and reporting stub integrates

existing malware analysis functionalities and utilities in a component based architecture.

RADAR demonstrates following main features: (a) Bridging the gap between the detection and analysis phase to deliver the first inclusive behavioral report about a zero-day attack. (b) Combines features of existing zero-day attack detection techniques (anomaly detection, signature-based detection and behavior-based detection) and finally offers better sensitivity and specificity. (c) It is based on a layered architecture where each layer is dedicated to a single functionality and works in parallel to improve system performance. Detection layer uses machine learning to detect zero-day attacks. Analysis layer combines static and dynamic malware analysis functionality to analyze the captured binary. Step by step manual analysis can also be done to help malware analyst in case of manual intervention is required. Resource layer supports the working of above two layers. (d) Implements a kernel based monitoring to track system objects during dynamic analysis in a reliable way. (e) Generates comprehensive report on zero-day malware behavior in HTML and PDF format.

RADAR is implemented and evaluated against various standard IDS evaluation metrics. The results shows high sensitivity and specificity. Also in this research work, reports generated by RADAR are compared to the information provided by online virus and behavioral scan engines. Results are published to research community in form of peer-reviewed journal and conference publications.

Contents

List of Figures xiii

List of Tables xiv

1 Introduction 1

1.1 Status of Network Security 2

1.2 Zero-Day Attacks 7

1.2.1 Vulnerability Life Cycle 8

1.2.2 Developing a Zero-day exploit 9

1.2.2.1 Analyze 9

1.2.2.2 Fuzz 9

1.2.2.3 Develop 11

1.2.2.4 Exploit 11

1.3 Research Motivation 12

1.4 Thesis outline 14

2 Literature Survey 17

2.1 Zero-day Attack Detection Techniques 19

2.1.1 Statistical-based 20

2.1.2 Signature-based 26
2.1.2.1 Content-based Signatures 28
2.1.2.2 Flexible Content-based Signatures 32
2.1.2.3 Semantics-based signatures 34
2.1.2.4 Vulnerability-driven signatures 35
2.1.3 Behavior-based 36
2.1.4 Hybrid Techniques 38
2.1.5 Comparative Analysis 40
2.2 Malware Analysis 46
2.2.1 Static Analysis 49
2.2.2 Dynamic Analysis 51
2.3 Honeynet: A Detection and Analysis Tool 59
2.4 Problem Formulation 61
2.5 Thesis Objectives 63
2.6 Summary 63

3 RADAR- Runtime Zero-day Attack Detection Analysis and Reporting 65

3.1 Introduction 65
3.2 Evolution of RADAR 66
3.3 RADAR Architecture 70
3.3.1 Design Principles for RADAR 70
3.4 Detection Layer 72
3.4.1 Misuse Detector 72
3.4.2 Tagger 74

3.4.3 Preprocessor 74
3.4.4 Detection Engine 77
3.4.5 Updater 81
3.5 Analysis Layer 82
3.5.1 Extractor 82
3.5.2 Analysis and Reporting Stub 82
3.5.3 Static Analysis Engine (SAE) 84
3.5.4 Dynamic Analysis Engine (DAE) 91
3.5.5 Manual Analysis 103
3.5.6 Reporting 104
3.6 Summary 110

4 Experimental Details and Results 111

4.1 Introduction 111
4.2 Experimental Setup 112
4.3 Experimental Results 114
4.3.1 Detection Results and Discussions 114
4.3.2 Analysis Results and Discussions 119
4.4 Comparison with Existing Techniques 129
4.5 Summary 131

5 Conclusions and Future Work 132

5.1 Conclusions 132
5.1.1 Key Findings 135
5.2 Future Work 137

References 162

List of Figures

1.1 Network Attacks Statistics 2014 [Pas15] 2
1.2 Attack Sophistication vs Intruder Technical Knowledge [Lip13] . 3
1.3 Zero-Day Attacks are Biggest Risk [Sta15] 5
1.4 Vulnerability Window [Pas12] 8

2.1 Classification of Zero-day Attack Detection Techniques 20
2.2 Zero-day Attack Detection Techniques Surveyed 21
2.3 Types of Malware Analysis 48

3.1 Basic System Components 67
3.2 Suspicious Traffic Filter (STF) Module 68
3.3 Evaluating Unknown Traffic Trace 69
3.4 Hotfix Update 69
3.5 Layered Architecture of RADAR 71
3.6 RADAR: Process Flow Diagram 73
3.7 Creation of Good Profile 78
3.8 Hypersphere Model 79
3.9 Analysis & Reporting Stub 83
3.10 SAE- Antivirus Scanner 85

3.11 Kernel-based Analysis Component 93
3.12 Minifilter Callback Mechanism 97
3.13 Reporting Attributes 105
3.14 RADAR Reporting Framework 106

4.1 RADAR Experimenal Setup 112
4.2 Working JAVA Reverse Bind Shell 119
4.3 VirusTotal Result for JAVA file 120
4.4 VirusTotal Result for EXE file 121
4.5 General Details 122
4.6 Sections 122
4.7 Imports 123
4.8 Exports 123
4.9 Resources 124
4.10 Extracted Strings 124
4.11 Dynamic Analysis Summary 125
4.12 Dynamic Activity w.r.t Time 126
4.13 API Calls 126
4.14 Registry Activity 127
4.15 File Activity 127
4.16 Handles 128
4.17 Run time Modules 128

List of Tables

1.1 Transition from Early Days 4

2.1 Comparative Analysis: Pros_Cons 40

2.2 Difference between Emulator and Virtual Machine 56

3.1 Extracted Features 76

3.2 File Manipulation IRP Function Codes 98

3.3 Registry Operation Notifications 99

3.4 Process and Thread Manager Routines 100

3.5 Run-time Filtering Layer Identifiers 101

4.1 Software Packages Used 113

4.2 Detection Results for Synthetic Dataset 116

4.3 Distribution of Benign Samples 117

4.4 Distribution of Malware Samples 117

4.5 Detection Accuracy For Zero-day Non-Obfuscated Malware 118

4.6 Detection Accuracy For Zero-day Obfuscated Malware 118

4.7 Comparison: Honeynet vs RADAR 129

4.8 Comparison: Analysis Tools vs RADAR Analysis 131

Chapter 1

Introduction

Internet has become a vital part of everyday life, owing to the revolutionary changes it has brought to various fields. Dependence on the Internet as an information highway and knowledge bank is exponentially increasing so that a going back is beyond imagination. Due to transfer of critical information through the Internet and with the tremendous growth in e-commerce, the Internet has become more prone to different types of network attacks and thus created a vital need for network security. The news of successful Internet attacks has become commonplace. Hackers broke into commercial sites to steal credit card information and into defense industry sites to seek top-secret military documents. Firewalls, intrusion detection (IDS) and prevention (IPS) systems across various enterprise networks log hundreds of hacker attempts everyday. Figure 1.1 shows the top 10 attack techniques (left) and top 10 targets (right) of 2014. This rising level of cyber crime is an indication of an enormous threat to national security and economics.

This chapter discusses the current status of network security and explains how

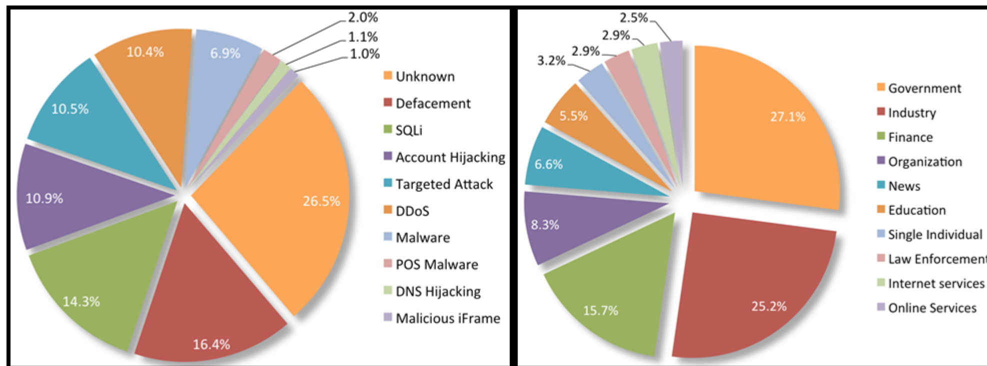


Figure 1.1: Network Attacks Statistics 2014 [Pas15]

the cyber attack space has changed over the years. This chapter provides a high level overview of zero-day attacks and presents the significant research challenges in the area of zero-day attack detection techniques. It then briefly discusses the need for the development of an efficient zero-day attack detection technique. It ends with a discussion on organization of the rest of the thesis.

1.1 Status of Network Security

In this era, cybercrime is regarded as a profession. A cyber criminal is no longer the nerd who loves to stay indoors but is now an organized gangster closely associated with drug trafficking, extortion and money laundering. Most amateur hackers and cyber criminals are teenagers. There are also organized hackers, professional hackers and discontented employees. The reason for organized hackers to do hacking is to fulfill their political bias, fundamentalism, etc. In case of professional hackers, their work is motivated by money. The group of discontented employees includes those people who have been either fired by their employer or are dissatisfied with their employer.

Likewise, hacking a computer is no longer difficult because most of the attacking tools are easily available on the Internet and require little or no skill. Sophisticated attackers usually build attack scripts and attack toolkits that the novice attacker can use with the click of mouse, with devastating effects. Figure 1.2 shows that intruder tools are becoming increasingly sophisticated and easy to use by novice intruders.

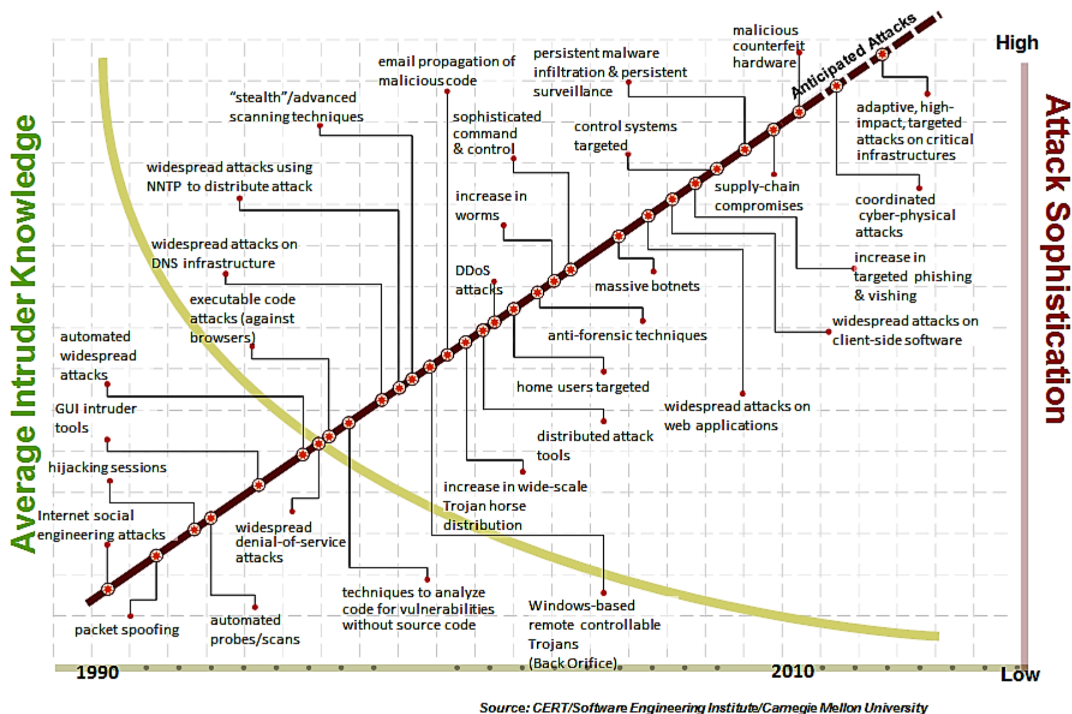


Figure 1.2: Attack Sophistication vs Intruder Technical Knowledge [Lip13]

Earlier hackers were looking for the easy breaches. They mainly used a broad "spray and pray" approach to opportunistically find targets. Also "signature-based" security solutions were mainly used to identify known, malicious code patterns and block them. But today, as organizations have strengthened their security, hackers also evolved. Now attacks are more sophisticated and targeted

than ever before. Rather than sending generic malware, hackers today carefully plan each and every attack, using unique, unknown (zero-day) exploits that render signature-based network security defenses nearly useless. Table 1.1 summaries this transition from the early days to today in network security.

Table 1.1: Transition from Early Days

	Early Days	**Today**
Attackers	Amateur, curious, shows off coding skills	Professional, well organised with a specific objective
Threat Vector	Generic malware, no obfuscation	Unknown/Zero-day malware, Advanced Persistent Threats (APTs)
Types of Attack	Opportunistic attacks	Targeted, well planned, stay unnoticed for longer periods
Spread Via	File transfer (virus), network (worm) or removable media	Drive-by-Download, Web or Email attacks, Social Engineering
Technology	Signature or Rule based	Behavior, Analytics based

All this makes the Internet a pervasive threat vector for various types of organizations. As new technologies are developed and adopted to meet changing business requirements, sneaky attackers wait to exploit vulnerabilities exposed.

Software or hardware vulnerabilities are of two types: known and unknown. Known vulnerabilities are the one which has been identified and fixed. On the other hand, unknown vulnerabilities also known as zero-day vulnerabilities are the one for which there is no prior knowledge of the flaw, and, therefore, no patch or fix is available for it. They are extremely dangerous and unpredictable. A zero-day vulnerability provides a back-door into any operating system or application and represents a serious threat. And the attacks that targets or exploits

zero-day vulnerabilities are known as zero-day attacks.

Zero day attacks are reality and their number reported each year increases immensely. In recent years, zero-day attacks have been dominating the headlines for political and monetary gains. They are a potent weapon in the hands of attackers and are being used as essential success vectors in various sophisticated and targeted attacks. These secret weapons give attackers a crucial advantage over their targets to break into traditional security products that identify only known, confirmed threats. The zero-day attacks are among the top security concerns that the modern enterprises face today. A recent survey from Palo Alto Networks reported that zero-day attacks and evasive malware represent the biggest risks (Figure 1.3). People talked about zero-day attacks few years back, but today every industry faces it. Another day, another breach and a company losses sensitive data.

Figure 1.3: Zero-Day Attacks are Biggest Risk [Sta15]

Reports on the Internet show an alarming increase of such attacks against both corporate and home user systems. According to Symantec's Internet Security

Threat Report of 2014 [Sym14] there is 91% increase in targeted attacks campaigns in 2013, 62% increase in the number of security breaches and 23 zero-day vulnerabilities were discovered. Another security threat report by Sophos [Sop14] reported that large tech companies like Apple, Facebook, Microsoft, Twitter and others were targeted with same zero-day Java vulnerability that attacks multiple customers. FireEye, Inc. an IT security company [Inc14] also discovered 11 zero-day attacks targeting softwares like Internet Explorer, Java, Adobe Flash, PDF and ActiveX in 2013. All such facts and figures represent a serious concern in today's network security.

The most dangerous zero-day exploits ever seen in cyberspace are Hydraq Trojan [Lel10], Stuxnet [FMC11], RSA Attack [Mar11], Duqu [Sym11] and Flame [NMT12] [GW12]. The Hydraq Trojan, also known as Aurora attack, a 2009 campaign aimed to steal intellectual property, compromise user accounts and spy on 20 targets including Google, Adobe Systems, Juniper, Yahoo, Symantec. The Stuxnet worm, known as malware of the century and was discovered in June 2010. It infects Programmable Logic Controller (PLC) in SCADA systems, contains 3 zero-day exploits, 3 rootkits, has military grade encryption and used unknown injection and spreading malware technique. In March 2011 attackers breached EMC's RSA Security division and stole secrets related to its widely used SecurID token authentication system, leaving millions businesses less secure. Duqu, discovered in September 2011 is related to Stuxnet worm. Part of this malware is written in unknown high level programming language. It exploits zero-day Windows kernel vulnerabilities, uses stolen digital keys and is highly targeted. Flame is a modular computer malware discovered in 2012 that exploits some same zero-day vulnerabilities in Microsoft Windows as Stuxnet. It is one

of the most sophisticated and complex malware ever encountered.

In January 2013, Facebook was hacked by a sophisticated zero-day Java exploit which used a zero-day to bypass the Java sandbox to install the malware [Par13]. The Sony Hack in December 2014 [Kul14], released 100 terabytes of confidential data belonging to Sony Pictures Entertainment. The data included huge trove of emails from senior executives, personal information of their employees and secrets about upcoming projects. It was reported that the attackers took advantage of a zero-day vulnerability and the intrusion had been occurring for more than a year, prior to its discovery.

1.2 Zero-Day Attacks

A zero-day attack is one that exploits zero-day vulnerability and has no signature. It can be in any form, whether a worm, virus, trojan or anything else. It takes advantage of a bug or problem before a patch has been created and occurs before the first day the vulnerability is known - hence the name zero day. They occur during the vulnerability window that exists at the time between when a vulnerability is first exploited and when software developers start to develop a counter to that threat. It is difficult to measure the length of the vulnerability window (Figure 1.4), as attackers do not announce when the vulnerability was first discovered. Even developers may not want to distribute data for commercial or security reasons or they may not know if the vulnerability is being exploited when they fix it. So the vulnerability may not be recorded as a zero-day attack. The vulnerability window however, can be of several years long. According to an empirical study [BD12], a typical zero-day attack may last for 312 days on

average and, after vulnerabilities are disclosed publicly, the volume of attacks exploiting them increases up to 5 orders of magnitude.

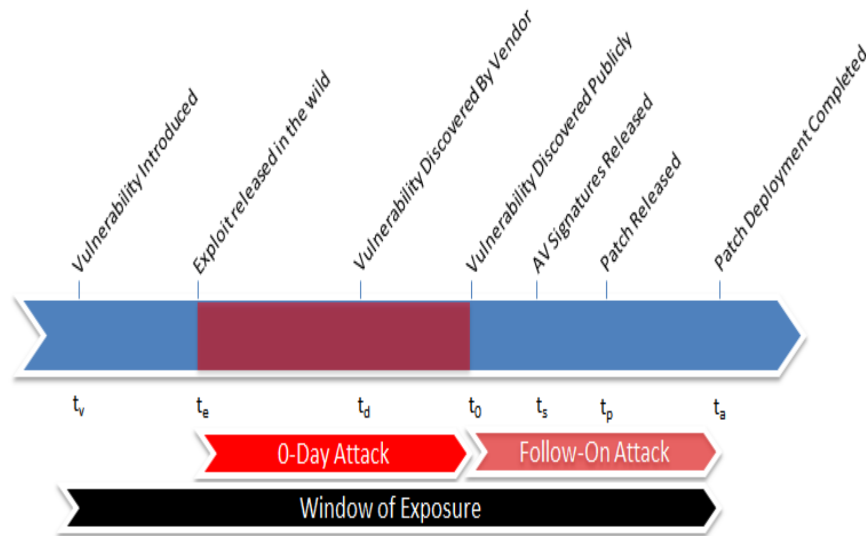


Figure 1.4: Vulnerability Window [Pas12]

1.2.1 Vulnerability Life Cycle

Amontip et. al. [JTA08] classifies vulnerability life-cycle into five categories and addresses various factors, such as availability of patches and exploit code that contribute to the probability of Zero-day attack.

i. **Zero-day attack (ZDA):** The vulnerability is discovered by a black-hat and is not publicized. The black-hat works quietly on an exploit code.

ii. **Pseudo zero-day attack (PZDA):** This is similar to ZDA. However, it results from leniency on the part of system administrators not applying a particular patch even though the patch was released by vendor some time ago.

iii. **Potential for pseudo zero-day attack (PPZDA):** This is similar to PZDA. However, vulnerability has not been attacked, yet has a high possibility of being exploited despite the availability of a patch.

iv. **Potential for attack (POA):** Vulnerabilities and their details are revealed and automated exploit code or programs are known. However, the vendors are not yet able to produce patches for wide distribution and thus, the vulnerability of this type becomes ZDA after the outbreak of attack.

v. **Passive:** In this, exploit codes have not yet been produced or available.

1.2.2 Developing a Zero-day exploit

To develop a zero-day exploit, a new vulnerability is first discovered and then exploited. The unknown vulnerability finding and exploiting is a process that consists of four phases:

1.2.2.1 Analyze

This phase focuses on Attack Surface Analysis [MW11]. To find and analyze a zero-day weakness, it requires substantial knowledge of protocols and systems from the attacker. For relatively inexperienced attackers such expertise can be built into the tools, like fuzzers.

1.2.2.2 Fuzz

This phase is optional and focuses on fuzz testing the identified attack vectors. Fuzzing uses invalid, unexpected, or random inputs to stress-test the computer program. The program is monitored for exceptions such as crashes, or failing

built-in code assertions or for finding potential memory leaks. The main aim of this test is to simulate various potential attack scenarios. Black-hat hackers mainly fuzz exploitable security bugs in unused or rarely-used software functionality. Unlike traditional security testing methodologies, fuzz tests are highly efficient in finding unknown vulnerabilities, because they are not based on earlier vulnerabilities, but on protocol models. Apart from fuzzing, other methods that are used to find zero-day vulnerabilities are:-

- *Source code analysis*: It is testing of source code for finding bugs. It can either be static or dynamic. In static, the source code is examined without actually executing the program. Whereas in dynamic real-time program testing is done to uncover subtle vulnerabilities.
- *Binary code analysis*: Binary analysis helps in finding vulnerabilities and defects in binary code without having access to the source code.
  - Static analysis: For static analysis, disassemblers like objdump (part of the GNU Binutils) and IDA-Pro [Eag11] are used to locate functions, recognize jumps, identify variables, etc. to understand the program behavior without running it.
  - Dynamic analysis: In dynamic analysis, the program is executed and monitored to find bugs and to observe the interaction of a program with Operating System through system calls.
- *Hybrid methods*: Combines various methods mentioned above.

1.2.2.3 Develop

After an unknown vulnerability is found, the next step is to develop a zero-day exploit. The critical part of an exploit is its shellcode. There are different methods to place shellcode in exploit depending on the size of available memory space on the stack. If the stack memory is big enough to fit the entire shellcode, it can either be located by a hardcoded stack address or could be referenced by using a register. Otherwise one or more trampolines can be used to jump to the shellcode. And, if the available size is too small to squeeze the entire shellcode, a technique known as egg hunting is used. Egg hunting technique or "staged shellcode", basically allows to use a small amount of custom shellcode to find actual (bigger) shellcode (the "egg") by searching for the final shellcode in memory. Otherwise stated, first a small amount of code is executed, which then tries to find the real shellcode and executes it.

1.2.2.4 Exploit

Attackers can use several different attack vectors to exploit zero-day vulnerabilities. Typical attack vectors include web browsing (users go to a website that disseminates malware), email or social media messaging (messages come with social engineering tactics to entice the user to open the attachment) and Universal Serial Bus (USB) (a traditional method to spread malware). These attack vectors can be categorized as:-

1. *Network Attack Vector*: It utilizes malicious network traffic to remotely compromise their target systems. Network vectors penetrate the target system, launch the attack's malicious payload and propagate itself without

human intervention. They predominantly exploit vulnerabilities in protocols and network-aware processes. These vulnerabilities are typically the result of programming errors that often provide opportunities for a buffer overflow.

2. *Application Attack Vector*: It utilizes executable files to attack and compromise target systems. Unlike network vectors, executable files typically require some form of user involvement to launch an attack. For example, sending malicious e-mail attachment to exploit unknown vulnerabilities in application opening the attachment.

1.3 Research Motivation

The results of recent research outputs have been the prime motivation for this research. Zero-day attack detection techniques are evolving and facing many open issues. Till date these issues have not been addressed correctly. The existing zero-day detection techniques do not "raise the bar" for the attackers, while their cost for the defender in terms of resources that need to be devoted to detection can be significant. Several research projects have addressed the problem of zero-day detection but unfortunately they exhibit one or more problems. As per literature survey there exist certain research proposals that have been promising, but they can also be easily defeated by using minor enhancements to the attack vectors. As attack tools are improving, reliance on minor improvements in the detection processes is insufficient. The current techniques have several drawbacks and requires a deep analysis of each mentioned aspect. Following are the most significant challenges identified in this area:

1. **Unknown Signatures**: The ultimate challenge in zero-day attack detection is that we have to deal with unknown attacks that do not have any signature available. These attacks can evade purely signature-based detection systems until a new signature is released to contain them. Therefore, the most deployed signature based detection systems are ineffective to detect unknown attacks or variation of known attacks [MPB+13].

2. **Evade Defenses Through Obfuscation**: Zero-day attacks together with evading techniques is a serious concern. Attackers use automated tool kits to generate several thousand malware variants at once with armoring techniques like run-time obfuscation, polymorphism and packers. It is estimated by security experts that more than 70,000 new instances of malware are released each day [Ras12]. With such obfuscation techniques malware authors are able to easily fool the detection engines. Thus, its needed to develop more robust and efficient solutions to deal with the exponential growth of zero-day malware arising from innumerable automated obfuscations. In an empirical study it was found that zero-day attacks are long lived and they may evade detection for more than a year [BD12].

3. **Low Detection Rate and High False Positives**: Another challenge is that zero-day attack detection systems still have a low detection rate. This is due to the novelty of zero-day attacks, sophisticated evasion techniques, inappropriate threshold settings and real-world data imperfections. The detection rate is decreasing as compared to identifying benign files as malware (false positive) which is becoming very high with high failure rate to detect obfuscated malware (false negative) [AVWA11].

4. **Manual Analysis**: Till date there is really no fully automated way to understand completely the malicious intents of a zero-day attack. For deep inspection of malicious code either static analysis or dynamic analysis is done and both require expert manual intervention. There is no single best approach for behavior analysis of new attacks, therefore, researchers combined the use of both static analysis and dynamic analysis [BCT11] [GCT13] [TSPM11]. But still its not a fully automated solution.

5. **Validation Gaps**: Various proposed zero-day attack detection techniques test their systems in a simulated environment with self-crafted known attacks that serve as zero-day attacks for those systems. Such evaluation leaves several gaps between simulated and real attack scenarios. Therefore, before the online deployment of the new systems they should be deployed and tested in a commercial environment with extensive testing on real network traffic.

1.4 Thesis outline

This section discusses the framework of this thesis. Thesis is organized as follows:

Chapter 1 *Introduction*

This chapter discusses current status of network security in terms of detecting novel attacks. It provides a high level overview of zero-day attacks, vulnerability life cycle and zero-day exploit development. Chapter clearly presents the motivation for the development of an efficient zero-day attack detection technique. Chapter ends with a discussion on the organization of the rest of the thesis.

Chapter 2 *Literature Review*

Chapter 2 provides a detailed survey to outline the research efforts concerning the detection of zero-day attacks. It classifies the existing techniques into statistical-based, signature-based, behavior-based and hybrid techniques. Comparison analysis of the zero-day attack detection techniques is presented. Chapter also discusses various malware analysis tools and techniques used to capture the behavior of a zero-day malware. The chapter finally concludes with the problem formulation and objectives.

Chapter 3 *RADAR Real-time zero-day Attack Detection Analysis & Reporting*

Chapter 3 presents the evolution, design and implementation of RADAR system to automatically detect and analyze zero-day attacks. RADAR is the first system to combine all the three zero-day attack detection techniques, namely statistical-based, signature-based and behavior-based. It also implements an analysis and reporting stub to analyze the captured zero-day binary. This stub integrates existing static malware analysis and dynamic malware analysis functionality to work as a single unit in a component based architecture where any of the functionality can be replaced in the future. It also provides manual analysis to do step by step analysis of binary if needed. RADAR reports zero-day malware behavior in HTML and PDF format.

Chapter 4 *Deployment, Testing and Validation of Proposed Technique*

Chapter 4 describes the test environment, deployment and results. Various tools used to implement RADAR system are mentioned. The system was tested using two types of dataset: synthetic dataset and real-time dataset. To generate

zero-day attacks, various obfuscation engines and techniques were used to mutate known shellcodes. These mutated known shellcodes act as zero-day attacks for our system. The real-time dataset comprises of malware collected from various online malware repositories. Validation of the proposed technique is done using various standard metrics for intrusion detection. Analysis reports were evaluated by matching analysis information provided by well-known online virus and behavioral scan engines. Detailed discussion on results is also done.

Chapter 5 *Conclusions and Future Scope*

Chapter 5 presents the conclusion of the thesis, describes the main contribution of the research work and highlights future research direction based on the results obtained.

Chapter 2

Literature Survey

Preventing zero-day attacks is the major concern for network administrators because they pass undetected through conventional defenses. A typical enterprise uses firewalls, intrusion-detection systems and antivirus software to secure its IT infrastructure. From these IDS is a core component in the network to protect against known attacks [SC13]. Firewall, IDS and antivirus offers good first-level protection, but despite their best efforts, they are unable to protect enterprise against zero-day attacks and are ineffective in keeping up with rapidly increasing volumes of malware variants. Today, attackers invest lots of time in designing attacks to avoid detection and to maximize their impact. They accurately identify their targets and approach them in the best way. Now hackers are incredibly resourceful, technically skilled, and handsomely rewarded for their efforts. They are constantly refining their techniques, and their attacks are swifter and stealthier than even before. A conventional reactive security solution, based on packet filters and signatures, is powerless against the new generation of sophisticated zero day attacks. As a result, enterprises require security systems to protect

them against both known and unknown attacks.

Honestly, there is actually no method of detection for zero-day attacks that is 100% reliable. However, the research community is devising new solutions and evolving the existing systems to stay one step ahead in the "cat and mouse game" between the hackers and defenders. Detecting zero-day attacks is like finding the needle in the haystack. It involves detecting and blocking zero-day incidents among millions of network traffic flows accurately, with few false positives or false negatives. To combat zero-day attacks smarter security solutions employing integrated layers of protection must be developed with key parameters to consider like efficiency, accuracy, scale, and timeliness of detection. It must continuously detect, prevent, analyze, and respond to zero-day attacks in a proactive manner, achieving zero-day protection while balancing enterprise needs for security and access.

This chapter is organised as follows: Section 2.1 provides a detailed literature survey to outline the research efforts in relation to detection of zero-day attacks. The detailed survey is published in [KS14b] and this chapter is part of that paper with more additions. The zero-day attack detection techniques are categorized as statistical-based in section 2.1.1, signature-based in section 2.1.2, behavior-based in section 2.1.3 and other hybrid techniques in section 2.1.4. Chapter also introduces malware analysis techniques used to capture the behavior of a zero-day malware in section 2.2. Static malware analysis is introduced in 2.2.1. Dynamic malware analysis techniques and tools are discussed in section 2.2.2. Honeynet is also presented as a detection and analysis setup in section 2.3 Based upon the above discussions section 2.6 summarizes the chapter after formulating problem statement in section 2.4 and stating clear objectives of the research in

section 2.5.

2.1 Zero-day Attack Detection Techniques

To defend against zero-day attacks, the research community has proposed various techniques. The zero-day attack detection techniques are either network-based or host-based. Network-based systems detect zero-days at the network level as the attack data travel across the network in the form of packets. Host-based systems detect zero-days at the system level once the attack reaches the vulnerable application and was processed. The host-based detection installs software on the machine to be monitored and because it runs on the machine itself, the level of analysis compared to network-based is much deeper. Unlike host-based detection that monitors a single system, a single network-based detection system can monitor multiple systems. The network-based systems are able to detect and contain the attacks in their early stage because initially the number of machines compromised is limited. Hence, it is unlikely that a host will see the early attack packets and be able to respond in the early critical period of attack [LSC+06]. Many host-based systems have been designed to detect Zero-day attacks like TaintCheck [NS05], DACODA [CSW05], COVERS [LS05], Packet Vaccine [W+06], Vulnerability signature [BNS+06] and Vigilante [C+05], etc. These host-based approaches work on the users' end hosts or honeypots and face problems like scalability and wide coverage. Besides this, host-based approaches affect the performance of the protected host and generate non-compatible signatures for network filtering. Therefore, exclusively using a host-based technique is not sufficient to defend against zero-day attacks. However, it can become a critical

component in the complete detection framework.

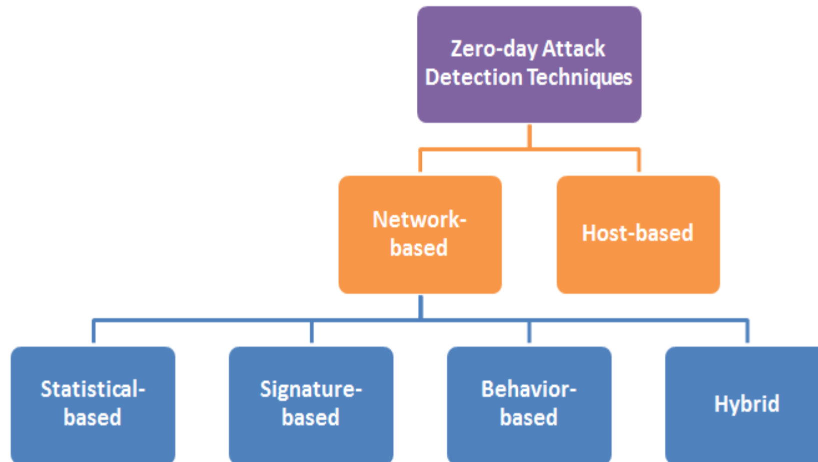


Figure 2.1: Classification of Zero-day Attack Detection Techniques

Due to their simplicity and the ability to operate online in real time, most widely deployed intrusion detection systems are network-based only. Therefore, literature primarily focuses and discusses network-based systems. The network-based zero-day attack detection techniques can be broadly classified into: statistical-based, signature-based, behavior-based and hybrid techniques as shown in the Figure 2.1. All the techniques that are surveyed in thesis are presented in Figure 2.2.

2.1.1 Statistical-based

The statistical-based techniques are independent of signatures and thus, take a different approach to detect zero-day attacks. The concept of statistical detection is to determine "normal" network activity and to flag out anomalous (not normal) that falls outside its scope. To identify an anomaly, the system uses previous

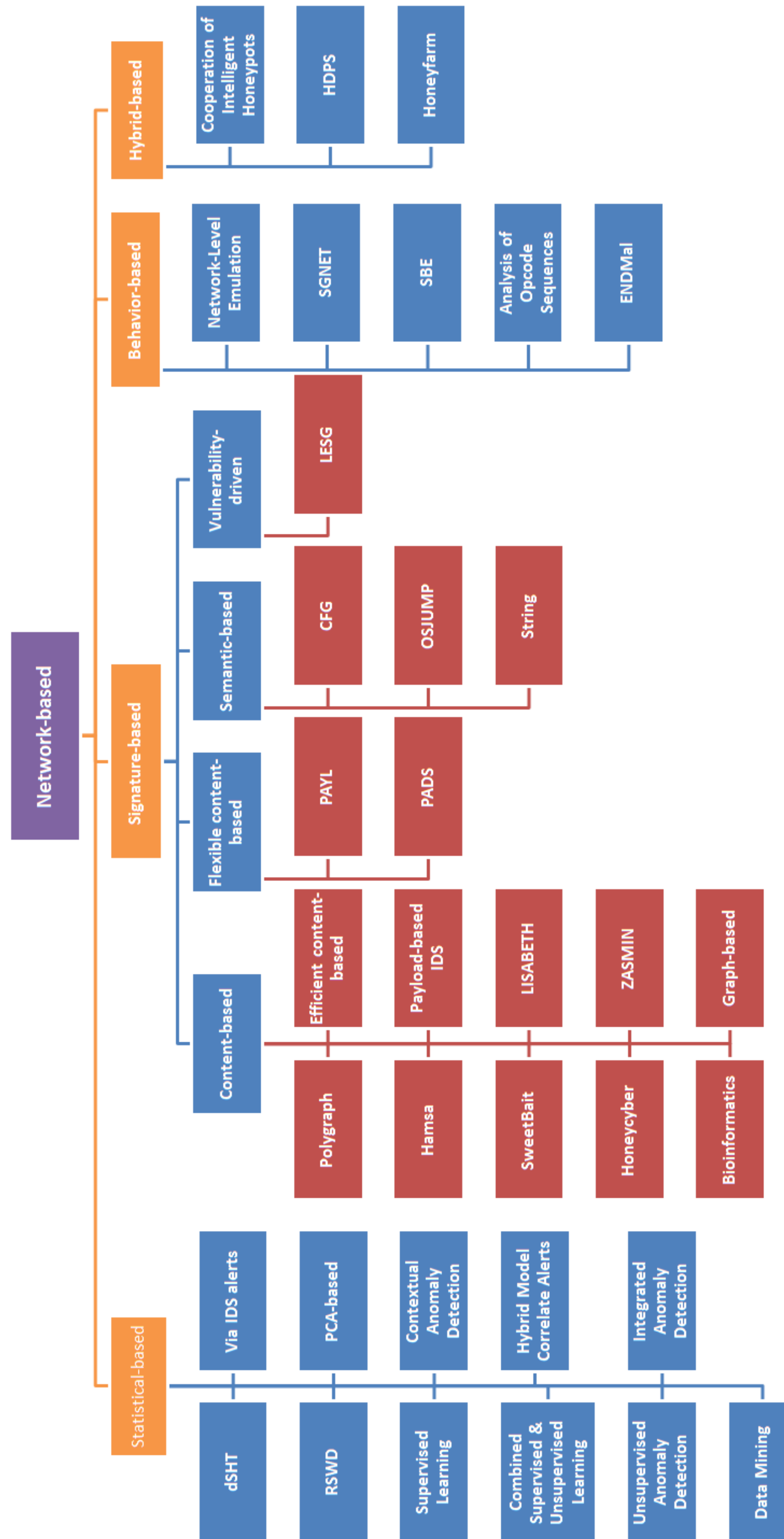


Figure 2.2: Zero-day Attack Detection Techniques Surveyed

network activities processed with complex statistical methods and algorithms. Most of these techniques are dependent on attack profiles build from historical data. Due to the static nature of attack profiles, the detection techniques are unable to adapt to the timely changes in the environment. For any change in the data pattern the system will require an updated profile with constant training. Setting the limit (or detection parameters) for judging new observations, is a critical step in designing a detection approach since it has a dramatic effect on the quality of the detection. If the threshold value is very narrow, it will frequently be exceeded resulting in a high rate of false positive alarms, and if it is very wide the limit will never be exceeded, resulting in many false negative alarms. In a statistical-based detection, the detection parameters are either manually extracted or adjusted to detect new attacks. All these factors limit the statistical detection approaches to work in off-line mode.

A distributed and decentralized detection technique is designed in [Che07]. It uses a statistical tool called distributed Sequential Hypothesis Testing (dSHT) to build a strong distributed worm detector from imperfect anomaly detectors that have high false positive rates. The research work addresses the two most important components of worm defense: detection and response. The techniques developed for the worm defense are independent of worm signatures. This dissertation also presents a generic testing framework based on EMULAB [W+02] and DETER [B+04] network testbed to evaluate worm defense models.

Detection via IDS Alerts is [STK08] a feature extraction method to detect zero-day attacks from IDS alerts. This feature extraction method is based on the basic features of IDS alerts such as source address and port, destination address and port, detection time and signature name. From these six original features some

new statistical features are defined. In order to detect 0-day attacks from IDS alerts with new features, an unsupervised learning technique, One-class Support Vector Machine (SVM) [SPST+01b] along with Sequential Minimal Optimization(SMO) algorithm [Pla99] is applied. In their previous work [SHT+07], authors proposed a feature extraction scheme based on "Incident ID" feature among the original features of IDS alerts. Incident ID feature gr2oups the two alerts having a similar Incident ID as correlated attacks. However, that approach didn't work because not all vendors provide Incident ID feature in their IDS product, and even if provided, its building mechanism is different from others. Hence, the solution cannot be applied to all IDS products.

Rough Set Worm Detection (RSWD) [SC09] is a novel scheme extended from Rough Set Theory (RST) to detect zero-day polymorphic worms. It provides a minimum set of filtering rules for network barrier equipment to block the worm spreading. This scheme is based on an assumption that, the polymorphic worm packets are generated by some specific worm program and attack the same vulnerability. Therefore, some patterns exist even if the polymorphic engine mutates dynamically and frequently. The system is implemented as client-server architecture as illustrated. A client is deployed within a network to collect traffic packets, to describe each packet's observable value in a characteristic vector and to report the vectors to a central server. The server then distills worm packets from large volumes of characteristic vectors and classifies them into suspicious clusters and normal clusters.

A technique is presented for detecting new attacks in low-interaction honeypot traffic [ACMZ09]. It is based on Principal Component Analysis (PCA), a widely used multivariate statistical technique for reducing the dimensionality of variables

and unveiling latent structures and detecting outliers in data sets [Jol02] [Jac03]. The proposed technique projects new observations onto the residuals' space of the least significant components and measures their distances from the k-dimensional hyperspace defined by the PCA model using the square prediction error (SPE) statistic. A higher value of SPE indicates that the new observation represents a new direction that has not been captured by the PCA model of attacks seen in the historical honeypot traffic.

Supervised Learning [AVWA11] is a novel method of employing several data mining techniques to detect and classify zero-day malware based on the frequency of Windows API calls. A machine learning framework is developed using eight different classifiers, namely Nave Bayes (NB) Algorithm, k-Nearest Neighbor (kNN) Algorithm, Sequential Minimal Optimization (SMO) Algorithm with 4 different kernels (SMO-Normalized PolyKernel, SMO-PolyKernel, SMO-Puk, and SMO-Radial Basis Function (RBF)), Backpropagation Neural Networks Algorithm, and J48 decision tree. This system proves to be better than similar signature-free techniques that detect polymorphic malware and unknown malware based on analysis of Windows APIs.

Contextual Anomaly Detection [AK12] is a contextual misuse and anomaly detection prototype to detect zero-day attacks. The contextual misuse detection utilizes similarity with attack context profiles, and the anomaly detection technique identifies new types of attacks using the One Class Nearest Neighbor (1-NN) algorithm. It uses information entropy and linear data transformation to generate feature-based and linear function-based attack profiles [AK14; AK13] and systematically creates contextual relationships between known attacks to generate attack profiles that capture activities of zero-day attacks.

Combined Supervised and Unsupervised Learning [CLS+13] technique is presented for zero-day malware detection. It employs machine learning based framework to detect malware using layer 3 and layer 4 network traffic features. It utilizes supervised classification to detect known malware and unsupervised learning to detect new malware and known variants. A tree-based feature transformation is also introduced to overcome data imperfection issues and to detect the malware classes effectively.

A hybrid model [AJA11] for correlating alerts of known and unknown attack scenarios consists of two parts. In the first part, an attack graph-based method is used to correlate alerts raised for known attacks and to hypothesize or predict missed alerts. In the second part, a similarity-based method is used to correlate alerts raised for unknown attacks which cannot be correlated using the first part and to update the attack graph. These two parts cooperate with each other such that if the first part could not correlate a new alert, the second part is applied. An additional method named alerts-bisimulation is also proposed for compressing graphs of correlated alerts.

Unsupervised Anomaly Detection System [STOK09] is based on clustering and multiple one-class SVM to detect 0-day attacks and to to improve the detection rate while maintaining a low false positive rate. It is able to construct intrusion detection models automatically without using labeled training data. In [STON13] the authors have optimized the values of parameters without predefining. This helps to construct models based on without tuning the parameters, and thus contributes to more practical operations in the real environment.

Integrated Anomaly and Misuse Detection [KLK14] method hierarchically integrates a misuse detection model and an anomaly detection model in a decompo-

sition structure. First, the C4.5 decision tree (DT) is used to create the misuse detection model that is used to decompose the normal training data into smaller subsets. Then, the one-class support vector machine (1-class SVM) is used to create an anomaly detection model in each decomposed region. Throughout the integration, the anomaly detection model indirectly uses the known attack information to enhance its ability when building profiles of normal behavior. This method needs improvement in terms of detection performance for unknown attacks and detection speed.
Data Mining based [SBUPB13] method to detect unknown malware variants. The model is based on the frequency of the appearance of opcode sequences to detect and classify malware. It describes a weighting technique to mine the relevance of each opcode to malicious and benign executables and assess the frequency of each opcode sequence. It then constructs a vector representation of the executables to train machine-learning algorithms to detect unknown malware variants.

2.1.2 Signature-based

The signature-based detection techniques mainly focus on zero-day polymorphic worms. Polymorphic worms, like other worms have the characteristic that it has some invariant byte stream but the sequence of these bytes is highly random. With every exploit, the worms tend to change the byte stream by removing some code portion, inserting some byte sequence or modifying certain bytes. This characteristic of polymorphic worms poses a great challenge to the security professionals leading to the development of new systems. Several polymorphic worm

signature generation schemes are surveyed. Based on their characteristics, the signatures are classified into 4 categories content-based, flexible content-based, semantic-based and vulnerability-driven signatures.

- *Content-based* detection relies on using byte pattern-based worm signatures to detect worm traffic. When the byte pattern of a given traffic flow matches the byte pattern defined by a worm signature, that traffic is identified as being worm traffic. In order to create these signatures, systems have been proposed to look for common byte stream patterns. These signatures capture the features specific to a worm implementation, thus might not be generic enough and can be evaded by other exploits.

- *Flexible content-based* signature generation systems work on byte level and are flexible in the way that they do not just try to match strings or substrings with incoming packets but, their signatures describe patterns of how malicious bytes are organized. These types of signatures cannot be transformed into Snort signatures. Hence, such systems cannot be deployed widely for online detection.

- *Semantic-based* signature generation approaches go beyond the byte level examination. Instead of using repeated substring found in the network stream, they either use the structure of the executable code present in the network stream or attack analysis information to generate semantic signatures. To identify the semantics-derived characteristics of worm payloads, existing techniques perform static analysis and/or dynamic analysis on the packet payloads to detect the invariant characteristics reflecting the semantics of malicious codes (e.g., behavioral characteristics of the decryption

routine of a polymorphic worm). These signatures are robust to evasion attempts but are computationally expensive to generate as compared to approaches based on substrings. Moreover, semantic-aware signatures cannot be implemented in existing IDS like Snort.

- *Vulnerability-driven* signature captures the characteristics of the vulnerability the worm exploits. It is inherent to the vulnerability and thus is hard to evade. Vulnerability-based signatures analyze vulnerabilities in a program, its execution, and conditions needed to exploit that vulnerability. The signatures based on the vulnerability typically do not change so they are robust against exploits that have variances and can morph. These signatures only require intimate knowledge of the vulnerabilities and can be developed prior to any known exploit, allowing them to be proactive.

2.1.2.1 Content-based Signatures

Polygraph is the first system designed to automatically generate signatures for polymorphic worms. It is based on the assumption that all the instances of the worm consist of multiple invariant substrings [NKS05]. Unlike Autograph [KK05] and EarlyBird [SSS03] [SEVS04], Polygraph uses multiple substrings to detect polymorphic worms which change the sequence of byte stream in every sample. Polygraph generates different classes of signature: *Conjunction signature* consists of a set of unordered tokens (contiguous byte sequences). *Token subsequence signature* consists of a set of ordering tokens. On the other hand, *Bayes signature* use probabilistic matching where each token is associated with score and threshold. The probability of the flow being worm is calculated using

the score of tokens in the flow. If the score is above the threshold it is considered as a worm.

Efficient content-based detection [AAM05] presents a novel method for detecting new worms based on identification of similar packet contents directed to multiple destination hosts. The worm detection method is based on four observations which are commonly found in known worms: diversity of destinations, spread by clients, payload repetition and small size. To identify new worms, the detection technique identifies common substrings appearing in the payload of several client packets, heading for lots of different destinations. This method uses a technique presented by Spring and Wetherall [SW00] for identifying repetitive information transfers using Rabin Fingerprints [Rab81], extended for worm detection. This method detects only TCP based polymorphic worms.

Hamsa [LSC+06] is a network based automated signature generation system for zero-day polymorphic worms which is fast, noise tolerant and attack resilient. Hamsa is based on the fact that well designed worms spread very fast within few seconds and the existing approaches for automatic signature generation are not effective for protection against zero day worm attack as they are not applicable to deploy on the high speed network links. Hamsa claims to outperform Polygraph in terms of efficiency, accuracy and attack resilience. Hamsa is a content based signature generator which allows to treat the worms as strings of bytes and do not depend upon any protocol or server information. Hamsa generates multiset signatures.

A new payload-based IDS [WHKM06] is introduced which, integrates header-based multidimensional flow clustering [WMK06] as front-end processing with content sifting (signature extraction). This method uses front-end clustering

to improve purity of the signature pools and to reduce complexity, making the system less complex than Earlybird and Polygraph. For signature generation this method builds a suffix tree for each suspicious cluster to extract signatures from polymorphic worms.

SweetBait [PB07] is a distributed system that is a combination of network intrusion detection and prevention techniques. It employs different types of honeypot sensors, both high-interaction and low-interaction to recognize and capture suspicious traffic. SweetBait automatically generates signatures for random IP address space scanning worms without any prior knowledge. And for the non-scanning worms, Argos is used to do the job. A novel aspect of this signature generation approach is that a forensics shellcode is inserted, replacing malevolent shellcode, to gather useful information about the attack process.

LISABETH [CLMM08] automatically generate signatures for polymorphic worms, Lisabeth uses invariant byte analysis of traffic content, as originally proposed in Polygraph [NKS05] and refined by Hamsa [LSC+06]. Lisabeth leverages on the hypothesis that every worm has its invariant set and that an attacker must insert in all worm samples all the invariants bytes. Lisbeth and Hamsa systems are equally sensitive to the suspicious flows pool size but Lisabeth is lesser sensible to innocuous flow pool size than Hamsa. Lisabeth has shown significant improvement over Polygraph and Hamsa in terms of efficiency and noise-tolerance.

In Honeycyber [MCV08] a "Double-honeynet" is proposed as a new detection method to identify zero-day worms and to isolate the attack traffic from innocuous traffic. It uses unlimited Honeynet outbound connections to capture different payloads in every infection of the same worm. It uses Principal Component Analysis (PCA) to determine the most significant substrings that are shared between

polymorphic worm instances to use them as signatures [MCV+10] [MP13].
ZASMIN [K+09a] [K+09b] a Zero-day Attack Signature Management Infrastructure is an early detection system for novel network attack detection. This system provides early detection function and validation of attack at the moment the attacks start to spread on the network. To detect unknown network attacks, the system adopted new technologies. To filter malicious traffic it uses dispersion of destination IP address, TCP connection trial count, TCP connection success count and stealth scan trial count. Attack validation is done by call function and instruction spectrum analysis. And it generates signatures using content analysis.
A bioinformatics approach [TXL09] is proposed which generates Simplified Regular Expression (SRE) signature based on multiple sequence alignment. The system addresses the problem of generating accurate exploit-based signature for a single polymorphic worm. First the system analyses and aligns the worm samples and noise flows. This alignment is represented as a coloured matrix in a column, where the greater the number of identical characters in a column, the darker its colour. The next step identifies the noise samples using a noise elimination algorithm. Then the remaining sequences are recognized as worm samples. From them, identical characters in the same columns are extracted as invariant bytes of the polymorphic worms. In the last step an SRE signature is produced by putting distance restrictions between adjacent invariant bytes. Thus, the system generates most specific signature of the worm.
A graph based classification framework [BS12] is proposed for content based polymorphic worm signatures. Also a new signature scheme, Conjunction of Combinational Motifs (CCM) is proposed to detect and create signatures for new versions of polymorphic worms. The scheme is based on conjunction of directed

edges and independent vertices. A vertex in this context, is a common invariant string found in majority of different versions of a polymorphic worm and an edge is directed sequence of two vertices. Firstly, vertices are extracted and vertex score is calculated. Then vertex score is calculated which is a probability of (vertex) appearing in a suspicious flow pool and an innocuous flow pool. So on the basis of this score vertices are differentiated as strong and weak. The strong vertices become part of the CCM signature set. Whereas, the strong edges in the signature set consist of weak vertices. Edge scores are also calculated and a path with maximum total score is discovered and weak edges are removed from that maximal path. The remaining strong edges are extracted as part of the CCM signature set. If at least one of the defined CCM signatures matches completely with a network flow, then a known polymorphic worm is present. If CCM signature partial matches the network flow, then there is a new version of a polymorphic worm with replaced strong vertices and strong edges.

2.1.2.2 Flexible Content-based Signatures

PAYL [WCS06] is an anomaly detection sensor that detects inbound anomalous loads, and correlates them with outgoing traffic on the same ports. This correlation between inbound and outbound traffic is used to generate a signature for the worm. The PAYL anomaly detection sensor computes a "normal profile" of a site using n-grams. When a new packet arrives, all possible n-grams are computed for it and their respected frequencies are registered. Then a simplified Mahalanobis distance [WS04] is computed between arriving packets and the n-gram distribution. If this distance is larger than a threshold and the incoming traffic intended for port i, then such packets are put into a buffer list of "suspects" for

port i. Any outbound traffic to port i, detected as anomalous by the anomaly detection sensor, is compared with this buffer. For the compared strings, a similarity score is computed based on a formula, which requires the generation of the Longest Common Substring (LCS) and the Longest Common Subsequence (LCSeq) of the two strings. If the similarity score is greater than a threshold, the outgoing traffic is blocked. As a by-product of the correlation between inbound and outbound traffic, a signature for the worm is generated in the form of a LCS and LCSeq.

Position Aware Distribution Signature (PADS) [TC07] has a byte frequency distribution instead of a fixed value for each position in the signature "string". The idea is to focus on the generic pattern of the signature while allowing some local variation. PADS signature not only captures the static elements in the executable, but also captures the set of likely values for the variable elements. Network traffic is first partitioned into clusters, each having similar traffic patterns. After partitioning, signature is calculated for each cluster. PADS identify a single significant region in an incoming byte sequence. Rather than a single PADS signature, a set of PADS, called the multi-segment position-aware distribution signature (MPAD) is used to identify the same worm. After computing a first PADS signature for MPAD, the significant region is removed from the worm samples and the next PADS is computed. And this is done until there are no more signatures that can produce good matching scores for all worm samples. When an incoming byte sequence is matched against MPAD, it is classified as a potential worm variant only when it's matching scores with all PADS signatures in the set is above zero.

2.1.2.3 Semantics-based signatures

Structure of Executable Code Detection [KKM+05] approach is presented to generate signatures for detecting polymorphic worms. It is based on Control Flow Graph (CFG) of an executable code. In this type of detection, instead of using repeated substring found in the network stream, structure of the executable code present in the network stream is used. First a linear disassembler extracts a sequence of valid instructions. Then a CFG is created for which a spanning tree is calculated. From this all possible k-node subtrees with a selected basic block as root node are generated. These trees also include non-spanning-tree links. The adjacency matrix of each tree is combined with node colors (14-bit vectors) which provide an indication of the instructions in a basic block. Out of the matrix a fingerprint is computed. The detection part is very similar to the Earlybird approach. The main difference is the mechanism used to index the prevalence table. While Earlybird uses simple substrings, this approach uses fingerprints extracted from CFGs. Thus worms are identified by checking for frequently occurring executable regions that have the same structure.

OSJUMP [WLZ07] a new worm attack model known is proposed. It detects on-line polymorphic worms by recognizing JUMP address using data-mining. This method classifies 4 byte substring at JUMP position flow. For this it uses direct match, Bayes match and Artificial Neural Network (ANN). Direct match method does exact pattern matching of the 4 bytes substring with listed valid values of JUMP addresses. But, this method can be easily violated by selecting a new JUMP address, as in case of Polygraph and Buttercup [PCLW04]. To overcome this disadvantage, OSJUMP, does Bayes matching. Bayes match provides prob-

abilistic matching of 4 byte substring. If the resulting probability is over the threshold, the string is classified to be JUMP address and flow to be a worm. Bayes match does not consider the sequence of 4 bytes therefore; BP ANN is used to classify the substring. ANN algorithm is implemented in C++ and after training the result indicates if a 4bytes substring is a JUMP address or not.
Sting [BNS07] is an end-to-end self-healing system that enables a computer program to automatically self-monitor its own execution behavior and detect errors or intrusions, self-diagnose the root cause of the error/intrusion, self-harden against further attacks and self-recover to a safe state. The Sting system does not require any access to the source code and thus can also work for Commercial Off the Shelf (COTS) software. This self-healing approach combines both proactive and reactive defense mechanisms to protect vulnerable programs and to enable critical services even under extremely fast worm attacks such as hit-list worms.

2.1.2.4 Vulnerability-driven signatures

Length-based Signature Generator (LESG) [WLC+10] is the first network-based vulnerability-signature detection mechanism. Other existing vulnerability-driven schemes are mostly host-based, like COVERS [LS05], Packet Vaccine [W+06], Vulnerability signature [BNS+06] and Vigilante [C+05]. These host-based approaches work on the users' end hosts or honeypots and face problems like scalability and wide coverage. Unfortunately, vulnerability-driven signatures are difficult to generate and due to the increased vagueness of the signature, this method can also lead to more false-positives. Moreover, the existing vulnerability-driven schemes are mostly host-based, and some suffer from computational overhead and are specific to buffer-overflow attacks only. These techniques may be ineffi-

cient if they are not directly based on the exact vulnerability analysis and lack vulnerability details.

LESG is a network-based automatic worm signature generator that generates length-based signatures for zero day polymorphic worms, which exploits buffer overflow vulnerabilities. The system generates vulnerability-driven signatures at network level without any host level analysis of worm execution or vulnerable programs. The LESG's architecture is similar to the basic framework of Polygraph and Hamsa. The network traffic is sniffed and classified as different application level protocols, based on port numbers or other protocol identifiers. For each protocol, known worms are filtered out and the traffic is separated into a suspicious traffic pool and a normal traffic reservoir using an existing flow classifier. The suspicious pool and the normal pool are input to a protocol parser to parse each protocol message into a set of fields. Each field is associated with a type and a length. The field length information of both the pools is then given as input to the "LESG core" module to generate the signatures. A three-step algorithm is designed to generate length-based signatures. It selects candidate signatures, optimizes the signature length for each field and finally, finds the optimal subset of candidate signatures with low false positive and false negative rate.

2.1.3 Behavior-based

Behavior-based techniques look for the essential characteristics of zero-day malware which do not require the examination of payload byte patterns. They focus on the actual dynamics of the malware execution to detect them. They monitor the dynamic behavior of malicious activity rather than its static characteristics

because no matter what, a piece of malware will behave badly while running. Unlike anomaly detection, a program or file is not previously classified as "good" or "bad". The executing processes are monitored to analyze their behavior in a controlled simulated environment. It is an effective way to detect new threats without waiting for them to do any harm.

Network-Level Emulation [PAM06] is a heuristic detection method to scan network traffic streams for the presence of previously unknown polymorphic shellcode. Their approach relies on a NIDS-embedded CPU emulator that executes every potential instruction sequence in the inspected traffic, aiming to identify the execution behavior of polymorphic shellcode. The proposed approach is robust to obfuscation techniques like self-modifications and non-self-contained polymorphic shellcodes [PAM07].

SGNET [LD07] [LD08] is a distributed framework to collect rich information and download malware for zero-day attacks. It automatically generates approximations of the protocol behavior in form of Finite State Machines (FSMs). Whenever the network interaction falls outside the FSM knowledge (newly observed activity), SGNET takes advantage of a real host to continue the network interaction with the attacker. In that case, the honeypot acts as a proxy for the real host. This allows building samples of network conversation for the new activity that are then used to refine the current FSM knowledge.

SBE [HZXF14] is a shellcode detection technique based on Emulation and Support Vector Machine. It comprises of two stages: train and classification. In the train phrase, data (including both shellcode and benign data) is obtained and labeled first, then it is emulated and all features (loop, xor, GetPC) are recorded before trimming redundant features with PCA algorithm, and finally a predictive

model is achieved after training procedure. In the classification phase, network traffic is emulated and classified by the SVM engine with the model acquired before to separate benign and malicious.

Analysis of Opcode Sequences [ZH14] is an approach which can detect new malware. First, executable files are analyzed in order to extract operation code sequences and then n-gram models are employed to discover essential features from these sequences. The iterative SVM clustering and Support Vector Data Descriptions (SVDDs) are applied to analyze feature vectors obtained and to build a benign software behavior model. This model is then used to detect new malicious executable files.

ENDMal [LWZ+13] is an anti-obfuscation, scalable and collaborative malware detection system. It utilizes the three basic intrinsic characteristics of a malware family to distinguish malware from benign. ENDMal consists of multiple monitors where each monitor takes charge of a network area and receives suspicious programs from end-host. Each monitor uses Iterative Sequence Alignment (ISA) method to defeat malware obfuscation and utilizes Handle dependences and Probabilistic Ordering Dependence (HPOD) technology to represent the program behaviors. All the monitors collaboratively identify the malicious program families by sharing HPOD-based behaviors via RENdezvous-based Sharing infrastructure (RENShare), based on Distributed Hash Tables (DHT).

2.1.4 Hybrid Techniques

Other surveyed techniques are basically hybrid techniques. They combine heuristics and different intrusion detection techniques like signature-based and anomaly-

based to detect zero-day polymorphic worms.

Cooperation of Intelligent Honeypots is a technique consisting of two types of honeypots; Cooperation based active honeypot and Self-Protection type honeypot. They are used to collect unknown malicious codes automatically while maintaining their concealment against malicious attackers [STO08]. Cooperation based active honeypot comprises of two parts: Observation Appliances and a Control Server. Each Observation Appliance has three components (i.e. TAP, Honeypot and FireWall), and a Control Server manages cooperation among the honeypots. Self-Protection type honeypot consists of mainly two components: Reboot Controller and Honeypot. The Reboot Controller monitors all outgoing traffic data by Snort, and if some serious attacks are observed, it commands Honeypot to reboot its operating system. Honeypot then reboots itself by using bootable OS images included in DVD-R, so that the system can be fed back to its initial state even if the system was compromised.

Hybrid Detection for Zero-day Polymorphic Shellcodes (HDPS) [TXZ09] is a hybrid detection approach. It uses an elaborate approach to detect NOP Sleds to be robust against polymorphism, metamorphism and other obfuscations. It employs a heuristic method to detect return address, and achieves high efficiency by incorporating Markov Model to detect executable codes. This method filters normal packets with accuracy and low overload. But this approach cannot block shellcodes in network packets and it is hard to obtain transition matrixes of Markov Model.

Honeyfarm [JS11] [JS12] is a hybrid scheme that combines anomaly and signature detection with honeypots. This system takes advantage of existing detection approaches to develop an effective defense against Internet worms. The system

works on three levels. At first level signature based detection is used to filter known worm attacks. At second level an anomaly detector is set up to detect any deviation from the normal behavior. In the last level honeypots are deployed to detect zero day attacks. Low interaction honeypots are used to track attacker activities while high interaction honeypots help in analyzing new attacks and vulnerabilities. The controller is responsible to redirect suspicious traffic to respective honeypots which are deployed in honeyfarm.

2.1.5 Comparative Analysis

The comparison between surveyed techniques based upon their advantages and disadvantages is shown in Table 2.1. It also highlights the unique contribution of each surveyed technique.

Table 2.1: Comparative Analysis: Pros_Cons

Technique	Pros	Cons	Contribution
Polygraph	Robust. Efficient. Low false positive rate.	New signatures are not supported by traditional IDSes. Computationally costlier. Not for Zero-day attacks. Detects only polygraph-specific attacks.	First to define the signature generation problem for polymorphic worms. Presented a suite of novel algorithms for automatic generation of signatures that match polymorphic worms.
Efficient content based	Low false positives.	Detects only TCP based worms. Cannot detect polymorphic worms. Can be exploited easily. Computional overhead. Detection delay.	Presented a novel method for detecting new worms based on identifying similar packet contents directed to multiple destination hosts.

Continued on next page

Table 2.1 – continued from previous page

Technique	Pros	Cons	Contribution
Hamsa	Efficient. Accurate.	Can be evaded easily. Prone to pool poisoning. Parameters are manually adjusted	Outperforms Polygraph. First to provide analytical guarantees for polymorphic worm signature generation systems.
Payload based IDS	Accurate. Low false alarms. Low complexity.	No unknown worm was tested.	Proposed an improved defense system. Addresses the issue of devising a proper methodology for polymorphic worm IDS testing.
SweetBait	Distributed and Open design. Requires no prior knowledge of attack types. Low response time. Accurate. Zero false positive rate. Refined signatures.	Crude signatures. Computationally expensive. Prone to spoofing attacks. Cannot work for encrypted traffic. Consider only monomorphic n/w signatures. Scalability and availability issues.	Pioneers to use methods used by worms for forensics. First to consider worm virulence rather than its presence.
LISABETH	Better than Polygraph and Hamsa. Resilient against attacks.	Not distributed. No Zero-day attack was detected.	Outperforms Hamsa. Able to improve the resilience to poisoning attacks.
Honeycyber	Low false positives. Minimum resource requirement	Require multiple worm instances before generating signatures. Implementation of proposed idea is missing.	Ability to distinguish worm activities from normal activities without the involvement of experts.
ZASMIN	Applicable to high-speed networks.	System was not well tested.	Presented an early detection and validation system for novel network attacks.
Bioinformatics	Noise tolerant. Few false positives. Specific signatures. Signature compatibility with current IDSs.	Cannot detect worms with all no invariant bytes. Can suffer from specially designed adversary attacks.	Proposed novel signature generation for zero-day exploit based polymorphic worms using a bioinformatics approach.

Continued on next page

Table 2.1 – continued from previous page

Technique	Pros	Cons	Contribution
Graph-based	Resilience to new versions of polymorphic worms. Low false positives and low false negatives. Good flow evaluation time performance.	Cannot detect complete zero-day polymorphic worms with new vertices and edges. Signature generation may take time if enough worms are not captured in suspicious pool.	Classified content based polymorphic worm detection signatures and proposed a new polymorphic worm detection signature scheme.
PAYL	Efficient. Low false positive rate.	Not well tested. Cannot detect altered attack code.	Accurately detects "zero-day" attacks upon their very first appearance, or very soon thereafter.
PADS	Flexible. Precice. Low false positive rate.	Cannot detect sophisticated worms. Signatures cannot be transformed into Snort signatures.	Introduced a new worm signature which fills the gap between traditional signatures and anomaly-based intrusion detection systems.
CFG	Robust. Can detect polymorphic worms that do not contain invariant strings at all.	Cannot detect worms with very small CFG. Cannot be implemented in real time. Computaionally expensive. Only for x86 architecture.	Described a novel fingerprinting technique based on control flow information of executable code.
OSJUMP	Low false negative rate. Low performance overhead.	False positives may increase if innocuous flow contains JUMP.	Proposed a worm attack model. Can detect perfect polymorphic worms that changes JUMP address.
Sting	Does not require access to source code. Zero false positives. Proactive. Reactive.	Performance overhead. Prone to brute-force attack.	First to design and develop a complete architecture capable of defending against hit-list worms. First to realize a self-healing architecture for COTS software.
LESG	Fast. Noise-tolerant. Efficient. Cannot be easily evaded.	For buffer-overflow attacks only. Computaionally expensive . Lack many vulnerability details. But	First to generate vulnerability-driven signatures at network level.

Continued on next page

Table 2.1 – continued from previous page

Technique	Pros	Cons	Contribution
dSHT	Provides holistic solution. Fault tolerant.	Depends on quality of local worm detectors. False alarms. Issues in parctical implementation of the model.	Detect as well as response from a content-independent perspective.
Via IDS Alerts	Simple	Cannot be applied to all IDS products. High false positive rate. Requires manual adjustment of detection parameters. Cannot detect 0-day attack if IDS does not raise any alert.	Introduced a feature extraction method to detect zero-day attack from IDS alerts instead of investigating raw traffic.
RSWD	Low computation overhead. Generates minimum set of blocking rules.	Manual adjustment of detection parameters. Effective only for known worms than zero-day attacks.	Capability to identify worm traffic under evading of dynamical context mutation. But the the detection method fails to manifest the detection of any unknown worm.
PCA-based	Requires no prior knowledge of attack types. Low computational requirements.	Incapable to adapt over time changes in correlation structure. Requires manual extraction of detection parameters.	First to use PCA to detect new attacks using honeypot traffic.
Supervised Learning	Signature-free system. Robust to obfuscation. Accurate and efficient.	Uses only static analysis to understand program behavior. Neglects runtime API call sequence and hidden execution paths.	Provides a novel method of employing several data mining techniques to detect zero-day malware.
Contextual Anomaly Detection	Accurate in detecting both known and unknown attacks.	Half of the execution mode works off-line. Takes time detecting zero-day attacks.	Quite effective in minimizing the rate of false positives.

Continued on next page

Table 2.1 – continued from previous page

Technique	Pros	Cons	Contribution
Combined Supervised & UnSupervised	Effective detection and classification of attacks. Handles data imperfection issues. Capable of discriminating new malware from known variants	Off-line learning. Prediction suffers with extremely large number of malware classes (hyperspheres). Manual threshold setting may result in more false positives.	Presented a novel machine learning framework by combining both supervised and unsupervised learning.
Hybrid Model Correlate Alerts	Accurate and Efficient to correlate alerts.	Set parameters manually. Requires human interference to upadte attack graphs. Time complexity.	Ability to correlate alerts of known attack as well as alerts of unknown attacks albeit at lower speed.
Unsupervised Anomaly Detection	Robust to an increase of attack data within the training data. Training and testing time is short.	Performance is not verified. System is not verified with real traffic data.	Proposed a new anomaly detection method which can automatically tune and optimize the values of parameters without predefining them.
Integrated Anomaly and Misuse	Low false positives. Reduces the time to train and test anomaly detection model.	Unable to decompose the normal data evenly. Degrades the misuse detection performance. Requires manual adjustment of parameters to increase detection rate.	Proposed a new method that hierarchically integrates a misuse detection model and an anomaly detection model in a decomposition structure.
Data mining	High detection rates. Low false positives.	Not robust against obfuscation. Long opcode sequences introduces high performance overhead.	Used an opcode-sequence-frequency representation of executables to detect and classify malware.
Emulation-based	Robust to obfuscation techniques. Zero false positives. Low cost.	Not robust against obfuscations such as non-self modifying and indirect control transfer instructions.	Presented a novel method to detect decryption routine of shellcodes by emulation.
SGNET	Open architecture. Scalable. Easy to install and configure.	Min IP space coverage. Broken malware downloads	SGNET is an open initiative, integrating together tools of different research teams, where anyone can participate.

Continued on next page

Table 2.1 – continued from previous page

Technique	Pros	Cons	Contribution
SBE	Robust. Independent of specific shellcode features.	Performance Overhead. Detection speed falls down for bigger datasets and require optimizations.	Proposed an novel shellcode detection method to detect both polymorphic shellcode and plain shellcode.
Analysis of Opcode Sequences	High accuracy rate as compared to similar existing techniques	Requires more time for training if training set contains large number of files.	Detect malicious executables within new files by analyzing opcode sequences.
ENDMal	Robust against obfuscation. Scalable and collaborative.	Identifies legitimate programs as malware.	Proposed a novel malware behavior representation to handle program dependencies and ordering between syscalls.
Cooperation of Honeypots	Secure. Accurate.	Unable to trace attacks to well-known ports. Cannot work against sequential scan for closed ports. Requires manual closing of ports.	Worked on maintaining the concealment of honeypots against malicious attackers. Is not reliable in detecting zero-day attacks.
HDPS	Robust against obfuscations. Efficient. Low overhead.	Cannot block shellcodes in Network Packets. Hard to obtain transition matrixes of Markov Model.	Proposed a hybrid detection approach against zero-day polymorphic and metaphophic worms.
Honeyfarm	High detection rate. Includes advantages of existing detection techniques.	High initial setup time. False alarm rate is significant.	Proposed a novel hybrid approach that integrates anomaly and signature detection with honeypots.

To contain zero-day attacks, zero-day malware analysis is must. It is one of the most critical efforts in network security. After detection, malware analysis is required to isolate, examine and mitigate zero day attack. Malware analysis identifies the unknown behavioral patterns which are later used to defend against zero-day attacks in future. Next section discusses techniques and methodologies utilized to respond to zero-day attacks.

2.2 Malware Analysis

Malicious software or malware is an integral component in security breaches. Any software that does something harm to a user, computer, or network can be considered as a malware, including viruses, trojan horses, worms, rootkits, ransomware, spyware, adware, scareware, and other malicious programs. To turn malicious programs inside out and to understand their inner workings a core set of tools and techniques is required for analyzing. Malware analysis is the art of dissecting malware to understand how it works, how to identify it, and how to defeat or eliminate it [SH13]. Malware analysis is a critical task for responding to computer or network security incidents as it allows to better assess the nature of a security incident and may even help to prevent further infections. Analyzing malware, especially when trying to achieve deeper insights on internal functionality, is a time-consuming task and it usually involves notable manual effort which by itself requires significant expertise to be carried out.

With the organizational perspective, knowing how to analyze malware helps to understand the context of the incident, its severity and repercussions. It assist to plan incident response, incident's scope and in some cases, understand what entities might be behind the intrusion. That is why today, malware analysts are no longer just anti-virus and threat researchers, but also system and network administrators, as well as general security professionals. They are required to understand the capabilities of malware that their organizations discover. Also, knowing how to analyze malware brings an element of control into an otherwise chaotic environment that exists around a security incident [Zel10].

Goals of Malware Analysis:

The primary goal of malware analysis is to provide the information needed to respond to a network intrusion. This information is required to understand how a specific malware functions so that existing defenses can be evolved or new defense systems can be built to protect an enterprize network. While analyzing the suspected malware, the aim is to answer various questions [KM07] such as:

- What actions the malware performed on the system that leads to system compromise?
- What exactly does the malware does?
- How does it spread?
- How does it communicate with the attacker?
- Did an attacker implant a rootkit or trojan on the systems?
- How to measure and contain its damage?
- How to detect it on the network?
- What is the sophistication level of the malware author?
- What other machines or network resources were affected by the same malware?

All these questions can be answered by analyzing the malware in a controlled environment. Once its clear which computer resources require full analysis, the next step is to to develop signatures to detect malware in the future. Malware

analysis can be used to develop both host-based and network signatures. Host-based signatures identify files or registry keys on a victim computer that indicate an infection. It focuses on what the malware did to the system, not the malware itself. Network signatures detect malware by analyzing network traffic. Malware analysis improves the quality of network signatures, resulting in high detection rate and few false positives.

Types of Malware Analysis:

There are basic two approaches for malware analysis as depicted in Figure 2.3, that security professionals perform: Static Malware Analysis and Dynamic Malware Analysis. Both these types accomplish the same goal of describing how malware works but, the tools, time and skills required to perform the analysis are very different.

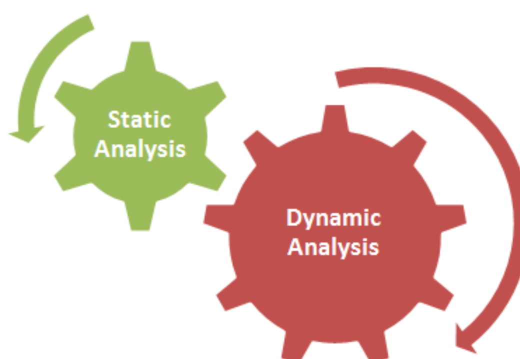


Figure 2.3: Types of Malware Analysis

2.2.1 Static Analysis

The static analysis allows to learn malware's capabilities by examining the code from which the program was comprised. While performing static analysis antivirus software is run to confirm maliciousness, hashes are used to identify malware, strings are searched, functions, headers and scripts are analyzed. Static analysis is mostly conducted manually and can be applied on different representations of a program. If the source code is available, information such as variables, data structures, used functions and call graphs can be extracted. Static analysis tools can even help to find memory corruption flaws and to prove the correctness of models for a given system. Static analysis is also used on the binary representation of a program. While compiling the source code into a binary executable, some information such as size of data structures or variables gets lost [ESKK12]. This loss complicates the further task of analyzing the code. Static analysis is tricky and time-consuming, because source code of malware is not always available. Instead, the complied executable's functionality is examined at the assembly level using a disassembler, which converts the instructions from their binary form into the human-readable assembly form [Eag11].

Advantages:

- Static analysis is fast and safe as the source code is not actually executed.
- Static analysis gathers the structure of code of program under inspection.
- Static analysis allows a comprehensive analysis of a given binary i.e, it can cover all possible execution paths of a malware sample.

- Static analysis can provide first level report on malicious behavior of a binary, which is helpful for further investigation.

Disadvantages:

- Static analysis alone is not sufficient to analyze the unknown malware.
- Static analysis requires code inspection but, the source code of malware is not easily available.
- Static analysis is time-consuming and cumbersome.
- Static analysis demands an expert knowledge of assembly language and deep understanding of Operating System functionality.
- Static analysis suffers from code obfuscation and packing techniques. The evade this issue is to actually execute the binary, which leads to dynamic malware analysis.

Static Analysis Techniques

Various static malware analysis methods have been proposed [CJ03] [CJS+05] [Fla04]. Static analysis offers a significant improvement in malware detection accuracy while compared to traditional pattern matching. But its main weakness lies in the difficulty to handle obfuscated and self-modifying code [MKK07]. Eureka [Sea08] provides a malware de-obfuscation framework, to assist in static analysis. It uses a novel binary unpacking strategy based on statistical bigram analysis and coarse-grained execution tracing. MaTR [Dea12] combines machine learning algorithm with static heuristic features for unknown malware detection. A program analysis tool [ZW14] is proposed to automatically derive data invariants from source code, using static analysis. The tool applies compiler technology

to analyze the control and data flows (e.g., assignments, function calls, and conditional statements) of a target program and hypothesizes likely invariants (e.g., constant, membership, bounds, and non-zero). API-CFG [EH12] extracts control flow graphs from programs and combines it with extracted API calls to have more information about PE files. Another approach uses static classification to analyse and classify malware [SBGS15]. This approach extracts static features like function call frequency and opcode frequency from the header and payload of malware and classifies malware using machine algorithms.

2.2.2 Dynamic Analysis

During dynamic analysis it is examined how the malware behaves and interacts with its environment when executed. In dynamic analysis the malware is executed on an isolated or virtual system, its interaction with overall system including file system, registry, system processes and network is observed. Sometimes, it is required to interact with the malware to discover its additional characteristics and for this debuggers are used to examine the internal state of a running malware. Generally, there are two main approaches for dynamic malware analysis. (1) *Analyzing the difference between defined states*: In this approach a given malware is executed for a particular time period and later on the modifications made to the system are analyzed by comparing the current system state to the initial system state. In this, comparison report states behaviour of malware. (2) *Observing run-time behavior*: In this approach, malicious activities initiated by the malware are monitored during execution time using a specialized tool [Hol09].

Advantages:

- Dynamic analysis is insightful and relatively fast.
- Dynamic analysis allows to easily detect the unknown malware by simply analyzing the behavior of the running binary.
- Dynamic analysis evades the restrictions of unpacking and obfuscation issues because dynamic analysis is performed during runtime when malware unpacks itself.
- Dynamic analysis can be automated, enabling analysis at a large scale.

Disadvantages:

- Dynamic analysis fails to detect multipath malware which depicts different behavior by different triggering conditions.
- Dynamic analysis is dangerous and can harm other systems if the analysis environment or system is not properly isolated or restricted respectively.
- Dynamic analysis suffers from anti-analysis techniques which malware authors use to alter malware behaviour.
- Dynamic analysis may miss details about malware of moderate and advanced complexity. That's where static analysis comes to play and can highlight additional properties of the malware that may have not discovered behaviorally. Therefore, both static and dynamic analysis are performed to gain a complete understanding on how a particular malware functions.

Dynamic Analysis Techniques

An in-depth survey of the different dynamic analysis techniques as well as ways how to implement them can be found in [ESKK12]. This section discusses the most common techniques.

Function Call Monitoring and Hooking: Functions call monitoring is about tracking the functions called by a given malware program to gain an overview about its behavior. One way is to intercept these calls by a process known as **hooking**. The analyzed malware program is manipulated to add an additional function called the hook-function, that implements the required analysis routines such as recording the invocation activity to a log file or analyze input parameters. A common type of hooking for dynamic analysis is **API-hooking**. The application programming interface (APIs) are instrumented for dynamic analysis by monitoring all the relevant system calls and their parameters. Much work has been done in monitoring system calls as it provides useful information about a program's behavior.

Function Parameter Analysis: Beside the actual function call, the parameters and their relations can be also taken into account. Function parameter analysis tracks the values that are dynamically passed/returned during a function call. This tracking allows the correlation of different function calls, which are related to the same object. Grouping these calls provides an insight into the malware behavior from an object-centric perspective.

Information Flow Tracking: It addresses the aspect of how a given malware actu-

ally handles the data. The monitored data is labeled tainted with a corresponding label and is tracked for all manipulations throughout a program's execution. During execution, once the data is processed the taint label is transmitted as well. Information flow tracking introduces three concepts: Taint source (the interesting data source), Propagation rules (the how-to) and Taint sinks (where the data flows to and allow to enforce security policies). Taint analysis forms the basis of many dynamic behavior malware analysis framework like Panorama [Y+07].

Instruction Trace: The sequence of machine instructions that the program executed while it was analyzed is called instruction trace. This is a valuable information source for a malware analyst because it can reveal additional information regarding the malware behavior that is not represented by analysis report of system and function calls.

Execution Environment

Following are the common malware execution environments in demand. Each has its own advantages over the other in achieving visibility and stealth.

Emulator: Its a software or hardware that imitates a full computer system or its resources like CPU, memory, system services, etc. Emulator allows the analysis component to control every aspect of a running malware by implementing it as part of the emulator. Which part of executing environment is emulated results in different form of analysis: *Memory & CPU Emulation* results in a *Sandbox*. In this a malware is executed by successively reading its instructions and performing equivalent operations in the emulated environment and the side effects of mal-

ware execution are contained in the sandbox. This type of emulation is usefull in case of packed or obfuscated malware. *Full System Emulation* provides the functionality of a real computer system including all required peripherals. For eg., Bochs is an well known open source emulator [Boc15] allowing a complete installation of any Operating System. The OS that runs in the emulator is *guest*, while the computer executing the emulator itself is called the *host*. The full system emulator performs analysis in a stealthy manner while monitoring the guest OS and all executing applications. Thereby, the analysis component can stay undetected. A major key problem in this approach is to "bridge the semantic gap" between the analysis approach and guest OS. Therefore, some information is extracted from the hardware level in order to map OS semantics [BMKK06].

Virtual Machine (VM): It is an isolated duplicate of the real machine. A host program known as a virtual machine monitor (VMM) is responsible for presenting and managing the programs to the virtual machine, and is in charge of the underlying hardware. It does not provide any VM direct access to privilege level of the real machine. In malware analysis, commodity virtual machine monitors (VMMs) such as Xen and VMWare are being used. They deploy analyzing tools in conjunction with virtualization technology, taking advantage of its strong isolation, and its ability to take snapshots and roll back the guest's state [N+09]. The analysis component is integrated directly into the VMM or occurs in additional VM. In the later case, the VMM has to send all necessary information to the analysis VM to perform its task. Since, the VMM only has access to state of VM and no knowledge about the structure of underlying OS, the informational gap between the VMM and the VM's perspective needs to be bridged. The VMM

derives the information from the raw data in the memory of the VM by accessing hardware based information to extract the relevant structures such as memory pages, states and registers. This knowledge is then used to map the information to OS semantics.

Table 2.2 summarizes the major difference between an emulator and a virtual machine.

Table 2.2: Difference between Emulator and Virtual Machine

Emulator	**Virtual Machine**
Imitates a real machine	Duplicate of a real machine
Emulates hardware like processor, memory, mass storage, network cards	Provides a virtualized interface to the real hardware
Malware is not executed directly on real processor.	Executes subset of instructions directly on real processor
Speed of execution may be problem	Run few instructions on real hardware, performance can be a issue
Offers complete control	Provides strong isolation
Difficult to detect by malware	Easy to detect than emulator

Network Simulation: Malwares require Internet access for their operation. For example, after compromising a system, a malware might download additional components or data before performing its villainous actions. Any approach that denies or allows all Internet traffic normally gives unwanted analysis results. Denying all Internet access may result in incomplete observation of malware's network activity and on the other hand, providing full Internet access may induce the malware to spread over the network. The solution is to provide a restricted

network access to the malware interacting with a remote host and simultaneously observing its behavior from within a safe environment. One such approach is to use *simulated network* in which commonly used network services like DNS, HTTP, SMTP, IRC, etc, are simulated and all traffic is redirected to these services. Another way is to grant *filtered Internet access* by limiting outbound and inbound network connections for a malware. This way the malicious effect and volume of malware generated traffic will be restricted.

Tools for Malware Analysis

This section introduces the automated dynamic malware analysis tools and frameworks best known in the research community. A brief summary of each tool is presented. These tools execute an unknown malware in an instrumented environment and monitor its execution. They save time and provide valuable insights about specimen's capabilities. The analysis reports generated by these tools provide valuable insights about the behavior of running malware. This information further helps in developing security solutions in a timely manner.

Anubis: Anubis stands for Analyzing Unknown Binaries focuses on automated dynamic malware analysis. It evolved from TTAnalyze [KKB06] and executes the sample under analysis in an emulated environment consisting of a Windows XP OS running as the guest in a modified version of Qemu [Bel05]. The analysis is performed by monitoring the invocation of Windows API functions, as well as system service calls to the Windows Native API. The parameters passed to these functions are also examined and tracked. Apart from analyzing, Anubis is also capable of clustering malware samples into families according to their behavior

[B+09].

Cuckoo: Cuckoo Sandbox is an open-source tool for dynamic malware analysis that uses the technique of API-hooking. The actual instrumentation of the running processes is done by injecting a dynamic linked library (DLL) that hooks Windows API functions and logs their parameters when called. This DLL also randomize the instructions written to the target function in order to evade anti-analysis techniques used by modern malwares [Cuc13].

CWSandbox: It uses API-hooking and code injection technique to analyse malware dynamically. CWSandbox executes the malware either natively or in a virtual Windows environment. The sandbox injects a monitoring DLL in the malware process, which implements API hook functions to trace relevant system calls. On the basis of these techniques CWSandbox generates an automated report that describes the behavior of malware sample with respect to file system and registry manipulation, network communication, and operating system interaction [WHF07].

Norman Sandbox: It emulates whole computer and a network connected to it. Norman Sandbox executes the sample in a tightly-controlled environment that simulates a Windows OS, attached local area network (LAN) and some Internet connectivity. Norman Sandbox focuses on the detection of email or P2P worms, as well as viruses that replicate over network shares [ESKK12]. Norman Sandbox also uses function call hooking and parameter monitoring techniques to detect malware [Nor14].

Joe Sandbox: Previously known as JoeBox [Joe14] run on a real hardware. It logs high-level information of the performed actions regarding file system, registry, and system activities. Joe Sanbox uses client server model, where a single controller instance coordinates multiple malware analysis clients and all analysis data is collected by the controlling machine. The throughput of complete system increases by adding more analyzing clients [ESKK12].

Next section presents Honeynet, which is one of the widely used framework to defend against zero-day attacks. It helps in zero-day attack detection and provides data for analysis at one place. Honeynet defense is based on analysis of the exploit's interaction with the target. Analysis data captured helps in predicting future malicious activity and in classifying exploits into behavior groups. Hence, Honeynet has the potential to detect and analyze zero-day exploits in real time.

2.3 Honeynet: A Detection and Analysis Tool

Honeynet is a collection of diverse high-interaction honeypots to capture in-depth information [Pro06]. It is a network setup deployed with intentional vulnerabilities, whose purpose is to enlarge the potential attack surface in order to study an attacker's activities and methods. Therefore, any traffic entering or leaving a honeynet is plain suspect. This specialized network architecture is aimed to fulfill the following requirements:

- Data control deals with the containment of malicious activity within the honeynet. It is specifically important to block any malicious attempt from

a honeypot to a non-honeynet systems. Techniques includes: counting outbound connections, network intrusion detection/prevention and bandwidth throttling.

- Data capture covers the logging and monitoring of all activities within the honeynet. It captures a variety of activity such as network activity, application activity and system activity, initiated by an attacker or a malware.

- Data analysis is a critical part and deals with the analysis of all the information captured from different elements while meeting the appropriate data analysis needs of the corresponding organization.

- Data collection applies to distributed honeynet deployment. It deals with the secure forwarding of all gathered information to a central collection point i.e., a repository or database.

To track the rapid evolution of zero-day attacks timely intelligence is essential for detection and analysis. This requires a dedicated framework for early and accurate detection with a meticulous analysis. *Gen-III Honeypot* is one such set up that allows detection of zero-days and provides data for analysis through Walleye interface [Pro05] . If honeynet encounters a known attack, honeywall blocks the attack and log its details. And, if a honeynet is hit by a zero-day attack, the honeywall redirects the unknown traffic to high-interaction honeypots with Sebek installed on them. The Sebek tool [Pro03] is implemented as a hiden kernel module that enables the analysis of encrypted communication by capturing data prior to encryption. The entire inbound and outbound communication of attacker is monitored and logged. For assisting manual analysis, walleye interface provides

unified view of network flow connection records, IDS events, OS Fingerprints and represents host process activity as process tree. It allows the analyst to easily navigate between network and host data and to visualize multiple data types together. For eg., the analyst can make use of following data analysis capabilities: (i) For a outbound connection get the information on related inbound connection. (ii) For an inbound connection get the related host activity. (iii) For a particular flow get the corresponding packet trace. (iv) For a host process get the keystrokes entered by an attacker.

2.4 Problem Formulation

As per the literature reviewed, the existing zero-day attack detection techniques do not make things harder for the attackers, while their cost for the defender in terms of resources that are needed for detection are significant. Several research projects have addressed the problem of Zero-day attack detection but unfortunately they exhibit one or more problems of: false positives, robustness, accuracy, efficiency and computation overhead. According to the survey their exists certain research proposals that have been promising, but they can also be easily defeated by using minor enhancements to the attack vectors. As attack tools are improving, reliance on minor improvements in the detection processes is insufficient.

Statistical-based detection techniques cannot be used for instant detection and protection in real time. They are dependent on static attack profiles and requires manual adjustment of detection parameters. Signature-based techniques are widely used but, need improvement in generating good quality signatures. They suffer from one or more limitations of high false positives, false negatives,

reduced sensitivity and specificity. Behavior-based techniques may detect a wide range of novel attacks but they are prone to evasion once the behavioral analysis method is known. They are computationally expensive and may not effectively capture the context in which the new attacks interact with the real victim machine. Other hybrid techniques combine heuristics and different intrusion detection techniques like signature-based, anomaly-based, etc. to detect zero-day attacks but they also suffer from high false positives, false negatives.

Another improvement in this area is to develop an integrated automatic zero-day malware analysis and reporting facility. Malware analysis is more of a manual process that is tedious and time-intensive. Also as the number of samples and diversity of malware that need to be analyzed has constantly increased, automatic malware analysis became a necessity. Today, the malware analysts requires an isolated environment comprising of physical or virtual systems, various behavioral-analysis, code-analysis and online analysis tools for dissecting the malware. This involves usage of numerous different independent tools resulting in scattered analysis information which needs to be assembled manually into more useable form. Thus, the data integration of such tools/utilities will play an important role in future of malware analysis.

In this work, the proposed system RADAR achieves the above mentioned vision of an efficient zero-day attack detection and analysis technique by bridging the gap between "attack detection phase" and "malware analysis phase". RADAR combines all the three zero-day attack detection techniques, statistical-based, signature-based and behavior-based. It also implements an analysis and reporting stub to analyze zero-day binary. This stub integrates existing static and dynamic malware analysis functionalities to work as a single unit in a component based

architecture. Moreover, detection and analysis approaches are incorporated in a layered design to improve system performance.

2.5 Thesis Objectives

The objectives of this research are:

i. To identify, explore and analyze existing Zero-day attack detection techniques.

ii. To propose efficient Zero-day attack techniques for better sensitivity and specificity.

iii. To design and implement analysis and reporting stub.

iv. To verify and validate the proposed techniques.

2.6 Summary

This chapter described the related research done in the area of zero-day attack detection. The zero-day attack detection techniques are categorized as statistical-based, signature-based, behavioral-based and other hybrid techniques. The statistical techniques focuses on extracting mathematical properties from network activities and constructing intrusion detection models or profiles to detect unknown activity. Signature-based techniques primely focuses on detection and generating signatures for zero-day polymorphic worms. Behavior-based techniques look for behavioral attributes or indicators of a zero-day malware rather than monitoring

patterns in the network packet. Hybrid techniques combines different detection techniques to do their job.

After zero-day attack detection the next crucial step is to do detailed analysis of the captured binary. Therefore, the chapter also discusses malware analysis techniques. Malware anaysis is the study of malicious program by dissecting its components to study its behavior on physical or virtual computer system. There are two main techniques for malware analysis: Static analysis involves studying the program code without executing it. Dynamic analysis is done by monitoring and logging the behavior of a running binary in a controlled environment. Different execution environments and automated dynamic malware analysis tools are also explained.

Later part of the chapter presented honeynet as a detection and analysis tool. Honeynet is a dedicated framework for accomplishing zero-day attack detection and provides data for analysis at one place. Chapter is concluded by defining the problem statement.

Chapter 3

RADAR- Runtime Zero-day Attack Detection Analysis and Reporting

3.1 Introduction

RADAR is an online system for zero-day attack detection, analysis and reporting. It integrates three zero-day attack detection techniques: statistical-based, signature-based and behavior-based techniques. RADAR implements an analysis and reporting module to analyze the captured zero-day binary. RADAR has a layered and modular design which helps it to achieve high performance, flexibility and scalability. It is designed to address the research problems with existing approaches in the field of zero-day attack detection and malware analysis, and tries to provide a complete solution to the whole problem.

This chapter is organized as follows: Section 3.2 discusses the evolution of

RADAR and explains the issues with the previous proposed approach. It also list the design principles for contriving the RADAR system. Detail architecture of RADAR is presented in Section 3.3. RADAR has a layered design where each layer (Detection Layer, Analysis Layer and Resource Layer) is dedicated to a single functionality. Detection Layer and Analysis Layer are discussed in detailed in Section 3.4 and Section 3.5 respectively. Section 3.6 summarizes the entire chapter.

3.2 Evolution of RADAR

RADAR system was developed after various iterations. The first proposed technique [KS14a] [KS15] consists of three basic components: Suspicious Traffic Filter (STF), Zero-day Attack Evaluation (ZAE) and Signature Generation (SG) as shown in Figure 3.1 This technique combines three intrusion detection techniques: anomaly detection in STF, behavior-based in ZAE and signature-based in SG.

- *Suspicious Traffic Filter (STF)*: This module is responsible for capturing the network traffic, filtering known attacks and finding suspicious network traffic for new attack. STF make use of anomaly detector to do the required job. The first proposed system uses Honeynet as an anomaly detector. The network traffic is passed simultaneously to both Honeynet and IDS/IPS sensors as in Figure 3.2. Honeynet capture unknown traffic traces and store them in a repository. Similarly, the IDS/IPS sensor filter known attacks and store rest of the filtered traffic in another online repository. The data of both repositories are then compared and analyzed to separate unknown

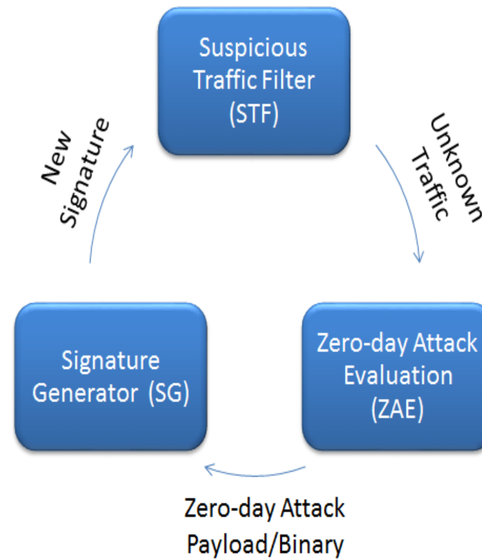


Figure 3.1: Basic System Components

traffic from the benign traffic. By comparing data of repositories a low-level attack validation is done at the first stage.

- *Zero-day Attack Evaluation (ZAE)*: In this module, the unknown traffic is evaluated for its malicious intentions. This is done by executing the unknown traffic trace in an emulator for per byte execution as shown in Figure 3.3. The trace is allowed to perform malicious activities inside the emulated environment. After executing, the system anomalies are analyzed by comparing the system status information to a standard. For this, the abstract method of analyzing system anomalies is used that is validating checksums of critical files like registry files, system configuration files, password files, system binaries etc. This analysis is based upon the fact that no matter what, malicious code will do one of three things: add, remove or modify system files, while running on the system. On the other hand, if no

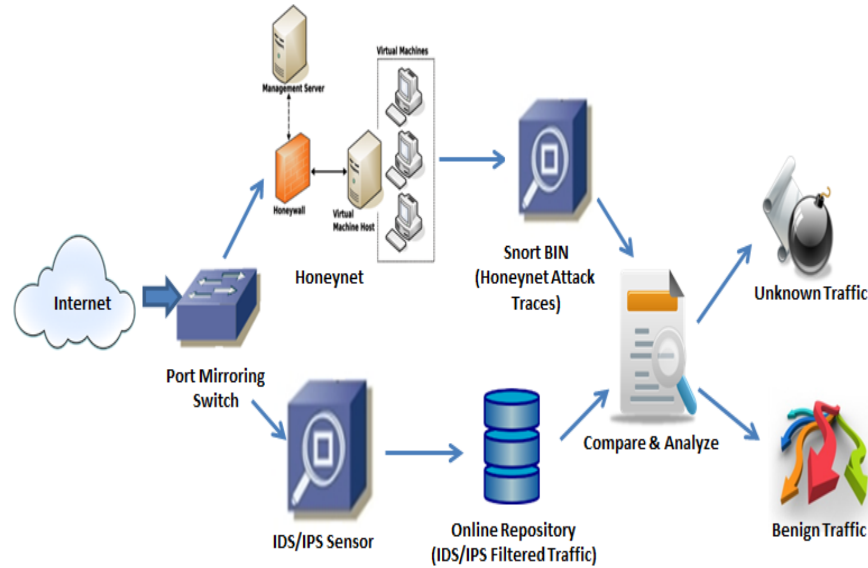


Figure 3.2: Suspicious Traffic Filter (STF) Module

critical file is changed then the candidate is false positive and the whitelist is updated.

- *Signature Generation (SG)*: After evaluation zero-day attack packets are fed to next module for signature generation. This module generates a common token-subsequence signature for a set of attack packets by applying the Longest Common Subsequence (LCSeq) algorithm. The algorithm compares two zero-day attack packets to get the longest common subsequence between them. Let two sequences be defined as follows: X = (x1, x2...xm) and Y = (y1, y2...yn). Let LCSeq(Xi, Yj) represent the set of longest common subsequence of prefixes Xi and Yj. The new attack signatures generated by ZAD (Zero-Day Attack Detection system) are sent to a server responsible for global IDS/IPS Hotfix update as depicted in Figure 3.4.

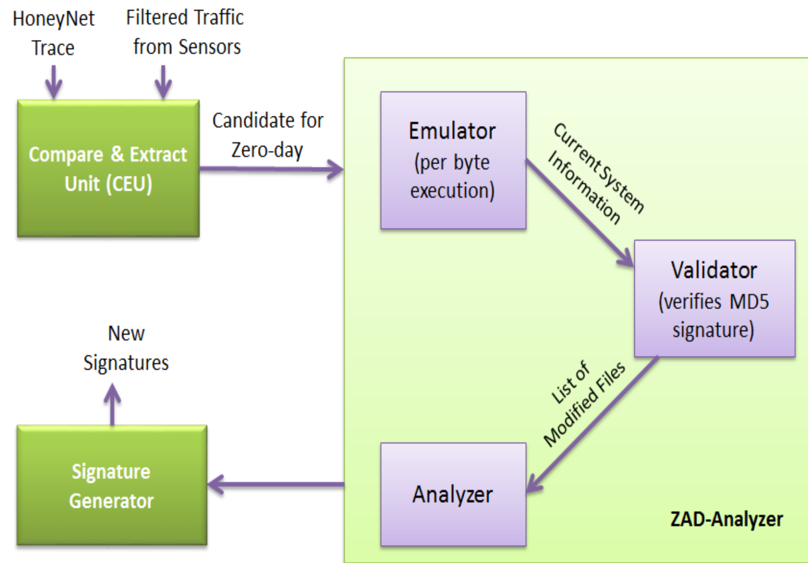


Figure 3.3: Evaluating Unknown Traffic Trace

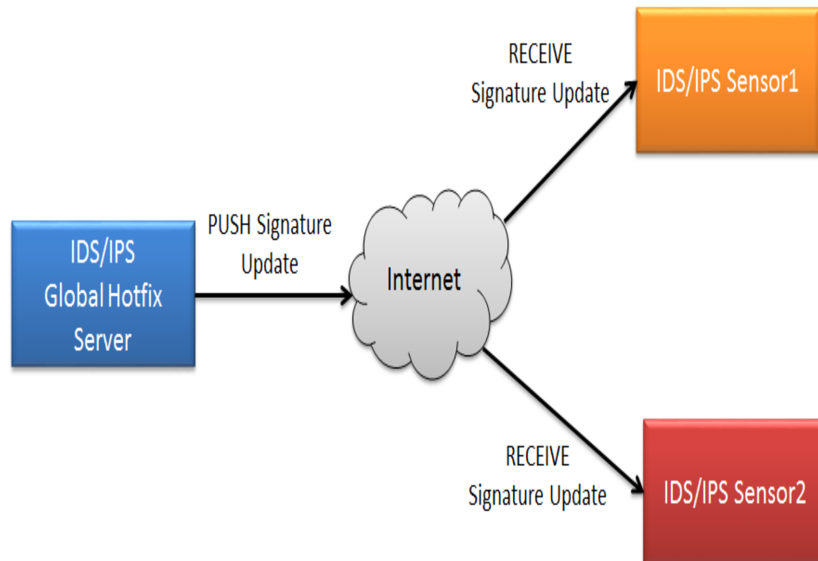


Figure 3.4: Hotfix Update

A two-level automated technique for detecting zero-day attacks is proposed as described above. It detects obfuscated zero-day attacks with two-level evaluation. At first level the system detects unknown by using Honeynet as an anomaly detector and at second level the system confirms malicious by analyzing behavior of unknown attack and at last generates new signatures automatically to update other IDS/IPS sensors via global hotfix. This two-level evaluation tends to decrease the false positives but faces various issues. The anomaly detector (honeynet) is not able to distinguish between normal traffic and unknown attack traffic at the first stage and requires the whole process of evaluation by ZAE to confirm that the captured unknown traffic is malicious. The evaluation module confirms malicious nature of the trace by analyzing just one system attribute i.e. msd5-checksum. Thus this method does not provide detailed insight about the behavior of a zero-day malware. It does not address problems regarding anti-analyses techniques.

3.3 RADAR Architecture

3.3.1 Design Principles for RADAR

A novel technique called RADAR is proposed to efficiently detect zero-day attacks and to provide automated malware analysis. RADAR uses component based layered architecture to achieve parallelism and to add/remove components easily at any stage. The key design principles that were considered while designing RADAR are:

- The system should be able to differentiate between benign traffic and ma-

licious traffic.

- The system should be capable of detecting zero-day attacks with high sensitivity and specificity.
- The system should analyse the binary automatically with minimum manual intervention.
- The analysis components must update all the captured behavior in a centralized storage.
- The system must generate a comprehensive report on zero-day malware behavior.

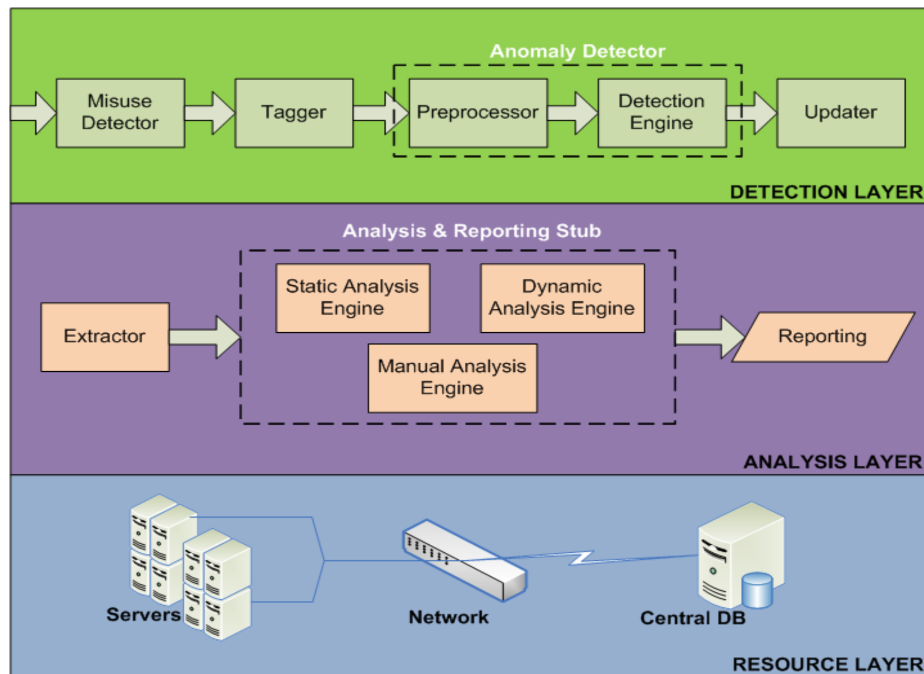


Figure 3.5: Layered Architecture of RADAR

Figure 3.5 provides an overall layered architecture of RADAR system. It has three layers: Detection Layer, Analysis Layer and Resource Layer. The detection

layer is responsible for detecting unknown attack. The analysis layer is required to analyze the behavior of captured binary. The resource layer provides the hardware resources like network, database and processing servers which helps in execution of components in above two layers. All these layers work in parallel to improve overall performance of the system. Detailed working of these layers is explained in the coming sections. Figure 3.6 shows process flow diagram of the RADAR system.

3.4 Detection Layer

The detection layer is the first layer of defense that detect unknown attacks or so called zero-day attacks. It constitutes of the following components:

3.4.1 Misuse Detector

A misuse detector models abnormal behavior. It has a well-defined set of malicious behaviors in terms of rules. Misuse detection systems are used to filter all known attacks as they are highly accurate in their decisions and have excellent throughput. An anomaly detector is not used in the first step because they cannot compete with misuse detectors when it comes to well-known attacks; therefore a misuse detector is deployed to filter incoming network traffic from known attacks. In RADAR, Snort has been used as a misuse detector. Snort [Roe99] is the most popular open source network intrusion prevention and detection system (IDS/IPS). It avoids known intrusions through signature matching. Snort analyzes the packets that arrive to the network interface, match their characteristics with those contained in the rules stored in its rule base. If a specific packet

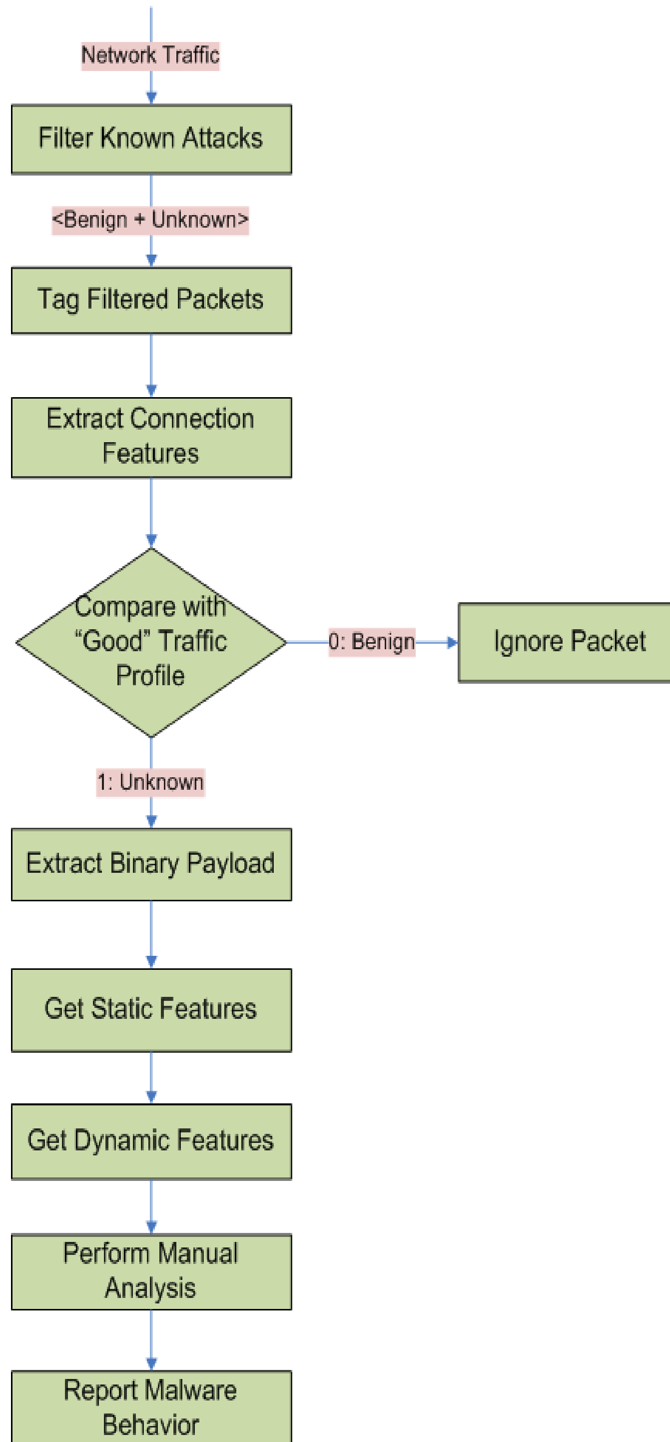


Figure 3.6: RADAR: Process Flow Diagram

matches the premises of any rule, this rule is executed and a specific action is generated to give notification of the fact. Here the snort drops all the known attack packets and passes filtered traffic for further processing. To drop known attacks snort is used in inline mode. All the "alert" rule actions of well-known attacks were changed to "drop" by a script.

3.4.2 Tagger

After filtering known attacks, all the remaining traffic is tainted and passed through an anomaly detector. As the anomaly detectors have either score or label based output techniques, therefore tainting is done to track the network packets which deviate from the normal profile. This way the unknown network packets are identified for further analysis. Traffic tainting is done by a component called "Tagger". It monitors all filtered traffic, tags it and sends it to the preprocessor. The tagger creates a new identifier based on 16-bit hash of a packet. The tag value and label for the filtered packet is stored in a table $< Tag, Label >$. The value for *tag* is calculated for the 6-tuple (*arrival_time, src_ip, dst_ip, src_port, dst_port, and protocol*) by using a fast and effective method of "XOR and shift". The *label* field is updated later with the result of detection engine like, $< 1 : anomaly, 0 : benign >$.

3.4.3 Preprocessor

A misuse detector presents an important flaw of not able to respond against unknown attacks. To overcome such shortcoming, anomaly detector is used in the next step. Anomaly detector model legitimate network traffic in order to

obtain potential deviations from the normal profile. Each deviation that is found significant enough is considered for further analysis. Also establishing a "good" network profile makes it easier to spot previously unknown bad behavior.

RADAR modifies Snort-AD as an anomaly detector. Snort-AD is modified to increase the probability of detecting unknown anomalies. Existing Snort-AD mainly extracts network traffic parameters based upon incoming/outgoing packets and download/upload speed [SABS12]. It mainly focuses on detecting denial of service (DoS) and spoofing attacks. It only detects what it is trained to do and thus is unable to detect attack with changed behavior. Hence, Snort-AD is ineffective to detect zero-day attacks efficiently.

The modified SnortAD consists of a preprocessor and a detection engine. The preprocessor extracts basic network traffic features including "same host" and "same service" features along with individual connection attributes. This facilitates to capture clear and detailed view of the suspicious connection records. Moreover, the detection algorithm is replaced with a semi-supervised machine learning algorithm, 1-class SVM, to get rid of manual threshold adjustment and over fitting problem. Remember that known attacks like DOS, U2R, R2L and probe are already filtered by misuse detector and hence, the system as a whole is able to defend against variety of attacks. The modified SnortAd is more efficient to detect zero-day attacks as (i) it captures more relevant features of suspicious connection records that may be part of zero-day attack (ii) uses machine learning over "good" traffic to find unknown anomalies, and (iii) the model self-learns from new or modified normal data.

The modified preprocessor receives the filtered tagged packets from the tagger and processes them. It extracts features, identifies most relevant parts of net-

Table 3.1: Extracted Features

FEATURES	DESCRIPTION
Duration	the length (number of seconds) of the connection
Protocol_type	type of the protocol, e.g. tcp, udp, etc.
Service	the connection's service type, e.g., http, telnet, etc
Source bytes	the number of data bytes sent by the source IP address
Destination bytes	the number of data bytes sent by the destination IP address
Count	the number of connections whose source IP address and destination IP address are the same to those of the current connection in the past two seconds.
Same_srv_rate	percentage of connections to the same service in Count feature
Serror_rate	percentage of connections that have SYN errors in Count feature
Srv_serror_rate	percentage of connections that have SYN errors in Srv_count(the number of connections whose service type is the same to that of the current connection in the past two seconds) feature
Dst_host_count	among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose source IP address is also the same to that of the current connection.
Dst_host_srv_count	among the past 100 connections whose destination IP address is the same to that of the current connection, the number of connections whose service type is also the same to that of the current connection
Dst_host_same_src_port_rate	percentage of connections whose source port is the same to that of the current connection in Dst_host_count feature
Dst_host_serror_rate	percentage of connections that have SYN errors in Dst_host _count feature
Dst_host_srv_serror_rate	percentage of connections that SYN errors in Dst_host_ srv_count feature
Flag	the state of the connection at the time the summary was written (which is usually when the connection terminated)
Pkt_count_legitimate_ports	among the past 100 connections whose destination port is same to the port in the legitimate ports list
Pkt_count_unexpected_ports	among the past 100 connections whose destination port is same to the port in unexpected ports list, especially on ports known to be backdoor ports

work traffic and normalizes data before sending to the detection engine. For constructing a candidate set of traffic features, total 17 significant and essential features were extracted and stored in a log file. These features identify relevant network traffic characteristics that may be part of a zero-day attack. The extracted features are listed in Table 3.1:

3.4.4 Detection Engine

The detection engine receives parsed packets from preprocessor and compares them with the existing "good" traffic profile. For collecting good traffic, a subnet of safe machines in the network have been identified which does not generate or generates less malicious content. This subnet has network admin's system, network analysts's system and other trusted faculty's and researcher's system. These systems are hardened and all possible security mechanisms are applied. These systems have defined security privileges and policies and does not participate in any malicious activity. Before granting access to network a routinely hardened posture checking is done to know possible risks, vulnerability and loopholes in these trusted system. The required security measures are applied if needed and network access in granted. Also a trust value has been assigned to these machines based upon the past posture checking reports and experience. This trust value ranges from 1 to 10, with 1 as a compromised machine generating malicious traffic and 10 as fully hardened with no security loopholes. The network admin's computer system has a 9 trust value and the analyst's computer system has trust value of 8.

All the traffic generated by this subnet is stored in a central database as "known good" traffic. An approximate of 200 GB raw network traffic is collected over a week from this trusted subnet in Thapar University (TU). This data is then used to train the machine learning algorithm implemented in the detection engine. The preprocessor extracts similar 17 statistical features from the trusted traffic to construct a good traffic profile. The detection engine then applies machine learning on two types of data, i.e., known good traffic (from trusted subnet) and

filtered traffic (from Snort), to detect zero-day attack. Figure 3.7 represents the creation of good profile.

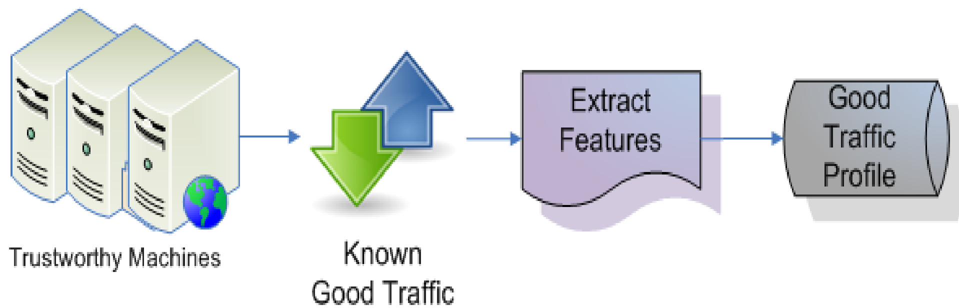


Figure 3.7: Creation of Good Profile

The detection engine employs Support Vector Machine (SVM) [Vap99] [Vap98] to detect unknown observations. SVM is one of the most developed machine learning technique. It has the ability to detect novelty and to provide protection against zero-day attacks. SVM learns incrementally i.e., for a given SVM model, it is possible to incorporate new training data without recalculating on all previous data. Therefore, SVM is preferred over other machine learning methods for its high accuracy, robustness and its capability to work well with different types of data.

For creating a SVM model it is easy to gather training data for normal situations but it is difficult, or just impossible to collect all possible abnormal scenarios for a zero-day attack. To deal with such problem in detection of zero-day attacks, 1-class SVM [SPST+01a] is the best option. 1-class SVM has many advantages over conventional statistical anomaly detection algorithms. 1-class SVM can easily overcome limitations like, over-fitting (1-class SVM is not complectly dependent on training data), requirement of pure normal data (it can be a mixture of normal data with some intrusion data) and reliance on threshold (for 1-class SVM the

threshold is not critical; as long as it is in some reasonable range, the result won't change much). The "good" traffic data is provided to 1-class SVM to create a representational model of the data. Now, if RADAR detection engine encounters new traffic data too different, from the model, it will be labeled as zero-day/unknown.

Given the unlabeled l data points, $\{x_1, ...x_2\}$ where $x_i \in R^n$; 1-class SVM maps the data points x_i into the feature space by using some non-linear mapping $\Phi(x_i)$, and finds a hypersphere which contains most of the data points in the feature space. Figure 3.8 shows the formal illustration of the hypersphere model. It is formulated with the center c and the radius $R > 0$ in the feature space, of which the volume R^2 is minimized. The data points that lies outside the hypersphere are regarded as anomalies.

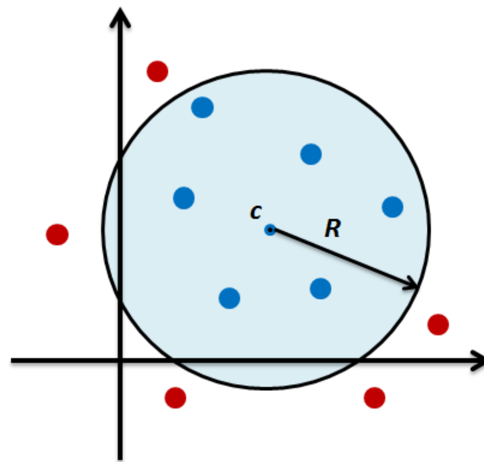


Figure 3.8: Hypersphere Model

Mathematically the problem of fitting a hypersphere around the data is formalized as:

$$\begin{aligned} \min_{R\in\Re,\xi\in\Re^l,c\in\mathcal{F}} \quad & R^2 + \frac{1}{vl}\sum_{i=1}^{l}\xi_i \\ \textbf{subject to}: & \|\Phi(x_i) - c\| \leq R^2 + \xi_i, \\ & \xi_i \geq 0, i = 1, ..., l. \end{aligned} \tag{3.1}$$

To prevent RADAR detection engine from over-fitting with noisy data, the non-negative slack variables ξ_i are introduced to allow some data points to lie on the “wrong”side of the hypersphere. Also, the parameter $v \in [0, 1]$ determines the trade off between the radius of the hypersphere and the number of the data points that belong to the hypersphere. When v is small, more data is put into the hypersphere. When v is larger, its size decreases. Since the center c belongs to the possibly high-dimensional feature space, it is difficult to solve the fundamental Equation 3.1 directly. Instead, it is possible to solve the fundamental problem by its dual form with kernel functions, $k(x, y)$ as in Equation 3.2:

$$\begin{aligned} \max_{\alpha\in\Re^l} \sum_{i,j=1}^{l}\alpha_i\alpha_j k(x_i, x_j) - \sum_{i=1}^{l}\alpha_i k(x_i, x_j) \\ \textbf{subject to}: \sum_{i=1}^{l}\alpha_i = 1, \\ 0 \leq \alpha_i \leq \frac{1}{vl}, i = 1, ..., l. \end{aligned} \tag{3.2}$$

After finding a hypersphere data can be classified as either normal or attack. In this classification, the following decision function, whether point x in the testing data is normal(i.e., inside of the hypersphere), is used as given in Equation 3.3:

$$f(x) = sign\bigg(R^2 - \sum_{i,j=1}^{l} \alpha_i \alpha_j k(x_i, x_j) \\ +2\sum_i \alpha_i k(x_i, x) - k(x, x)\bigg). \quad (3.3)$$

The points with $f(x) = -1$ are considered to be anomalies because this means that they exist outside of the hypersphere. Otherwise they are considered to be normal, because they are members of the hypersphere. LibSVM SDK has been used to implement 1-class SVM.

3.4.5 Updater

This component serves as the output-plugin for anomaly detector and pushes the result to a central database. Let P_i be one filtered packet. The tagger gives it a tag, $< T_i >$. After feature extraction and comparison with "good" profile, the detection engine output results as $< T_i, 1 >$ meaning, that the packet with tag T_i is anomalous. The same information is updated in "Tagger" table and the Label entry changes to 1. The corresponding packet P_i with tag $< T_i >$ is saved on the file system with same name as the tag value. This packet is then processed by the second layer for further investigation.

3.5 Analysis Layer

This layer is responsible for analyzing and reporting malicious behavior of zero-day malware. This layer comprises of following components:

3.5.1 Extractor

The extractor on receiving suspicious packet P_i, parses it and extracts the malicious *binary_i* for further analysis and reporting. The initial header of the packet is Ethernet. The structure of each successive headers is known to identify the location of fields containing the current header length and the next header type. Ethernet header contains information about next header type i.e. IP header and length of current Ethernet frame is 14. These two values are used to locate the position of IP header in the packet. This process is applied to subsequent headers and is repeated till all headers are processed. In the end, the application protocol stream is processed to strip the last header and to extract the data. This data is then saved as a binary file and is sent to next component for detailed examination.

3.5.2 Analysis and Reporting Stub

There is really no automated way to understand completely the malicious intents of a malware. There is no single best approach for malware analysis so it demands to combine static, dynamic and manual malware analysis. This analysis and reporting stub is a collection of malware analysis functionality in a component-based architecture, where any analysis function can be replaced in the future. Different analysis functions have been integrated into the system that

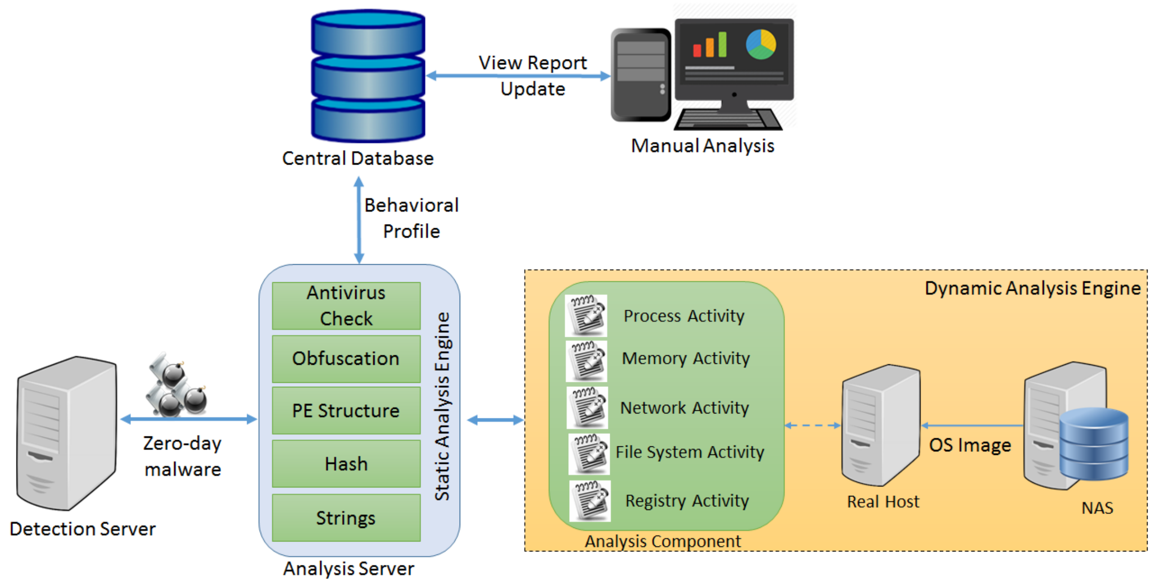


Figure 3.9: Analysis & Reporting Stub

work together automatically to provide detailed result about a malware's behavior. The ultimate goal of the analysis and reporting stub is to gain a quick and detailed understanding of the malicious activity performed by zero-day binary while minimizing the time frame between the discovery of zero-day vulnerability and generation of a security solution. Figure 3.9 depicts the basic architecture of the analysis and reporting stub. The captured zero-day malware/ binary is fed to this unit for behavioral analysis. The result of integrated procedures is stored in a central database from where the reporting module generates malware analysis report. The report generated is comprehensive and easy to understand by an analyst.

3.5.3 Static Analysis Engine (SAE)

SAE comprises of various static analysis functions, running parallel in the background. To implement these functions, a python script named "static.py" is written to inspect binaries for static properties. It checks captured binary for structural attributes. It reports about antivirus scanning, packer signature, PE structure, hashes and strings. These structural details are then stored in the database by the same script. Following are the features provided by SAE:

Antivirus Scanning: SAE provides the capability to scan the binaries against large number of antivirus programs using different signatures and heuristics. This is achieved by using the VirusTotal SDK [Vir14]. VirusToatl uses more than 50 different antivirus products and helps in capturing analysis details like malware name, file size, hash, behavioral information (if available) and the detection rate (total number of antivirus products that marked the file as malicious divided by total number of antivirus products). SAE- Antivirus Scanner (Figure 3.10) scans captured binary to check whether same binary has been earlier identified by other antivirus program or not.

SAE- Antivirus Scanner is designed and integrated with retry mechanism to retry scanning incase there is some form of delay from VirusTotal in processing the request. The integration requires two queues, two monitors and a VirusTotal database accessed by VirusTotal Public APIs v2.0. These APIs uses HTTP POST request with JSON object response format for sending and retrieving scan reports respectively. The first queue $Q1$, loads up to 4 binaries as the API requests are

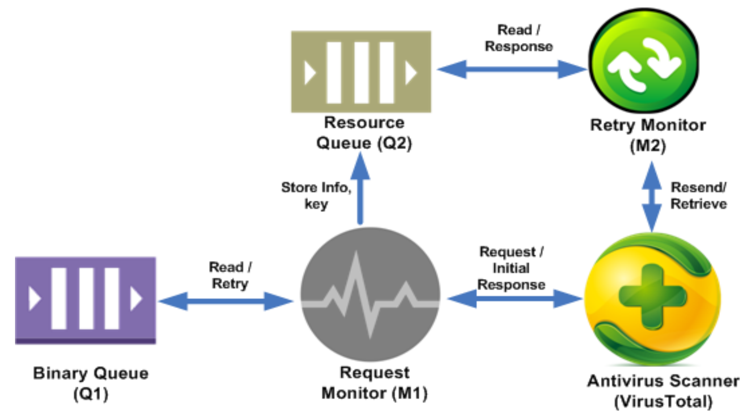


Figure 3.10: SAE- Antivirus Scanner

limited to 4 requests in 1 minute time frame. The monitor $M1$, reads the file, to send from $Q1$ and formats the send message into a HTTP POST request. The request message contains host details, file information and apikey (for accessing public APIs). For each sent HTTP POST request the JSON response object contains a parameter known as $response_code$, which determines the response result.

If the $response_code$ is 0, it means the binary is not present in VirusTotal's dataset. In that case, the binary is fed to other components in SAE to perform additional analysis. If $response_code$ is 1, meaning the file searched is present in VirusTotal database and can be retrieved then, SAE fetches the existing analysis report to upload in the central database and stops further analysis. SAE fetches various scan report attributes from VirusTotal like $< resource_id, scan_id, perma_-link, md5, sha1, sha256, scan_date, positives, total >$ and stores in VT_basic table and $< antivirus, result, update >$ in $VT_verbose$ table. This binary is then marked as "known" in the database. The $response_code$ is -2 just in case the request is still queued for analysis. In such a scenario the requested item data like resource information and apikey is pushed to queue $Q2$ for later processing. The

next file from $Q1$ is fetched and processed by $M1$. The monitor $M2$, is responsible for sending the pending request again from $Q2$ and retrieving its response. The *null response_code* indicates communication failure with the VirusTotal server.

```
Sending file:
host = "www.virustotal.com"
selector = "https://www.virustotal.com/vtapi/v2/file/scan"
fields = [("apikey", "xyz")]
file_to_send = open("pe_file", "rb").read()
files = [("file", "pe_file", file_to_send)]
json = postfile.post_multipart(host, selector, fields, files)

Retrieving scan report:
url = "https://www.virustotal.com/vtapi/v2/file/report"
parameters = "resource": "md5 of pe_file", "apikey": "xyz"
data = urllib.urlencode(parameters)
req = urllib2.Request(url, data)
response = urllib2.urlopen(req)
json = response.read()
```

Packer Detection: Malwares often use obfuscation techniques to evade detection systems. One such popular obfuscation technique is packing. To detect the type of packer employed PEid [Ald13] has been utilized. To integrate this feature, *static.py* accesses PEid database, *UserDB.TXT*, which contains 1832 packer signatures. Once the database is loaded, the binary is read for matching packer signature. An option is also provided to add more signatures later in the

database file or to load an alternative database for aggregating more signatures. The database file has packer name as the section name and two keys: the *signature* key containing the byte pattern and the *ep_only* key. The *ep_only* property can be true or false. This property specifies if the signature has to be found at the PE file's entry point (true) or can be found anywhere (false). The binary is scanned to find the matching packer signature which is then updated in the central database.

```
pe = pefile.PE(pe_file)
signatures = peutils.SignatureDatabase("peidDB.txt")
matches = signatures.match_all(pe, ep_only = True)
update "matches" in database
```

PE Header Information: Any binary executable file includes a header to describe its structure like, the base address of code section, data section, list of functions imported, exported, etc. To execute the file, the Operating System simply reads the header first and loads the binary data from the file to code/data segments of the address space for the corresponding process. During dynamic linking the OS relies on file's import table to determine the entry addresses of the system functions. Most binary executable files on Windows follows the following structure: DOS Header (64 bytes), PE Header, sections (code and data). DOS Header starts with magic number 4D 5A 50 00, and the last 4 bytes is the location of PE header in the binary file.

The PE header contains significantly more information and is more interesting. At run time, Windows loader loads the PE header into a process's address space. PE header consists of three parts: (1) a 4-byte magic code, (2) a 20-byte file

header and its data type is IMAGE_FILE_HEADER, and (3) a 224-byte optional header (type: IMAGE_OPTIONAL_HEADER32). The optional header itself has two parts: the first 96 bytes contain information such as major operating systems, entry point, etc. The second part is a data directory of 128 bytes. It consists of 16 entries, and each entry has 8 bytes (address, size). The PE header contains useful information for the malware analyst and the important fields that can be obtained from a PE header are:

- Imports: Functions from other libraries that are used by the malware.
- Exports: Functions in the malware that are meant to be called by other programs or libraries.
- Time Date Stamp: Time when the program was compiled.
- Sections: Names of sections in the file and their sizes on disk and in memory.
- Subsystem: Indicates whether the program is a command-line or GUI application.
- Resources: Icons, menus, and other information included in the file.

To extract this valuable information the *static.py* script uses a Python PE parsing module, *pefile* 1.2.10-139, to inspect PE header, to retrieve all the sections, imports, exports, resources, their information and data. The output is get in the desired format and stored in the central database.

Attributes: Image Base: hex(pe.OPTIONAL_HEADER.ImageBase)

Entry Point Address : hex(pe.OPTIONAL_HEADER.AddressOfEntryPoint)

CPU type: pefile.MACHINE_TYPE[machine]

dll = pe.FILE_HEADER.IMAGE_FILE_DLL

Subsystem: pefile.SUBSYSTEM_TYPE[pe.OPTIONAL_HEADER.Subsystem]

Compile Time:

datetime.datetime.fromtimestamp(pe.FILE_HEADER.TimeDateStamp)

Number of RVA & Sizes: pe.OPTIONAL_HEADER.NumberOfRvaAndSizes

Sections:

Number of Sections: pe.FILE_HEADER.NumberOfSections

for section in pe.sections:

section.Name, hex(section.VirtualAddress),

hex(section.Misc_VirtualSize) section.SizeOfRawData, E(section.data)

Resources:

For res in pe.DIRECTORY_ENTRY_RESOURCE.entries

update "res.name, res.data.struct.OffsetToData, res.data.struct.Size, res.filetype, res.data.lang" in database

Imports:

for entry in pe.DIRECTORY_ENTRY_IMPORT:

update "entry.dll" in database

for imp in entry.imports:

update "hex(imp.address), imp.name" in database

Exports: for exp in pe.DIRECTORY_ENTRY_EXPORT.symbols:

update hex(pe.OPTIONAL_HEADER.ImageBase + exp.address), exp.name, exp.ordinal

Hashing: The main purpose of using this feature is to generate various hashes for the binary. These hashes provides a unique fingerprint for the malware. The *static.py* file generates various hashes like MD5, SHA-1 and SHA-256 for the captured binary. The script also returns a ten digit representation of the size of file processed. Therefore, along with hash value the file size is extracted as well and saved in database.

```
fileStr= open("pe_file","rb").read()
hashlib.md5(fileStr).hexdigest()
hashlib.sha1(fileStr).hexdigest()
hashlib.sha256(fileStr).hexdigest()
update in database
```

String Extraction: A malware program contains strings if it has to print a message, connect to a URL, or has to copy a file to a specific location. Searching these strings can help to get hints about the program functionality. Like, the legitimate programs always include many embedded strings but an obfuscated or packed malicious program contains very few strings. So, if few embedded strings are returned (either make sense or not) then the tested binary is likely to be malicious. The *static.py* file examines ASCII and Unicode strings in binary data. All the printable strings from the binary file are saved in database and reported.

```
fileStr=open("pe_file", "rb").read()
if fileStr in string.printable:
result += fileStr
update database
```

SAE is completely modular and this makes it flexible and extensible. SAE has a main python script *static.py* that starts each functionality and extracts its output to save in database. With the preliminary static analysis it is possible to extract valuable information that will determine the profile of the malware.

3.5.4 Dynamic Analysis Engine (DAE)

After reporting static properties the binary is passed to DAE for dynamic analysis since static analysis is not foolproof. DAE focuses on behavioral analysis which helps to understand the nature and the purpose of the malicious binary and reveals which files are read or accessed and which operations has been carried out. DAE performs behavioral analysis by executing the binary in a real environment with Kernel-based monitoring. Real physical hardware is used in analysis to allow binary run its complete instructions. The Operation System of real host is replaced with new Operating System image after each execution of binary through network-attacked storage (NAS). It is also possible that malware authors design their malware to check the execution environment. If executing environment is detected, the malware can either stop running or raise an exception or loop for a long time, thus evading its analysis. As DAE executes the binary on real hardware, it cannot be fooled by malware's anti-analysis techniques. Moreover, full visibility allows full behavior detection and does not require hardening host, vig-

ilant patching and use of anti-malware defenses.

Kernel-based Monitoring:

Traditional sandboxes uses API hooking, native API hooking, system call hooking techniques on 32-bit Windows Operating System to monitor system activities. They hook API functions or system calls to route them to their analysis function and sometimes manipulates the call by changing parameters. User-mode API hooking technique requires rewriting of the target function and this rewriting can be detected by a malware, as integrity of APIs can be checked. On the other hand, Kernel-API hooking is not appropriate for implementing a reliable software also, it is not Microsoft approved approach. It is not recommended to hook vital kernel parts as kernel functions may change due to kernel updates or at time of installing new service packs. Hooking kernel functions is critical to system stability and security, besides it is difficult to hook kernel functions in modern Operating Systems. Windows 64-bit version includes additional protection mechanism like Kernel Patch Protection (KPP) that is not easy to bypass [Rie15].

To extract behavioral information, RADAR uses Kernel-based monitoring. This technique is implemented for Windows-based system by using a minifilter driver intercepting Kernel functions in its PreOperation Callback. The minifilter driver is reliable and compatible with all modern versions of Microsoft Windows (2000, XP, Vista, Server, 7 and 8), including 64 bit versions. The minifilter driver is also resistant to malware's anti-analysis technique (like anti-debug or anti-reversing techniques) as it works at lower-level without deteriorating the system efficiency. Figure 3.11 shows RADAR's Kernel-based analysis component. It has following

three main parts:

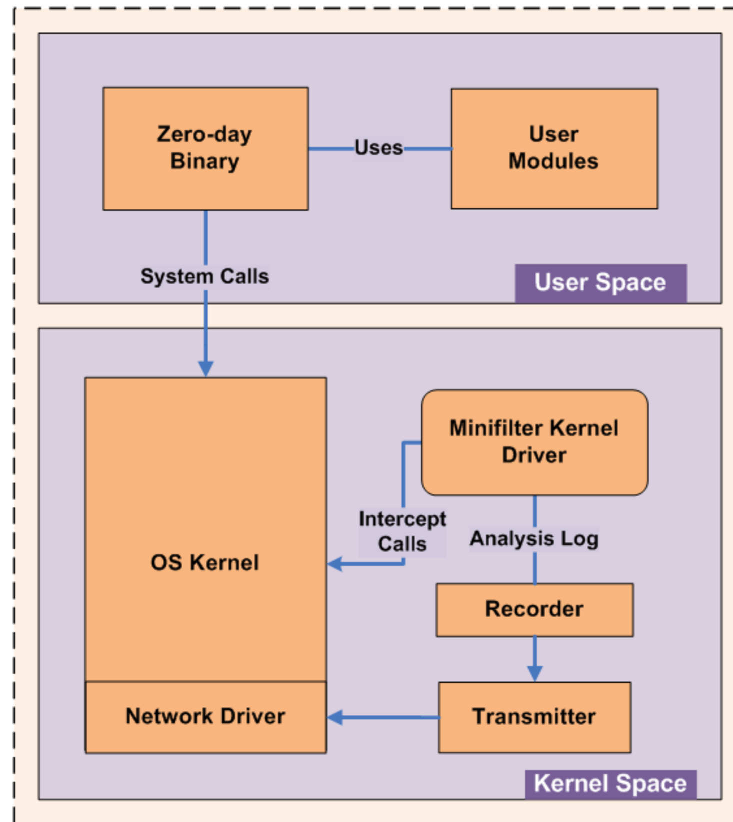


Figure 3.11: Kernel-based Analysis Component

- **Minifilter Driver**: The minifilter driver is used to dynamically monitor the activities of the malicious binary. It intercepts the request before it reaches its intended target and tracks the functionality provided by the original target of the request. It does so by intercepting a Kernel function in its PreOperation-callback and monitor its parameter block. The minifilter driver tracks various system objects like file system, registry, processes, network and the memory. The minifilter driver performs more fine grain analysis as it has complete control over the system. It is a stand-alone driver and just needs a configuration file. Also it does not need a

service and fires up immediately after the kernel initializes. As everything is done in driver itself there is no communication into real-mode thus, the driver remains invisible to the malicious binary that is currently under analysis. The minifilter driver named "MiniSnoop" is created using Windows programming with Windows Driver Kit (WDK). The analysis component is developed using C programming with Visual C++. The real host has minifilter driver "MiniSnoop" installed to monitor system objects.

- **Recorder**: The recorder works along with the driver to write the entire analysis information gathered by the driver into a simple log file.
- **Transmitter**: This component transmits the captured logs to the analysis server. These logs are then parsed and updated into the database in their respective tables.

Minifilter Callback Mechanism:

The minifilter utilize a callback mechanism. This callback mechanism specifies what kernel functions are of interest in filtering. At first, in *DriverEntry()* routine which is must-have routine of every driver, minifilter call to *FltRegisterFilter()* routine which takes a *FLT_REGISTRATION* structure as a parameter to define its own callback function for each type of operation it want to filter. At the time correspondent request is processed, minifilter's pre-operation callback is called. When the filter manager calls a minifilter driver's preoperation callback routine for a given operation, the minifilter driver temporarily controls that operation. It gives minifilter a chance to examine and record arguments of the request. The parameters of pre-operation callback routine include a pointer to *FL_CALLBACK_DATA* which hold information such as type of operation, file object, flags.

Algorithm 1 depicts implementation of minifilter driver intercepting functions in pre-operation callback:

Algorithm 1 MiniSnoop Kernel Filtering

```
1: procedure initializeDriverRoutine()
2:     Initialize Driver by Invoking...
3:     NTSTATUS;
4:     *PDRIVER_INITIALIZE(
5:     IN   PDRIVER_OBJECT   DriverObject,
6:     IN   PUNICODE_STRING   RegistryPath);
7: end procedure
8: procedure minisnoopDriverRegistration()
9:     Register Driver by Invoking...
10:    status = FltRegisterFilter(
11:    DriverObject,
12:    &FilterRegistration,
13:    &MiniSnoopData.FilterHandle);
14: end procedure
15: procedure registerPreOperation()
16:    for all (CallbackRoutines) do // as mentioned in below tables.
17:        const   FLT_OPERATION_REGISTRATION   Callbacks();
18:        {IRP_MJ_CREATE, 0, PreCreate, NULL},
19:        {IRP_MJ_WRITE, 0, PreWrite, NULL},
20:        Similarly, other monitoring routines can be registered.
21:    end for
22: end procedure
23: procedure initiateFilteringProcess()
24:    status   = FltStartFiltering(MiniSnoopData.FilterHandle);
25:    if (!NT_SUCCESS(status)) then
26:        FltUnregisterFilter(MiniSnoopData.FilterHandle);
27:    end if
28:    if (processName==“binary.exe”) then
29: return FLT_PREOP_SUCESS_WITH_CALLBACK;
30:    else
31: return FLT_PREOP_SUCESS_NO_CALLBACK;
32:    end if
33: end procedure
```

```
34: procedure processPreOperationCallbacks()
35:     typedef FLT_PREOP_CALLBACK_STATUS
36:     *PFLT_PRE_OPERATION_CALLBACK(
37:     IN OUT PFLT_CALLBACK_DATA Data,
38:     IN PCFLT_RELATED_OBJECTS FltObjects,
39:     OUT PVOID *CompletionContext);
40: end procedure
41: procedure record&transmitCapturedLogs()
42:     Parse log file captured by Minisnoop.
43:     Transmit to analysis server.
44: end procedure
45: procedure networkFilterEgine()
46:     Get incoming values list FWPS_INCOMING_VALUES
47:     Fetch general information like ethernetHeaderSize, EndPointHeader
48:     Fetch additional details based upon different layers.
49:     for (IncomingValues -> LayerID) do
50:
51:         if (FWPS_LAYER_INBOUND/OUTBOUND_IPPACKET_V{4/6}) then
52:             Fetch ipHeaderSize, sourceIPadd, destIPadd
53:             Capture bytesTransferred;
54:         end if
55:         if (FWPS_LAYER_INBOUND/OUTBOUND_TRANSPORT_V{4/6}) then
56:             Fetch tcpHeaderSize, protocol, sourcePortID, destPortID
57:             Capture bytesTransferred;
58:         end if
59:         Fetch more information from other layers as required.
60:     end for
61: end procedure
```

The minifilter driver in DAE performs monitoring using callback functions registered in kernel managers (such as I/O, Filter, Registry, Process, Network and Memory) which are present in a Windows Kernel. Figure 3.12 consists of several kernel managers and minifilter driver's monitors used to filter out access to system resources.

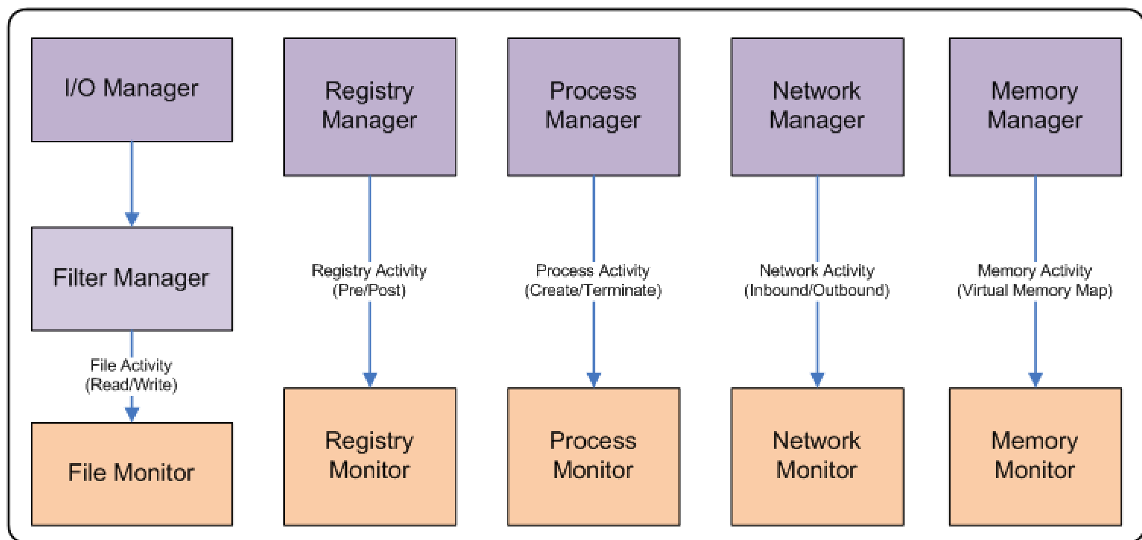


Figure 3.12: Minifilter Callback Mechanism

File Monitor: File monitor checks for file system changes. The file system changes are operations performed on files or directories that include: rename, set attributes (e.g. timestamps, file attributes, security permissions, etc), delete, create, write to an existing file or directory, read from file system. File monitor, registers a pre-operation callback on every operation made on file system. Whenever the minifilter need to handle an I/O operation, the filter manager makes a call to the appropriate callback routine. All I/O requests passed to the drivers (filter manager and minifilter) uses a standard IRP (I/O Request Packet) structure. Table 3.2 represents IRP major function codes (IRP_MJ_XXX) registered

for monitoring file manipulation.

Table 3.2: File Manipulation IRP Function Codes

IRP Major Function Codes	**Operation**
IRP_MJ_CLEANUP	Close the file object handle
IRP_MJ_CLOSE	Indicates that the handle of the file object has been closed and released
IRP_MJ_CREATE	Open a handle to a file object or device object. This request is sent when a driver calls CreateFile() routine
IRP_MJ_FILE_SYSTEM_CONTROL	Sent when I/O Manager or kernel-mode driver want, for example, mount a volume or verify a volume
IRP_MJ_QUERY_INFORMATION	Sent when a user-mode application call GetFileInformationHandle() routine to retrieve information such as : access mask, file name, file attribute
IRP_MJ_READ	Sent when a user-mode application has called ReadFile() routine
IRP_MJ_SET_INFORMATION	Sent when a user-mode application has called GetSecurityInfo() routine
IRP_MJ_DIRECTORY_CONTROL	Sent when a user-mode application has called ReadDirectoryChangeW() routine to request for notification of changes to the directory or to query for directory information
IRP_MJ_WRITE	Sent when a user-mode application has called WriteFile() routine
IRP_MJ_ACQUIRE_FOR_SECTION_SYNCHRONIZATION	Sent when a use-mode application want to map that file to memory for read, write or execute

Registry Monitor: Registry monitor receives information about registry operations by registering a callback routine, **RegistryCallback**, to Windows Kernel through **CmRegisterCallbackEx**. The callback gives information to get the full key name, the kind of access: create, rename, delete, etc . A **REG_NOTIFY_CLASS** value (Table 3.3)in **RegistryCallback** data structure contains

the information about each registry operation performed.

Table 3.3: Registry Operation Notifications

REG_NOTIFY_CLASS Value	Operation
RegNtDeleteKey	Notify when a registry key is being deleted.
RegNtSetValueKey	Notify about a new setting for a registry key's value entry.
RegNtDeleteValueKey	Notify when a registry key's value is being deleted.
RegNtRenameKey	Notify about the new name for a registry key whose name is about to be changed.
RegNtPostCreateKey	Notify the result of an attempt to create a registry key.
RegNtEnumerateKey	Notify about one subkey of a key whose subkeys are being enumerated.
RegNtEnumerateValueKey	Notify about one value entry of a key whose value entries are being enumerated.
RegNtQueryKey	Notify when the metadata for a key is about to be queried.
RegNtQueryValueKey	Notify about a registry key's value entry that is being queried.
RegNtQueryMultipleValueKey	Notify about multiple value entries that are being retrieved for a key.

Process Monitor: Process monitor registers various process and thread manager routines (Table 3.4) callback. On registering a **PsSetCreateProcessNotify-RoutineEx** callback, on each process creation, and before the main thread starts to run the process filter callback is notified and receives all the necessary information. It receives the process name, the file object, the PID, and so. Similarly, new threads are filtered with the **PsSetCreateThreadNotifyRoutineEX** callback. Each time a thread is created, the process monitor is notified with the TID and

the PID. Process monitor also get information about images that can be loaded into memory. An image is a PE file, either a EXE, a DLL or SYS file. This is done by registering **PsSetLoadImageNotifyRoutine**. That callback allows to be notified when the image is loaded into virtual memory, even it's never executed. It can then be detected when a process attempts to load a DLL, to load a driver, or to fire a new process. The callback gets information about the full image path, and the Image base address (for in-memory analysis).

Table 3.4: Process and Thread Manager Routines

Process/Thread Manager Routines	**Operation**
PsGetCurrentProcess PsGetCurrentProcessId	Returns a pointer to the process of the current thread. Identifies the current thread's process.
PsGetCurrentThread	Identifies the current thread.
PsGetCurrentThreadId	Identifies the current thread and returns the thread ID.
PsGetCurrentThreadTeb	Returns the Thread Environment Block (TEB) of the current thread.
PsSetCreateProcessNotifyRoutineEx	Registers or removes a callback routine that notifies the caller when a process is created or exits.
PsSetCreateThreadNotifyRoutineEx	Registers a driver-supplied callback that is subsequently notified when a new thread is created and when such a thread is deleted.
PsLookupProcessByProcessId	Accepts the process ID of a process and returns a referenced pointer to EPROCESS structure of the process.
PsSetLoadImageNotifyRoutine	Notifies whenever an image is loaded or mapped into memory.

Network Monitor: Network monitor is used to acquire maximum visibility on network traffic while the binary is running. To accomplish this network moni-

tor uses a new kernel-based filtering engine called Windows Filtering Platform (WFP). The WFP gives various APIs for packet filtering and for monitoring the TCP/IP traffic at any layer of the TCP/IP stack. A filtering driver is implemented to monitor network traffic. For achieving this, run-time filtering layer identifiers mentioned in Table 3.5 needs to be registered.

Table 3.5: Run-time Filtering Layer Identifiers

Run-time Filtering Layer Identifier	**Filtering Layer Description**
FWPS_LAYER_INBOUND_IP-PACKET_V4/6	Located in the receive path just after the IP header of a received packet has been parsed but before any IP header processing takes place.
FWPS_LAYER_OUTBOUND_IP-PACKET_V4/6	Located in the send path just before the sent packet is evaluated for fragmentation. All processing is complete and all extension headers are in place.
FWPS_LAYER_INBOUND_-TRANSPORT_V4/6	Located in the receive path just after a received packet's header has been parsed by at the transport layer, but before any transport layer processing takes place.
FWPS_LAYER_OUTBOUND_-TRANSPORT_V4/6	Located in the send path just after a sent packet has been passed to the network layer for processing but before any processing takes place.
FWPS_LAYER_STREAM_V4/6	Located in the stream data path. This layer allows for processing network data on a per stream basis. At the stream layer, the network data is bidirectional.
FWPS_LAYER_ALE_FLOW_ES-TABLISHED_V4/6	Allows for notification of when a TCP connection has been established, or when non-TCP traffic has been authorized.
FWPS_LAYER_STREAM_-PACKET_V4/6	Allows for inspection of network data on a per-TCP packet basis, including handshake and flow control exchanges. At this layer, the network data is bidirectional.

Memory Monitor: Memory monitor is used to enumerate memory map of the binary process by using **ZwQueryVirtualMemory** kernel driver routine present in NtosKrnl.lib. **ZwQueryVirtualMemory** is necessary for Kernel-level memory scanning. It uses **MEMORY_BASIC_INFORMATION** structure that contains information about a range of pages in the virtual address space of a particular process. **ZwQueryVirtualMemory** queries the pages of a process in the process address map from the base address upward until the entire range of pages has been scanned.

Summary of Operating System activities reported by MiniSnoop minifilter:

- **File System and Registry Changes**: The analysis component monitors real-time file system and registry activities. It logs added, deleted and modified system files and registry keys.
- **Process Activity**: This provides valuable insight into the currently running processes on the system. The analysis component logs new processes and threads created by the binary while execution, modules loaded by them (Loaded DLL libraries), Runtime DLLs, open Operating System resource handles and various process properties.
- **Network Activity**: The analysis component retrieves network information while binary is executing like network connections (both incoming and outgoing), network protocol statistics and traffic trace in pcap format. Information on active connections like protocol, local IP-address, foreign IP-address, process-id is collected. Network traffic for malicious communication attempts, such as DNS resolution requests, bot traffic, or downloads is

also recorded.

- **Memory Activity**: The analysis component logs memory statistics for current running process like memory written, memory allocated, memory protection changed and memory dumps for advance analysis.

3.5.5 Manual Analysis

Manual Analysis is an indispensable step in analyzing zero-day attacks as both static analysis and dynamic analysis have their own limitations. However, information collected from static analysis and dynamic analysis is useful for a human analyst while dissecting a zero-day binary. But still, if some part of analysis is left in SAE and DAE then that can be manually performed by an expert. For this the binary is run in a debugger, OllyDbg [Oll13] to animate instructions in a slow and controlled fashion. To do so, the Ctrl+F8 (ANIMATE OVER) is used to stepover until an address is arrived, which is the call to the main function. Next, the Crtl+F7 (ANIMATE INTO) is used to step-into the call to the main function. This is continued to step forward using F7 and F8 while noting the behavior of the sample. To evade anti-debugging techniques of malicious binary, anti-anti-debugger plugin [Plu10] for OllyDbg has been used. The aadp plugin avoids anti-debugging techniques like anti-debugging APIs or flags. In debugger a running program can be resumed in three different ways:

- **breakpoint:** stops a program whenever a particular point in the program is reached.
- **watchpoint:** stops a program whenever the value of a variable or expression changes.

- **catchpoint:** stops a program whenever a particular event occurs, analyze CPU environment (memory, registers).

3.5.6 Reporting

RADAR generates zero-day attack analysis report in HTML and PDF format. The analysis report is generated from analysis and reporting stub data. SAE and DAE stores all the structural and behavioral information of zero-day binary directly in a central database. While additional manual analysis findings are uploaded in form of notes in the database. Figure 3.13 shows reporting attributes captured by SAE, DAE and through manual analysis.

The analysis report describes the results of the malware analysis process and covers following areas:

- **Analysis Summary**: Key outcome from the analysis report regarding the malware's nature, origin, capabilities, relevant characteristics, indicators of compromise, followup actions and lessons learned.

- **Identification**: The file type, its size, hashes (such as MD5, SHA1, and SHA256), file name, anti-virus detection potential.

- **Characteristics**: Capability to infect files, self-preserve, spread, data leakage, communicate with the attacker, etc.

- **Dependencies**: System resources (like files, network, memory) related to the malware's functionality, initialization files, DLLs, executables, URLs, and scripts.

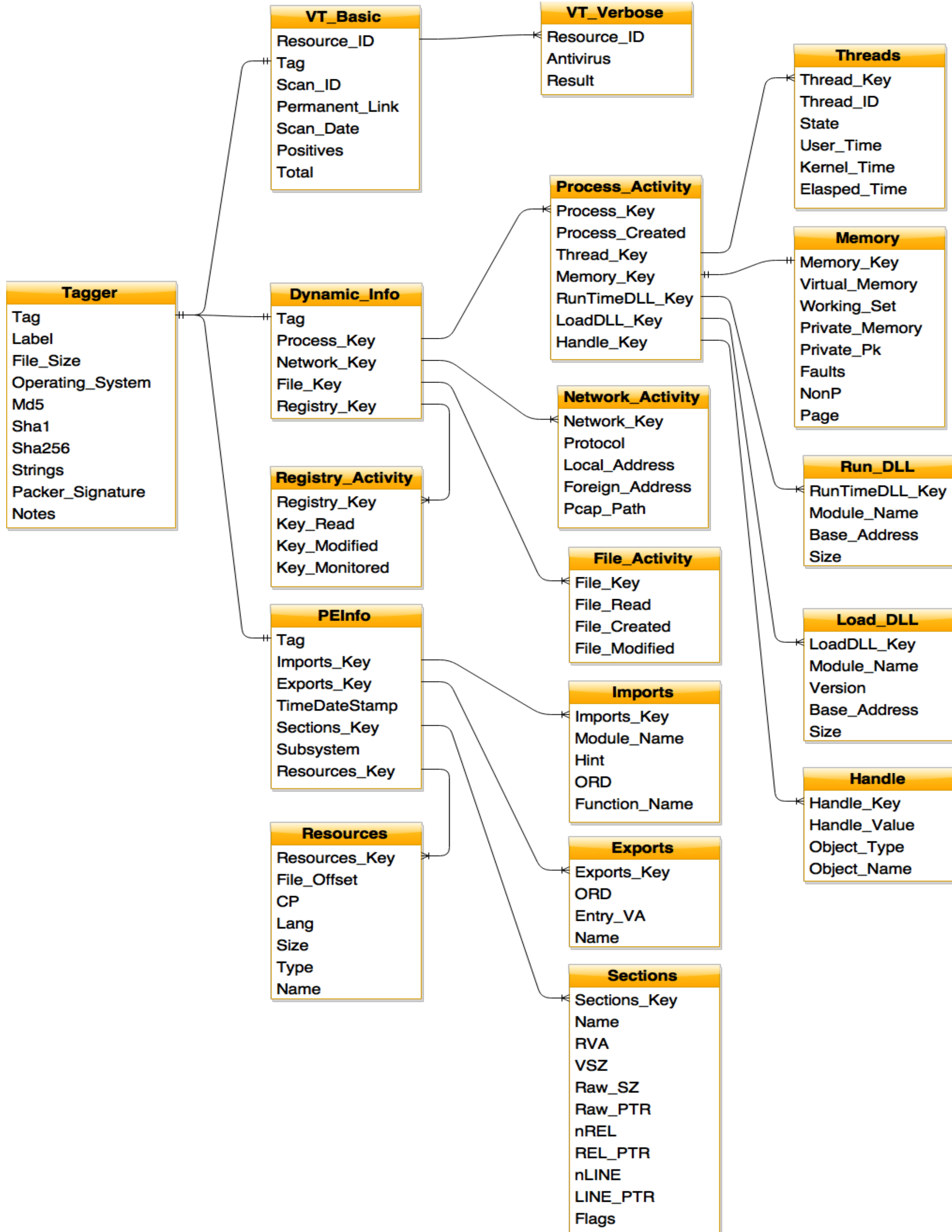


Figure 3.13: Reporting Attributes

- **Supporting Artifacts**: Logs, pcaps, dumps, string extracts, function listings, figures and other relevant system and network statistics.
- **Manual Analysis Findings**: Overview of the manually done static and dynamic code analysis observations.

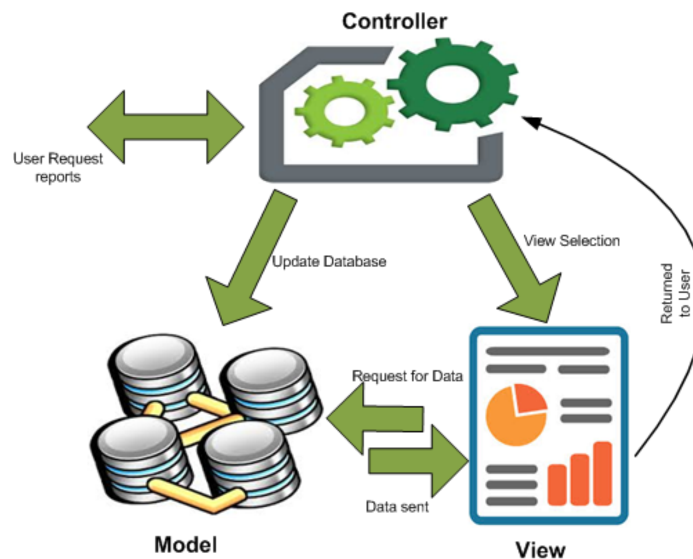


Figure 3.14: RADAR Reporting Framework

RADAR reporting framework is developed using the Struts2.0 MVC based architecture (Figure 3.14). MVC stands for model, view and controller. Model helps the reporting system to systematically query the database. All the data to be displayed in the reports is fetched through model layer. View presents the jQuery based interactive user interface that helps to fasten the manual analysis process. UI lists down the reports with both textual and graphical data. Graphs are dynamically generated when the screen loads. JFreeCharts are used to draw graphs. The role of controller here is to help the user to successfully navigate through the pages by keeping the smooth communication between the model and view.

The complete RADAR system process is listed in the form of algorithms. Algorithm 2 and Algorithm 3 explains the working of detection layer and analysis layer respectively.

Algorithm 2 RADAR Detection Layer

```
1: procedure RADAR_Detection()
2:     for Pcur in P do
3: Snort(Pcur);
4: TagPacket();
5: NormalizeData();
6: DetectionEngine();
7:     end for
8: end procedure
9: function FilteredPacket Snort(currentPacket)
10:    if (pkt_content.matches(snort_rules)) then
11:        drop(currentPacket);
12: return filtered_Pkt;
13:    end if
14: end function
15: function TagPacket()
16:    Calculate Tag = HASH(arrival_time, src_ip, dst_ip, src_port, dst_port,
   protocol);
17: UpdateDatabase(Tag);
18: return Tagged_Pkt;
19: end function
20: function String NormalizeData()
21:    for Tagged_Pkt from 1 to N do
22:        PARSED_PKT='Use snort preprocessor module for parsing packet'.
23:        Filter features defined in Table 3.1 from PARSED_PKT.
24:        Arrange the features in decimal format understandable by SVM.
25:    end for
26: return string;
27: end function
28: function DetectionEngine(String normalizeData, boolean isTrain)
29:    if (isTrain) then
30:        for Trust_Value > threshold do
31:            Capture good data from trustworthy systems.
32:            Extract features and update the SVM database to act as profiler.
33:        end for
```

```
34:     else
35:         boolean unknown = Compare the normalizedData with data as per
    SVM profile
36:         if unknown then
37:             resultSet = Form the INSERT statement.
38: UpdateDatabase();
39:         end if
40:     end if
41: end function
42: function UpdateDatabase(String resultSet)
43:     INSERT resultset.
44:     Save packet pkt with same Tag value at /usr/home/packets.
45: end function
```

Algorithm 3 RADAR Analysis Layer

```
 1: procedure RADAR_Analysis()
 2:     List binaries = extractBinary();
 3:     for binary in binaries do
 4: StaticAnalysis(binary);
 5: DynamicAnalysis(binary);
 6: ManualAnalysis(binary);
 7:     end for
 8: end procedure
 9: function List ExtractBinary()
10:     Read pkt from /usr/home/packets.
11:     hdr = initialType
12:     pos = 0
13:     while hdr ≠ DONE do
14:         len = GetHeaderLen(pkt, hdr, pos)
15:         hdr = GetNextHeaderType(pkt, hdr, pos)
16:         pos = pos + len
17:     end while
18:     Extract MessageBody of the application protocol.
19:     Save MessageBody into a binary file.
20: return Packet_Binary;
21: end function
22: function StaticAnalysis(binary)
23:     invoke uploadVirusTotal(binary);
```

```
24:     repeat
25:         response = getVirusTotalResponse();
26:     until response==null
27:     if (detectionRatio! = 0) then
28:         Upload VirusTotal result in database.
29:         BREAK;
30:     else Continue;
31:     end if
32:     packer = Obfuscation(binary);
33:     Upload packer information
34:     header= PEstructure(binary);
35:     Upload header
36:     hash= Hash(binary);
37:     Upload hash
38:     response[] = Strings(binary);
39:     Upload list of embedded strings in database
40: end function
41: function DynamicAnalysis(binary)
42:     for binary in DB do
43:         ExecuteOnReal(binary);
44:         filesystem_activity [] = FileSystem_Monitor();
45:         Upload database filesystem_activity
46:         registry_activity [] = Registry_Monitor();
47:         Upload database registry_activity
48:         process_activity [] = Process_Monitor();
49:         Upload database process_activity
50:         network_activity [] = Network_Monitor();
51:         Upload database network_activity
52:         memory_activity [] = Memory_Monitor();
53:         Upload database memory_activity
54:     end for
55: end function
56: function ManualAnalysis(binary)
57:     Animate instructions in ollydbg.
58:     Upload result to database.
59: end function
60: function Reporting()
61:     Process user request- Controller();
62:     Identify report format- View();
63:     Read analysis data from database- Model();
64:     Populate the View with analysis data.
65:     Export the report into HTML or PDF.
66: end function
```

3.6 Summary

This chapter discussed the design and implementation of RADAR system. RADAR is a hybrid system combining anomaly, behavioral and signature detection techniques. RADAR addresses the research problems with existing approaches in zero-day attack detection and analysis and tries to provide a complete solution to the whole problem. It does so by a layered designed where each layer is dedicated to a single functionality and works in parallel to improve performance. RADAR employs 1-class SVM as an anomaly detection technique in detection layer to detect zero-day attacks that diverts from the good traffic profile.

The analysis layer in the RADAR system captures both static and dynamic behavior of malicious binaries captured in the detection layer. The analysis and reporting stub in analysis layer integrates existing static and dynamic malware analysis functionality and utilities to work as a single unit in a component based architecture where any of the function or utility can be replaced in the future. The SAE combines popular static tools and provides the basic information to profile the malicious binary. The DAE captures run-time behavior and has the capability to evade anti-emulation and anti-debugging checks of a malicious binary which may hinder the malware analysis process. Manual analysis is also intromitted to do step by step analysis of binary if needed. RADAR reports zero-day malware behavior in HTML or PDF format.

Chapter 4

Experimental Details and Results

4.1 Introduction

This chapter discusses the process of realizing RADAR in practice according to the proposed work. For implementing RADAR an isolated network is setup in the research lab. The setup requires different dedicated systems to run detection software, analysis functionality and the other supporting software. After setting the environment, RADAR is tested to detect zero-day attacks and to generate comprehensive analysis reports. The detection results are evaluated using standard metrics and analysis reports are evaluated by matching analysis information with the analysis results provided by well-known online virus and behavioral scan engines. RADAR was tested with two types of datasets: Synthetic dataset and Real traffic dataset.

Section 4.2 discusses the details about the environment and the technologies used to implement RADAR. Overall experimental results of RADAR are divided in two categories: detection results (Section 4.3.1) and analysis reports (Section

4.3.2). Section 4.4 provides a comparison between RADAR and other existing tools and techniques. RADAR is compared with Honeynet, where it is proved that RADAR offers a single integrated solution by bridging the gap between detection and analysis phase. Further RADAR's analysis and reporting stub is compared with known malware analysis tools. Section 4.5 summaries the chapter.

4.2 Experimental Setup

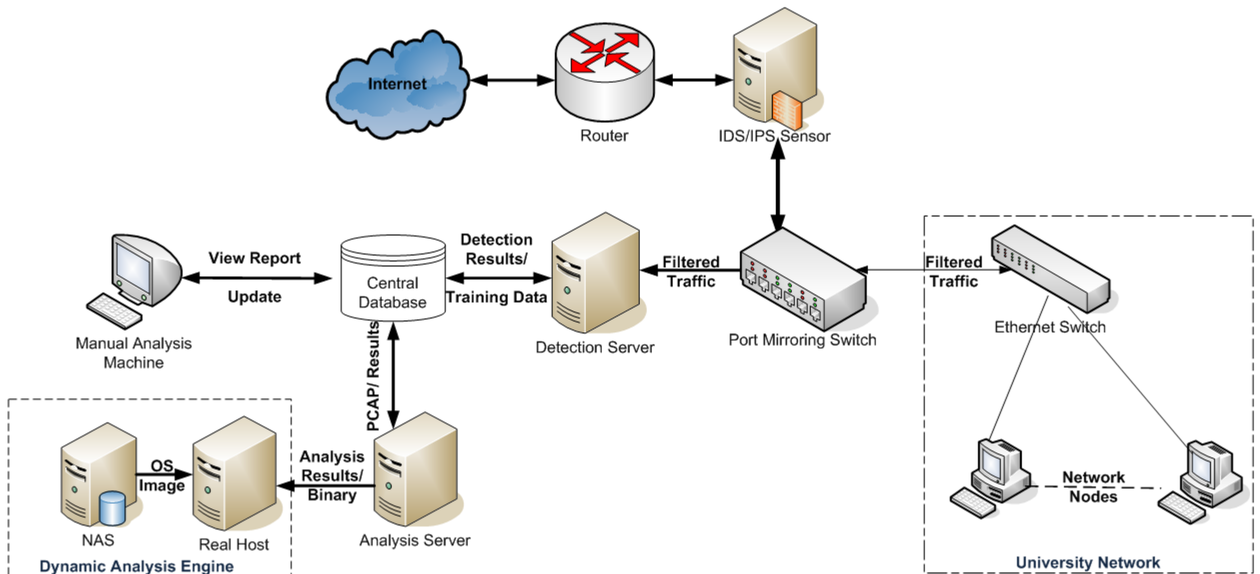


Figure 4.1: RADAR Experimenal Setup

Figure 4.1 represents an implementation setup of RADAR system. It comprises of router, IDS/IPS sensor, Ethernet switch, Intranet machines and RADAR components: the detection server (Core i7, 8Gb, 1Tb, Linux), analysis server (Core i7, 8Gb, 1Tb, Linux), a real host (Intel Core i5, 8GB, 1Tb, Win 8.1) with kernel-based monitoring (KBM) and a central database. The detection server receives raw traffic from router, detect unknown attacks and pass results to a central

database. From central database the analysis server fetches the current results and accordingly extracts binary for further analysis on real host. A system is directly connected to the analysis server for viewing reports and to do manual analysis. The RADAR system is setup in an isolated environment in the research lab of Thapar University.

Software packages used to implement RADAR are shown in Table 4.1

Table 4.1: Software Packages Used

Tools	**Purpose**
Snort	to filter known attacks
SnortAD 3.0	modified, to detect unknown attacks
LibSVM SDK	to implement SVM
Mysql	as central database
Visual Studio Enterprise 2015, WDK 10, SDK 10, SDK 8.1 on Windows 8.1	to develop 32-bit minifilter driver
JFreeCharts 1.0.19	to report analysis results
Ollydbg 2.01	for manual analysis
Python 2.7	for implementing SAE and data parsing
Oracle Java SDK	to implement and integrate various components in RADAR
Eclipse IDE for Java	for Java development

4.3 Experimental Results

The overall experimental results of RADAR are divided into: (1) Detection Results and (2) Analysis Results.

4.3.1 Detection Results and Discussions

RADAR was tested with two types of datasets: Synthetic dataset and Real traffic dataset. Synthetic dataset is small dataset comprising of fabricated malware. Real traffic dataset is a larger dataset comprising of original attacks collected from various online malware repositories. To evaluate the performance of RADAR following standard metrics were used: True Positive Rate (TPR), False Positive Rate (FPR), F-Measure, Total Accuracy (ACC) and Receiver Operating Characteristic (ROC) curve.

TPR is the percentage of correctly identified malicious code shown in Equation 4.1. FPR is the percentage of wrongly identified benign code (Equation 4.1). F-measure is a measure of a test's accuracy by combining recall (same as TPR) and precision (Equation 4.2) scores into a single measure of performance as in Equation 4.3. ACC is the percentage of absolutely correctly identified code, either positive or negative, divided by the entire number of instances as shown in Equation 4.4. In ROC curve the TPR rate is plotted in function of the FPR for different points. The ROC curve shows a trade-off between true positive and false positive. In the equations below, True Negative (TN) is the number of correctly identified benign code and False Negative (FN) is the number of wrongly

identified malicious code.

$$TPR = \frac{|TP|}{|TP| + |FN|}; \quad FPR = \frac{|FP|}{|FP| + |TN|} \tag{4.1}$$

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{4.2}$$

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{4.3}$$

$$ACC = \frac{|TP| + |TN|}{|TP| + |FP| + |TN| + |FN|} \tag{4.4}$$

Testing with Synthetic Dataset:

For generating synthetic dataset, various known attacks have been fabricated to act as zero-day for the system. Metasploit Framework [Fra14] is used to generate and encode known payloads. The existing payloads were encoded by both simple encoding techniques like few bit manipulations and by advance encoding engines. Encoders like XOR Encoder, Base64, Alpha2, Countdown, JmpCallAdditive, FnstenvMov, and ShigataGaNai shellcode were used. Advance polymorphic shellcode engines like Clet [DUMU03], ADMmutate [K214], and TAPiON [Ban05] were utilized. Some shellcodes were also encoded using an emerging public-key cryptography technique, Elliptic Curve Cryptography (ECC) [GSK10].

A total of 1300 exploits were used for the experiment. Table 4.2 represent detection results for synthetic dataset. In best case i.e. for simple encoding methods like XOR Encoder, sensitivity is 99.6% and specificity is 99%. In the

worst case, where a complex cryptography technique is used to mutate shellcode, sensitivity drops to 88% and specificity to 88.3%.

Table 4.2: Detection Results for Synthetic Dataset

Polymorphic Engines/Encoders	TPR	FPR	ROC
AdMmutate	0.990	0.018	0.981
Clet	0.959	0.038	0.966
CountDown	0.948	0.051	0.967
JmpCallAdditive	0.986	0.023	0.982
FnStenvMov	0.934	0.063	0.932
ShikataGaNai	0.981	0.022	0.982
TAPiON	0.983	0.025	0.934
XOR Encoder	0.996	0.013	0.991
Base64	0.995	0.021	0.983
ECC	0.880	0.117	0.875

Testing with Real Traffic Dataset:

This dataset comprises of 5000 samples in total consisting of 4000 malware samples (both obfuscated & non-obfuscated) and 1000 benign samples. The unknown malware samples were collected from various sources like Honeynet project, VX heavens [Hea14] and free online malware repositories like Malshare, Contagio, Open Malware, etc. The benign samples include: application software, system software, legitimate executables, documents and many other user applications. These benign samples were collected from various trusted systems in the TU production network. The distribution of benign samples are represented in Table 4.3. The distribution of malware samples both non-obfuscated and obfuscated are

Table 4.3: Distribution of Benign Samples

Benign Sample Type	**No. of Samples**
Application Software & User Applications	500
System Software	50
Legitimate PE files	250
Legitimate Documents	200

shown in Table 4.4. Common malware types like viruses, network worms and trojans constitutes the major samples in the dataset. Other types included backdoors, buffer overflow exploits and rootkits.

Table 4.4: Distribution of Malware Samples

Malware Type	**No. of Samples**	**Non-Obfuscated**	**Obfuscated**
Virus	1200	500	700
Worm	1000	400	600
Rootkit	100	50	50
Backdoor	600	250	350
Exploit	200	50	150
Trojan	900	350	550

To compute the detection accuracy of RADAR, both benign and malicious samples were targeted to the system setup. The packets unknown to the system were identified by the detection layer with few escapes. The The recorded values of TPR, FPR, Precision, Recall, F-Measure, ACC and ROC for non-obfuscated zero-day malware are presented in Table 4.5. In best case, Sensitivity = 98.4% and Specificity = 97.7%. In the worst case, Sensitivity = 90.2% and Specificity = 96.7%.

Table 4.5: Detection Accuracy For Zero-day Non-Obfuscated Malware

Malware Type	**Non-Obfuscated**						
	TPR	*FPR*	*Precision*	*Recall*	*F-Measure*	*ACC*	*ROC*
Virus	0.937	0.031	0.961	0.937	0.952	0.951	0.965
Worm	0.966	0.052	0.954	0.966	0.931	0.944	0.934
Rootkit	0.984	0.023	0.992	0.984	0.986	0.971	0.981
Backdoor	0.973	0.026	0.981	0.973	0.982	0.976	0.975
Exploit	0.984	0.032	0.968	0.984	0.983	0.985	0.982
Trojan	0.902	0.033	0.959	0.902	0.935	0.956	0.935

Same standard intrusion detection metrics were recorded for obfuscated zero-day malware are shown in Table 4.6. In best case, Sensitivity = 97.2% and Specificity = 96.9%. In the worst case, Sensitivity = 89.8% with Specificity = 93.5%.

Table 4.6: Detection Accuracy For Zero-day Obfuscated Malware

Malware Type	**Obfuscated**						
	TPR	*FPR*	*Precision*	*Recall*	*F-Measure*	*ACC*	*ROC*
Virus	0.918	0.056	0.946	0.918	0.933	0.932	0.931
Worm	0.942	0.081	0.924	0.941	0.932	0.931	0.927
Rootkit	0.966	0.033	0.970	0.966	0.961	0.964	0.956
Backdoor	0.959	0.061	0.933	0.959	0.945	0.961	0.967
Exploit	0.972	0.031	0.968	0.972	0.980	0.981	0.979
Trojan	0.898	0.065	0.933	0.898	0.928	0.945	0.935

The above values recorded in Table 4.5 and Table 4.6 for both non-obfuscated and obfuscated zero-day malware respectively, varies w.r.t the encoding or encryption technique applied for a particular malware.

4.3.2 Analysis Results and Discussions

For assessing the analysis result the analysis report generated by RADAR is compared to the information provided by popular online malware scan engines like Cuckoo and Anubis. The prime objective of the evaluation is to determine to what extent analysis result matches the characterization provided by these scan engines. The analysis report generated by RADAR shows somewhat different results. The logical reason for this variation is analysis on different execution environment with different analysis technique implemented. To assess the information provided by RADAR's analysis layer a case study is presented.

Figure 4.2: Working JAVA Reverse Bind Shell

Case Study with JAVA Reverse Bind Shell:

For this study, a reverse bind exploit is written in java named *RADARReverse-shell.java*. The running program is shown in Figure 4.2. The reverse shell connects to host and opens a port for attack. This file is submitted to VirusTotal to know which antivirus software can detect java reverse bind shell. Figure

4.3 shows VirusTotal scan results for *RADARReverseshell.java* where all good antivirus software's were evaded. *Detection ratio = 0/56.*

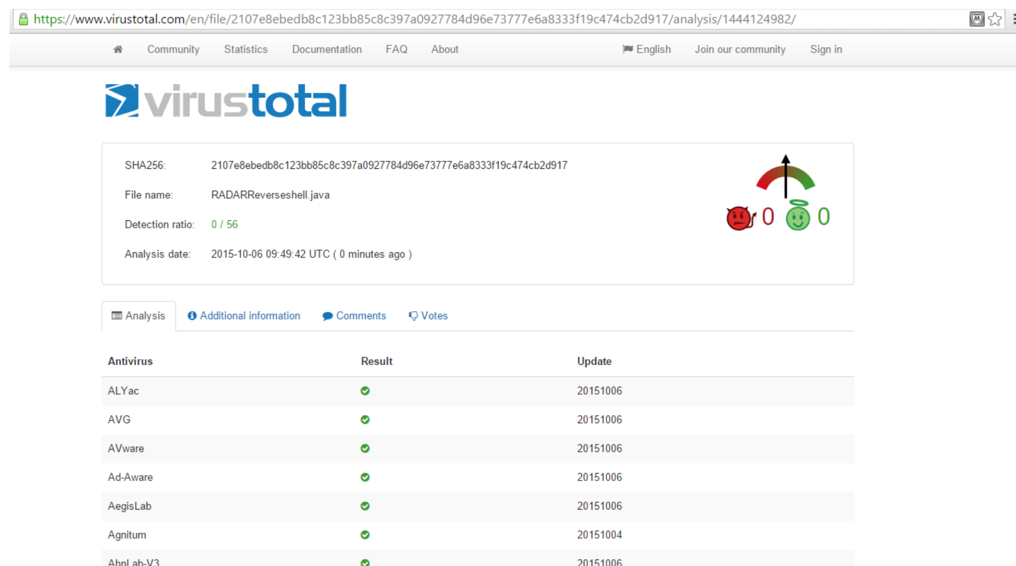


Figure 4.3: VirusTotal Result for JAVA file

Now converting *RADARReverseshell.jar* to *RADARReverseshell.exe* and scanning the file on VirusTotal. Figure 4.4 shows VirusTotal detection results. *Detection ratio = 2/56.* On searching Web, no detailed description of *Trojan.PWS.Panda.9140* was provided at *http://live.drweb.com/* site. Searching for *Trojan.PWS.Panda* and *Trj/Genetic.gen* gave simple explanation of generic trojan family. Reverse bind shell is a trojan that opens a backdoor for the attacker but none search result accurately indicated it as reverse bind exploit. *RADARReverseshell.exe* is not a pure zero-day but it has evaded existing *54* antivirus engines including Symanetc, BitDefender, McAfree, Kaspersky. Therefore, *RADARReverseshell.exe* is a suitable candidate for testing analysis reports.

Generating Analysis Reports: RADAR generates simple and detailed analysis

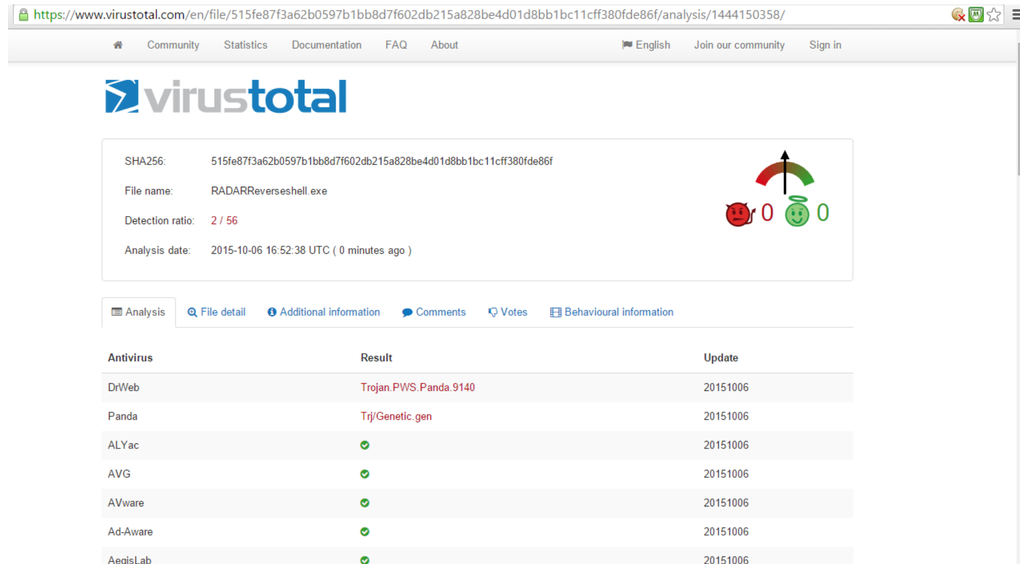


Figure 4.4: VirusTotal Result for EXE file

reports for zero-day malware. Each report is further sub-categorized into different sections- general, static, dynamic, manual. General section gives a brief overview of the malware. Figure 4.5 shows general details of the *RADARReverseshell.exe* file.

Under static category, structural details of the malware such as sections, imports, exports, resources and extracted strings are presented. Figure 4.6 represents sections of the executable under study. It has 5 sections, .text (holds executable code), .rdata (for constant/read-only data), .data (holds global variables), .SHMMESS (an unknown section) and .rsrc (contains resource information).

Figure 4.7 list APIs from libraries(DLL) that will be used by the exe at runtime. The import table creates an array of pointers at runtime, each pointing to the memory address of an imported api.

Figure 4.8 list export functions in the exe which can be shared by other programs or libraries.

RADAR
Real Time Zero Day Attack Detection Analysis and Reporting

THAPAR UNIVERSITY

Search Malware...

General | Static | Dynamic | Manual | Help

General	
Sample Name:	RARARReverseshell.exe
Filetype:	PE32 executable (console) Intel 80386, for MS Windows
Size:	181417 bytes
MD5 Hash:	692f55c89a0d11a8ce2f5fa6d826a2f4
SHA1 Hash:	f13b258dba9ff559764322b2be767487018da034
SHA256 Hash:	515fe87f3a62b7950b1bb8d7f602db215a828be4d01d8bb1bc11cff380fde86f
Description:	This file is the main program(Executive Created by Jar2Exe v2.0, jar2exe.com)
Packer:	F-PROT appended, ZIP
Time Date Stamp:	30-08-2015 14:01
Subsystem:	IMAGE_SUBSYSTEM_WINDOWS_CUI
Number of Sections:	5
Summary:	Found potential IP address in binary/memory; No dropped files; No HTTP request made; No DNS request made; No hosts connected; No domains Connected.

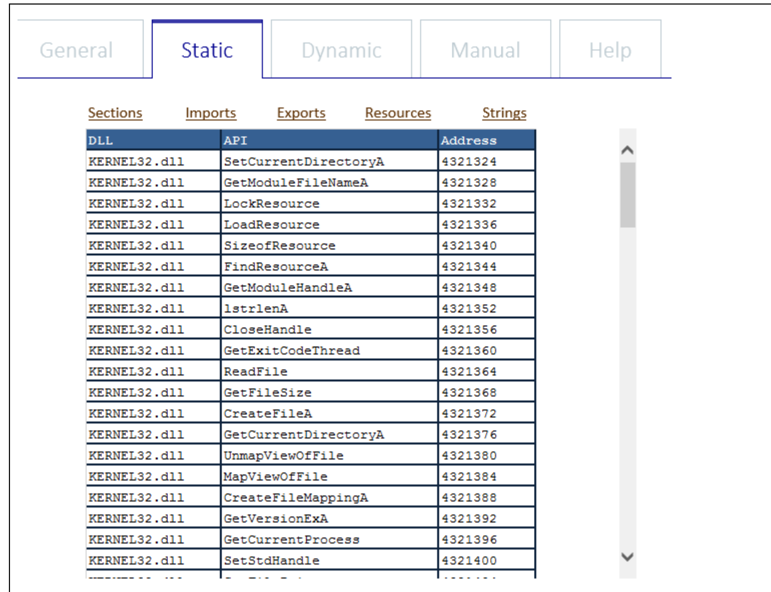
Figure 4.5: General Details

General | Static | Dynamic | Manual | Help

Sections   Imports   Exports   Resources   Strings

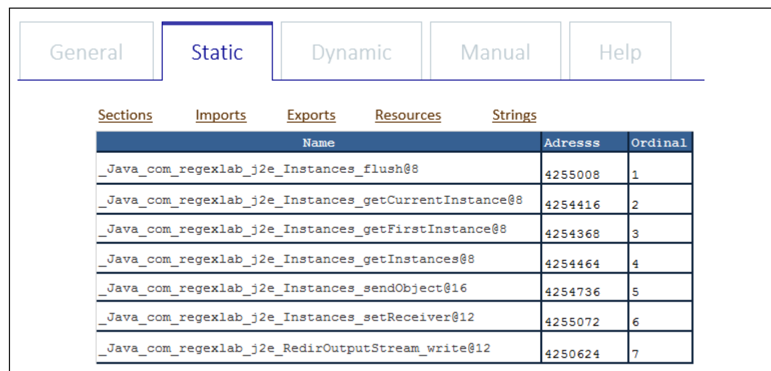
NAME	VIRTUAL ADDRESS	VIRTUAL SIZE	SIZE OF RAW DATA	MD5	ENTROPY
.text	4096	121346	122880	8c484d0c1c752347f2bbd3bc7a132	6.56282712
.rdata	126976	14688	16384	d0de0c7076f68c946cd5a5079a120	4.62871617
.data	143360	22372	16384	89a00f0ab9b2d6078a2b6de502417	2.32544669
.SHMMESS	167936	3072	4096	620f0b67a91f7f74151bc5be745b7	0.0
.rsrc	172032	10184	12288	2c505e319fea0b04bcd4332f71e02	3.50293526

Figure 4.6: Sections



DLL	API	Address
KERNEL32.dll	SetCurrentDirectoryA	4321324
KERNEL32.dll	GetModuleFileNameA	4321328
KERNEL32.dll	LockResource	4321332
KERNEL32.dll	LoadResource	4321336
KERNEL32.dll	SizeofResource	4321340
KERNEL32.dll	FindResourceA	4321344
KERNEL32.dll	GetModuleHandleA	4321348
KERNEL32.dll	lstrlenA	4321352
KERNEL32.dll	CloseHandle	4321356
KERNEL32.dll	GetExitCodeThread	4321360
KERNEL32.dll	ReadFile	4321364
KERNEL32.dll	GetFileSize	4321368
KERNEL32.dll	CreateFileA	4321372
KERNEL32.dll	GetCurrentDirectoryA	4321376
KERNEL32.dll	UnmapViewOfFile	4321380
KERNEL32.dll	MapViewOfFile	4321384
KERNEL32.dll	CreateFileMappingA	4321388
KERNEL32.dll	GetVersionExA	4321392
KERNEL32.dll	GetCurrentProcess	4321396
KERNEL32.dll	SetStdHandle	4321400

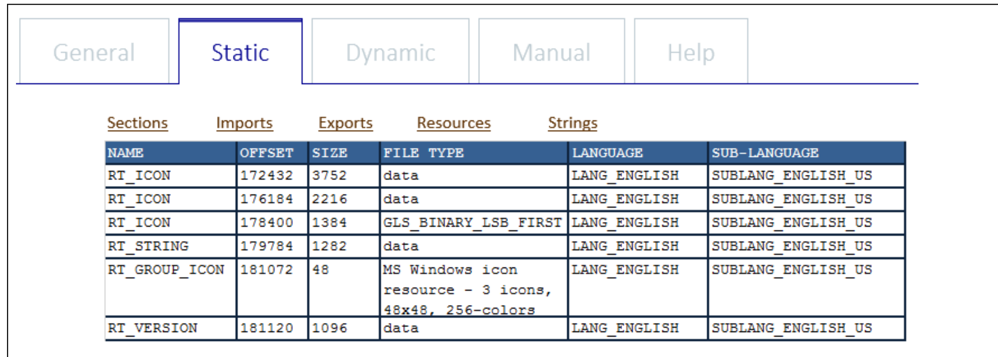
Figure 4.7: Imports



Name	Adresss	Ordinal
_Java_com_regexlab_j2e_Instances_flush@8	4255008	1
_Java_com_regexlab_j2e_Instances_getCurrentInstance@8	4254416	2
_Java_com_regexlab_j2e_Instances_getFirstInstance@8	4254368	3
_Java_com_regexlab_j2e_Instances_getInstances@8	4254464	4
_Java_com_regexlab_j2e_Instances_sendObject@16	4254736	5
_Java_com_regexlab_j2e_Instances_setReceiver@12	4255072	6
_Java_com_regexlab_j2e_RedirOutputStream_write@12	4250624	7

Figure 4.8: Exports

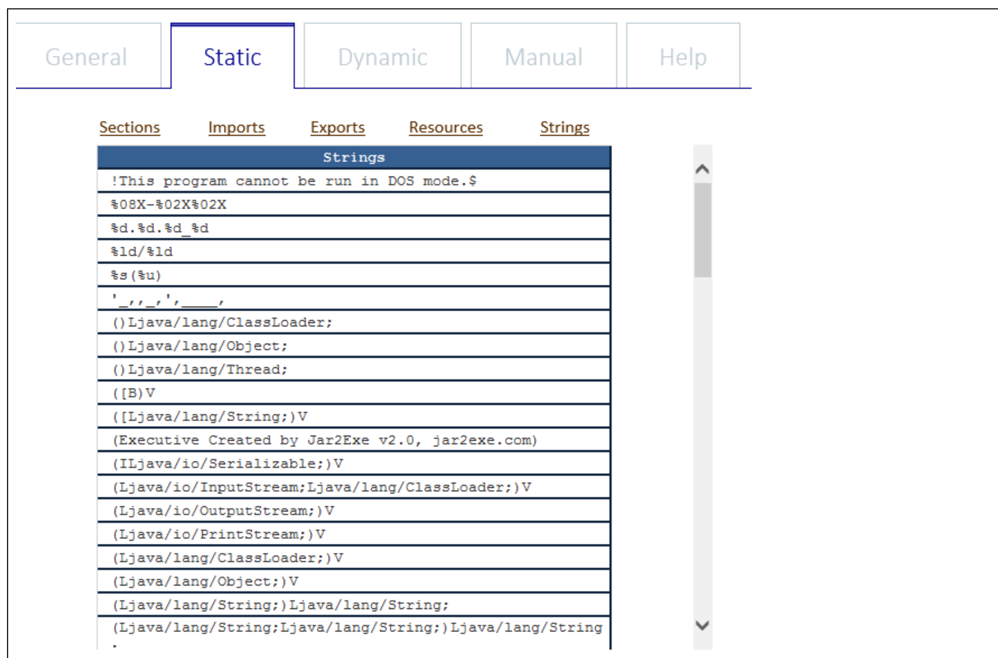
Figure 4.9 represents resources of the exe. It contains icons, string data and version data.



NAME	OFFSET	SIZE	FILE TYPE	LANGUAGE	SUB-LANGUAGE
RT_ICON	172432	3752	data	LANG_ENGLISH	SUBLANG_ENGLISH_US
RT_ICON	176184	2216	data	LANG_ENGLISH	SUBLANG_ENGLISH_US
RT_ICON	178400	1384	GLS_BINARY_LSB_FIRST	LANG_ENGLISH	SUBLANG_ENGLISH_US
RT_STRING	179784	1282	data	LANG_ENGLISH	SUBLANG_ENGLISH_US
RT_GROUP_ICON	181072	48	MS Windows icon resource - 3 icons, 48x48, 256-colors	LANG_ENGLISH	SUBLANG_ENGLISH_US
RT_VERSION	181120	1096	data	LANG_ENGLISH	SUBLANG_ENGLISH_US

Figure 4.9: Resources

Figure 4.10 shows extracted ASCII/Unicode strings from the exe file.



Strings
!This program cannot be run in DOS mode.$
%08X-%02X%02X
%d.%d.%d_%d
%ld/%ld
%s(%u)
'_,,_,',____,
()Ljava/lang/ClassLoader;
()Ljava/lang/Object;
()Ljava/lang/Thread;
([B)V
([Ljava/lang/String;)V
(Executive Created by Jar2Exe v2.0, jar2exe.com)
(ILjava/io/Serializable;)V
(Ljava/io/InputStream;Ljava/lang/ClassLoader;)V
(Ljava/io/OutputStream;)V
(Ljava/io/PrintStream;)V
(Ljava/lang/ClassLoader;)V
(Ljava/lang/Object;)V
(Ljava/lang/String;)Ljava/lang/String;
(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;

Figure 4.10: Extracted Strings

The dynamic tab displays behavioral information captured during the execution of *RADARReverseshell.exe* on real host. Figure 4.11 represents summary of

dynamic report. More detailed graphs are further shown in Figure 4.12 after clicking link under the piechart.

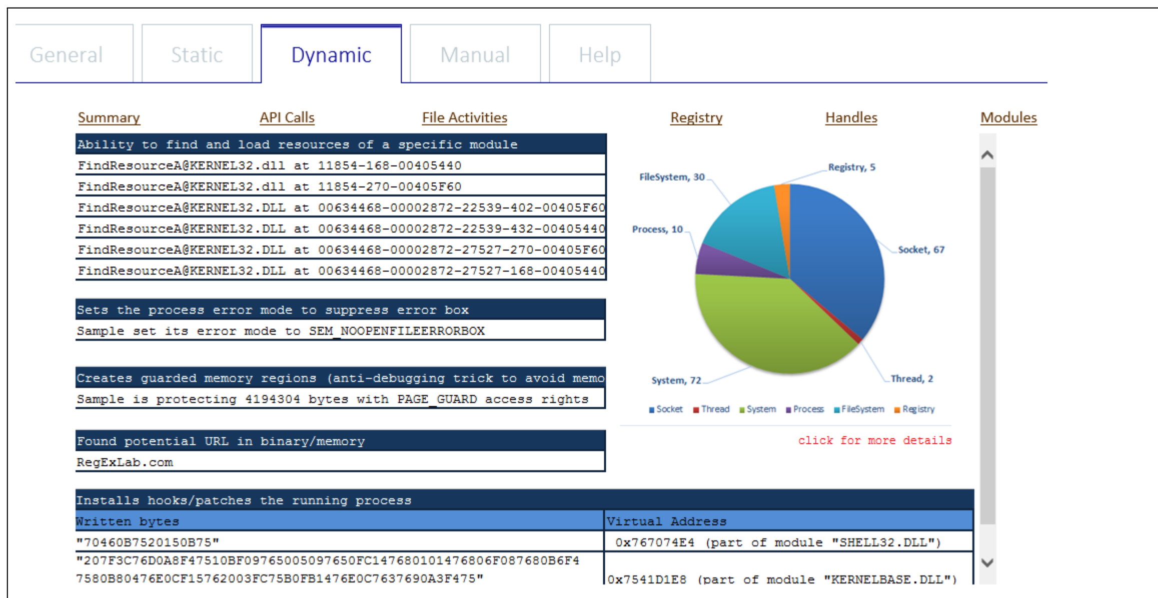


Figure 4.11: Dynamic Analysis Summary

Dynamic tab also provides information regarding API calls and Registry. Figure 4.13 shows a call to NtWriteFile native api in NTDLL.DLL. This api is writing program code contents to an open file handle.

Figure 4.14 shows registry files open and values read for Winsock and TCPIP services. Indicating values read for sending data over TCP connection.

Figure 4.15 shows file accessed by malware while running. Figure 4.16 shows information about which handles have been opened. Figure 4.17 displays modules loaded during run time.

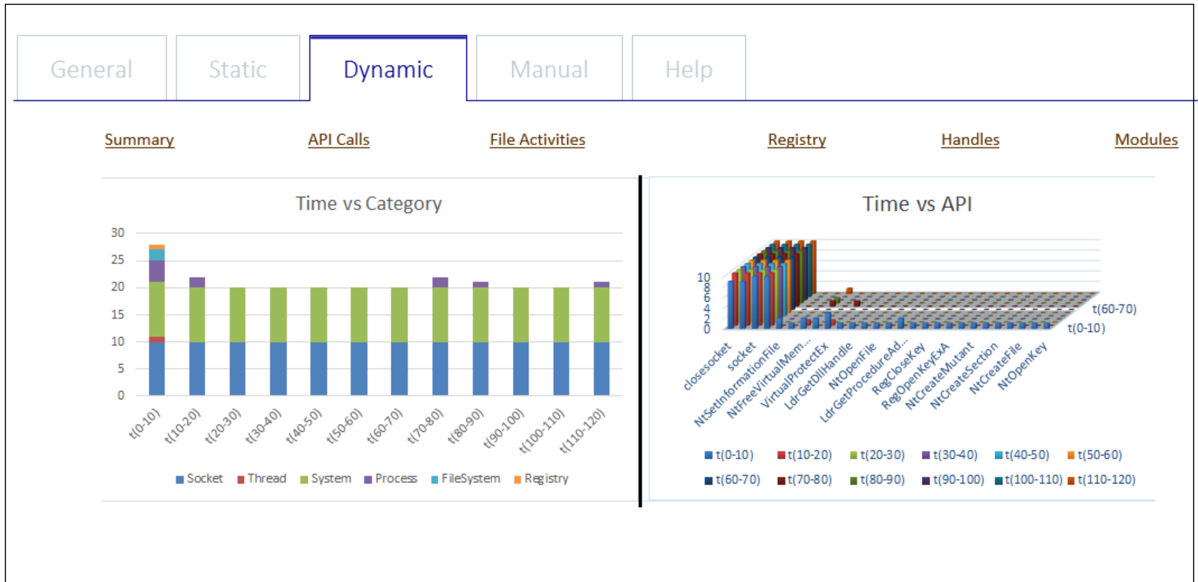


Figure 4.12: Dynamic Activity w.r.t Time

General | Static | Dynamic | Manual | Help

Summary | API Calls | File Activities | Registry | Handles | Modules

Name	Parameters	Values
NtWriteFile@NTDLL.DLL	FileHandle	28
	Buffer	Starting Reverse bind shell ....
	Length	33
NtWriteFile@NTDLL.DLL	FileHandle	28
	Buffer	ListenIP: 127.0.0.1
	Length	19
NtWriteFile@NTDLL.DLL	FileHandle	28
	Buffer	Listenport : 41445
	Length	18
NtCreateFile@NTDLL.DLL	DesiredAccess	3222536192
	ObjectAttributes	18000000000000064f6be014200000000000000000000000
	ShareAccess	3
	CreateDisposition	3
	EaLength	57
	(name)	\Device\Afd\Endpoint

Figure 4.13: API Calls

General | Static | Dynamic | Manual | Help

Summary | API Calls | File Activities | Registry | Handles | Modules

Action	Group	Path
QUERYVAL	HKLM	\System\CurrentControlSet\Control\Nls\Locale
QUERYVAL	HKLM	\System\CurrentControlSet\Control\Nls\Locale\Alternate Sorts
QUERYVAL	HKLM	\System\CurrentControlSet\Control\Nls\Language Groups
QUERYVAL	HKLM	\SOFTWARE\JavaSoft\Java Runtime Environment
QUERYVAL	HKLM	\SOFTWARE\JavaSoft\Java Runtime Environment\1.6
QUERYVAL	HKLM	\System\CurrentControlSet\Services\Tcpip\Parameters
QUERYVAL	HKCU	\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders
OPEN	HKLM	\REGISTRY\MACHINE\SOFTWARE\POLICIES\MICROSOFT\SQMCLIENT\WINDOWS
OPEN	HKLM	\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\SQMCLIENT\WINDOWS
QUERYVAL	HKLM	\REGISTRY\MACHINE\SOFTWARE\MICROSOFT\SQMCLIENT\WINDOWS
OPEN	HKLM	\REGISTRY\MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\WINSOCK\PARAMETERS
QUERYVAL	HKLM	\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK\PARAMETERS
QUERYVAL	HKLM	\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\WINSOCK\PARAMETERS
OPEN	HKLM	\REGISTRY\MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\TCPIP\PARAMETERS\WINSOCK
QUERYVAL	HKLM	\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPIP\PARAMETERS\WINSOCK
QUERYVAL	HKLM	\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPIP\PARAMETERS\WINSOCK
OPEN	HKLM	\REGISTRY\MACHINE\SYSTEM\CURRENTCONTROLSET\SERVICES\TCPIP6\PARAMETERS\WINSOCK
QUERYVAL	HKLM	\REGISTRY\MACHINE\SYSTEM\CONTROLSET001\SERVICES\TCPIP6\PARAMETERS\WINSOCK

Figure 4.14: Registry Activity

General | Static | Dynamic | Manual | Help

Summary | API Calls | File Activities | Registry | Handles

File Activities
C:\DOCUME~1\User\LOCALS~1\Temp\RADARReverseshell.exe
C:\DOCUME~1\User\LOCALS~1\Temp\./jre/*
C:\DOCUME~1\User\LOCALS~1\Temp\./jre/bin*
C:\DOCUME~1\User\LOCALS~1\Temp\./jre/jre\bin*
C:\DOCUME~1\User\LOCALS~1\Temp\./jre/..\jre\bin*
C:\DOCUME~1\User\LOCALS~1\Temp\.hotspotrc
C:\
C:\DOCUME~1\User\LOCALS~1\Temp\\hsperfdata_User*.*
C:\DOCUME~1\User\LOCALS~1\Temp\hsperfdata_User
C:\DOCUME~1\User\LOCALS~1\Temp\hsperfdata_User\1592
C:\Program Files\Java\jre6\lib\resources.jar
C:\Program Files\Java\jre6\lib\rt.jar
C:\Program Files\Java\jre6\lib\sunrsasign.jar
C:\Program Files\Java\jre6\lib\jsse.jar
C:\Program Files\Java\jre6\lib\jce.jar
C:\Program Files\Java\jre6\lib\charsets.jar
C:\Program Files\Java\jre6\lib\modules\jdk.boot.jar
C:\Program Files\Java\jre6\classes
C:\Program Files\Java\jre6\lib\meta-index
C:\Program Files\Java\jre6\bin\client\classes.jsa

Figure 4.15: File Activity

General Static Dynamic Manual Help

Summary API Calls File Activities Registry Handles Modules

Type	Value	Handle
KeyHandle	340	\REGISTRY\MACHINE\SOFTWARE\Microsoft\SQMClient\Windows
KeyHandle	344	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\WinSock2\Parameters\Protocol_Cata
KeyHandle	352	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\WinSock2\Parameters\Protocol_Cata
KeyHandle	360	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\WinSock2\Parameters\NameSpace_Cat
FileHandle	376	\Device\HarddiskVolume2\Users\Payload\AppData\Local\Temp\hsperfdata_Payload
FileHandle	368	\Device\HarddiskVolume2\Users\Payload\AppData\Local\Temp\hsperfdata_Payload\2872
FileHandle	392	\Device\HarddiskVolume2\Program Files\Java\jre1.8.0_25\lib
FileHandle	400	\Device\HarddiskVolume2\Program Files\Java\jre1.8.0_25\lib
FileHandle	404	\Device\HarddiskVolume2\Program Files\Java\jre1.8.0_25\lib
SectionHandle	588	\KnownDlls\SHCORE.dll
SectionHandle	592	\Sessions\1\BaseNamedObjects\windows_shell_global_counters
FileHandle	940	\Device\HarddiskVolume2\Program Files\Java\jre1.8.0_25\lib\accessibility.properti
KeyHandle	944	\REGISTRY\MACHINE\SOFTWARE\Microsoft\SQMClient\Windows
KeyHandle	952	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\Winsock\Setup Migration\Providers
KeyHandle	956	\REGISTRY\MACHINE\SYSTEM\ControlSet001\Services\TCPIP6\Parameters\Winsock

Figure 4.16: Handles

General Static Dynamic Manual Help

Summary API Calls File Activities Registry Handles Modules

Module Path	Base Address
C:\Windows\SYSTEM32\VERSION.dll	74040000
C:\Windows\SYSTEM32\SspiCli.dll	74EA0000
C:\Windows\system32\KERNELBASE.dll	75350000
C:\Windows\system32\CFGMGR32.dll	75430000
C:\Windows\system32\msvcrt.dll	75470000
C:\Windows\system32\RPCRT4.dll	755A0000
C:\Windows\system32\SETUPAPI.dll	75780000
C:\Windows\system32\PSAPI.DLL	75B40000
C:\Windows\system32\ADVAPI32.dll	75B50000
C:\Windows\SYSTEM32\sechost.dll	77340000
C:\Windows\system32\GDI32.dll	773F0000
C:\Windows\system32\USER32.dll	77510000
C:\Windows\SYSTEM32\ntdll.dll	77850000
C:\Windows\system32\KERNEL32.DLL	75680000
C:\Program Files\Java\jre1.8.0_25\bin\splashscreen.dll	69200000
C:\Program Files\Java\jre1.8.0_25\bin\client\jvm.dll	648F0000
C:\Program Files\Java\jre1.8.0_25\bin\java.dll	67BB0000
C:\Program Files\Java\jre1.8.0_25\bin\verify.dll	691F0000
C:\Windows\system32\SHELL32.dll	75F20000
C:\Program Files\Java\jre1.8.0_25\bin\zip.dll	67B90000

Figure 4.17: Run time Modules

4.4 Comparison with Existing Techniques

In this section features of RADAR system are compared with existing zero-day attack detection and analysis techniques. First comparison is done with Honeynet system (Table 4.7). A Honeynet is a network setup that invites attackers to compromise the system (honeypots) and do harm in a controlled and isolated environment, while their activities are monitored and studied to increase network security. Honeynet has been found effective against zero day attacks. It identifies the mechanism of a new attack and collects evidence for attacker's activities, which is later analyzed by a human expert. This analysis is done by first preparing a toolkit comprised of (but not limited to) physical or virtual systems, behavioral analysis tools, code analysis tools and online analysis tools.

Table 4.7: Comparison: Honeynet vs RADAR

Techniques → Features ↓	**Honeynet**	**RADAR**
Known Attack Detection	Snort in honeywall log and report known attacks	Snort in inline mode and VirusTotal is used to keep check on known attacks
Zero-day Attack Detection	The unknown traffic is redirected to honeypots to monitor interactions between attacker and honeypot	Utilized machine learning algorithm, 1-class SVM to detect unknown attacks that deviate from the good traffic profile
Obfuscation Detection	The obfuscated binary is allowed to run on honeypot with Sebek to track commands	Detect obfuscation in SAE and later the binary is allowed to run on a real host.
Attack Analysis	Analysis is only done manually.	Automated analysis: static and dynamic.
Reporting	Report data through Walleye with limited information	Detailed reporting of malware behavior in HTML and PDF format.
Response Time	Manual analysis takes time to analyze the behavior of malicious binary	Layered architecture does detection and analysis in parallel. Further, SAE and DAE provides detailed and useful information for manual analysis (if required). Hence reducing response time.

All such tools are run separately with human intervention. This takes time sometimes weeks or months and requires high expertise to report a zero-day attack behavior. To address these issues RADAR provides a single automated solution combining static and dynamic malware analysis. On the other hand, Honeynet is not a detection system, it only traps and monitors unknown attack activities. The detection layer in RADAR detects zero-day attacks against good traffic profile build from trustworthy systems. The following comparison shows RADAR to be more efficient in delivering a complete solution to zero-day attack detection and analysis.

Next comparison (Table 4.8) is done with popular malware analysis tools such as Anubis, CWSandbox, Norman, JoeBox and Cuckoo. RADAR's analysis and reporting stub is compared on the basis of analysis implementation, target and technique used. In case of zero-day attacks where there is no advance knowledge of attack behavior, analysis implementation on a physical machine offers full visibility to capture full zero-day attack behavior. Moreover, kernel-mode component allows simple and stealthy analysis in comparison to user-mode component. RADAR's analysis and reporting stub targets modern versions of Windows including 64-bit. It dynamically monitors system activity by intercepting kernel functions. Intercepting kernel functions is more reliable, clean and stealthy technique than hooking.

Table 4.8: Comparison: Analysis Tools vs RADAR Analysis

Features	Anubis	CWSandbox	Norman	JoeBox	Cuckoo	RADAR
Analysis Implementation	Full system emulation	User and kernel mode component	Full system simulation	User and kernel mode component	Virtual machine monitor	Kernel mode component
Analysis Target	Windows XP (SP2/SP3), 32 bit	Windows (2000, XP, Vista), 32 bit	Windows (2000, 2003, XP), 32bit	Windows (XP, Vista, 7, 7x64, 8), 32 bit	Windows (XP-SP3, Vista, 7), 32 bit	Windows (2000, XP, Vista, Server, 7, 8), 32/64 bit.
Analysis Technique	API, System Call and Native APIs hooking	API and System Call hooking	API hooking	API and System Call hooking	API and Native APIs hooking	Intercept Kernel Routines

4.5 Summary

This chapter deals with the experiments and results part of the thesis. After giving the basic requirements for our implementation we have proposed the system that deals with the problems discussed in Chapter 2. The system is implemented using JAVA as programming language, and few other tools are modified and integrated to develop a system called RADAR. The RADAR system was evaluated by various standard metrics. In experiments it was shown that RADAR provides the best detection rate of nearly 98% with 0.02 false positives. Furthermore, the comparison with Honeynet system depicts that RADAR system would need very little human intervention for analyzing zero-day malware hence, will minimize the response time in zero-day attack detection and analysis.

Chapter 5

Conclusions and Future Work

This chapter summarizes the thesis by providing a meta-view of the work done, the findings, the conclusions reached, and suggestions for further research. The proposed RADAR system in this thesis helps in providing an efficient solution to the problem of zero-day attack detection and analysis. It does so by bridging the gap between attack detection and malware analysis phase. Moreover, RADAR has a layered and modular design which helps it to achieve high performance, flexibility and scalability. Contributions and findings of this thesis are summarized in section 5.1. Section 5.2 presents the future aspects related to the research done.

5.1 Conclusions

Zero-day attacks are among the top security concerns and they provide numerous research challenges to traditional intrusion detection systems. With modern technology they are becoming more stealthier and advanced than ever before and can thus easily evade existing detection systems. Furthermore, the malware authors churn out a large number of new malware variants every day. Analyzing such a high number of malware files manually or semi-automatically is tedious and

time-consuming. So far, the detection of an attack and its analysis are altogether two different phases and are performed at different locations and environments. Thus, creating a big gap between detection and analysis phase. This gap delays the first insight about the malware behavior and signature generation for future containment of the attack. Therefore, to deal with stated issues and to bridge the gap in detection and analysis of zero-day attacks, a real-time network security solution is required that is able to identify zero-day attacks and in parallel analyze the new malware to provide a first comprehensive behavioral report. This thesis work presented a RADAR system capable of detecting zero-day attacks by identifying benign traffic based on important traffic features and creating a baseline to seek unknown deviations. It also implements a stub to analyze zero-day binary in parallel. The stub integrates existing static and dynamic malware analysis functionalities to work as a single autonomous unit.

This research work has been funded by Tata Consultancy Services (TCS) through the TCS Research Scholarship Program for PhD students among India. This thesis contributes in the following ways:

- Done an extensive survey on existing zero-day attack detection techniques specifically to zero-day polymorphic worms. The survey identifies current research challenges and lessons learned in the field of zero-day attack detection.

- The proposed system known as RADAR combines features of existing zero-day attack detection techniques (anomaly detection, signature-based detection and behavior-based detection) that offers better sensitivity and specificity. It has a layered designed where each layer is dedicated to a single

functionality and works in parallel to improve performance.

- RADAR addresses the research problems with existing approaches in zero-day attack detection and analysis by integrating detection and analysis capability in a single framework. Hence, reducing the delay in providing the first insight about the zero-day malware behavior.
- RADAR is able to identify benign traffic based on important traffic features and creates a baseline, which is used to seek deviations to detect zero-day attack. It uses a machine learning algorithm to create the baseline.
- A component-based analysis and reporting stub has been designed and implemented to analyze malware automatically with minimal manual intervention. It integrates the advantages of static, dynamic, and manual analysis (optional) and generates a comprehensive report on zero-day malware behavior in HTML and PDF format.
- A kernel minifilter driver has been developed to monitor system objects during dynamic analysis in a clean and reliable way. The minifilter provides more fine-grained analysis in a stealthy manner and is compatible with all modern versions of Microsoft Windows (2000, XP, Vista, Server, 7 and 8), including 64 bit versions.
- RADAR has been tested with both synthetic dataset and real-time dataset. The synthetic dataset includes various types of known malware that are fabricated to behave as unknown for the system. On the other hand, real-time dataset comprises of unknown malware both obfuscated and non-obfuscated collected from various online malware repositories.

- The evaluation of detection results is performed using standard IDS evaluation metrics like TP Rate, FP Rate, Precision, Recall, F-Measure, Accuracy and ROC Area. Results show that the proposed system has achieved high accuracy with near zero-false positives. For assessing analysis results the reports generated by RADAR are compared to the information provided by online virus and behavioral scan engines.

5.1.1 Key Findings

Following are the chapter wise key findings:

In Chapter 1, introduction to contemporary network security with respect to detection of unknown attacks is discussed. Top security and anti-virus companies reported zero-day attacks as certainly the biggest security concern that modern enterprises face today. Zero-day attacks were talked about few years back, but today every industry faces it. Another day, another breach and a company losses sensitive data. This chapter also identifies underlying challenges and necessary prerequisites in the area of zero-day attack detection, which motivates for the development of an efficient security solution.

In Chapter 2, a survey on zero-day attack detection techniques is provided. The survey is done to classify the existing zero-day attack detection techniques into three different broad categories: statistical-based, signature-based, behavior-based and other hybrid techniques. The chapter further discusses some popular malware analysis tools and techniques used to capture the behavior of a malware. At last Honeynet is explored as a dedicated framework for unknown attack detection and data analysis at same site. Problem statement is then formulated to conclude the chapter.

Chapter 3 presented Run-time zero-day Attack Detection Analysis and Reporting (RADAR) system to efficiently detect zero-day attacks and to provide a comprehensive automated malware analysis report. RADAR has a layered designed where each layer is dedicated to a single functionality and works in parallel. RADAR employs 1-class SVM as an anomaly detection technique in detection layer to detect zero-day attacks that diverts from the good traffic baseline. Good network traffic is collected from a trusted subnet which does not generate or participate in any malicious activity. The unknown anomalous packet is parsed to extract the malicious binary for behavioral analysis.

The analysis layer in the RADAR captures both static and dynamic behavior of malicious binary captured in the detection layer. The analysis and reporting stub of analysis layer integrates both existing static and dynamic malware analysis functionalities and utilities to work together as a single unit in a component based architecture. This provides a freedom to replace any of the function or utility in the future. The Static Analysis Engine (SAE) provides the basic static information to profile the malicious binary. The Dynamic Analysis Engine (DAE) captures run-time behavior of the binary. It also has the capability to evade anti-emulation and anti-debugging checks of a malicious binary which may hinder the malware analysis process. Manual analysis is also intromitted to do step by step analysis of binary if needed. RADAR reports zero-day malware behavior in HTML or PDF format.

Chapter 4 mention details about the experiments and results for validating RADAR. This chapter starts with the listing down of various hardware and software requirements. Experiments are performed to validate the proposed system. RADAR is evaluated against various standard metrics like True Positive Rate (TPR), False

Positive Rate (FPR), F-Measure, Total Accuracy (ACC) and Receiver Operating Characteristic (ROC) curve. Two types of datasets namely synthetic dataset and real-time dataset are used. Synthetic dataset contains fabricated unknown attacks. This is done by applying various obfuscation engines and techniques to mutate known shellcodes, to act as zero-day for the system. On the other hand, real-time dataset contains real traffic captured at the TU network. RADAR is tested for both obfuscated and non-obfuscated zero-day attacks from the datasets obtained.

To summarize, this thesis has laid the foundation for the development of an efficient zero-day attack detection and analysis system known as RADAR. It identifies benign traffic based on important traffic features and creates a SVM baseline to seek deviations to detect zero-day attacks. RADAR also bridges the gap between attack detection phase and analysis phase to provide early comprehensive behavioral report for zero-day attacks. With these novel contributions, this thesis opens up opportunities for future research in relation to real-time integrated zero-day attack detection and analysis.

5.2 Future Work

This thesis advances the state-of-the-art in area of zero-day attack detection and analysis through its contributions. The investigations conducted in this thesis reveal several areas where more work needs to be done. Moreover, the contributions of this thesis have led to new challenges that should be addressed through further research. This section briefly describes some of these challenges within the scope of the thesis. In the future work it is planned to include following research directions:

For Zero-day Attack Detection:

- The primary goal of future work will be to further reduce false positive rate to allow efficient zero-day attack detection. For this more relevant network traffic features must be extracted to generate good traffic profile.
- The system can be enhanced to generate detailed signatures for zero-day obfuscated binaries in Snort format to auto-detect the same attack in future.
- Extending zero-day protection to Internet of Things (IoT). IoT is a network of smart objects and different types of devices that communicate via Internet Protocol [MAC+15]. This future outlook provides new attack vectors against IoT devices and induce serious security and privacy issues [Cha13]. One way of defense can be locking of the critical embedded system in an IoT device after detection of zero-day malware.

For Zero-day Malware Analysis:

- The entire analysis process can be fully automated to extract more behavioral information about zero-day binaries without any sort of human intervention.
- Binary fuzzing can be utilized to explore multi-execution paths of a binary. Black box fuzzing explores new paths by blindly guessing new inputs. It is where nothing is known about the program or its input with a goal that the program should not hang or crash. Black box fuzzing generate input from

scratch, mutate it n-times, generate new input. Mutations include bit-flips, arithmetic operations and other standard stuff.

- Current kernel-based analysis techniques have limitation against Advanced Persistent Threats (APTs). They especially targets the solution proposed for behavioral analysis. It is difficult to analyse attacks like:
  - Zero-days rootkits enlarging its privileges to kernel mode to hide its objects.
  - Session based malware that is executed on the fly without using well known API-functions to load malicious code.
  - Reflective DLL injection: Attacker does not use any of the Operating System featured functions to load executable code into memory (like LoadLibrary, Exec, CreateProcess, etc.)
  - Transient malware in combination with scripting hosts
- The framework can be improved to increase performance, scalability and throughput by analyzing multiple zero-day binaries at one time. This can be easily achieved by performing automatic malware analysis using Cloud Computing.

References

[AAM05] P. Akritidis, K. Anagnostakis, and E. P. Markatos. *Efficient Content-based Detection of Zero-day Worms*. Proceedings of IEEE International Conference on Communications, Seoul, Korea, pp. 837-843, August 2005. 29

[ACMZ09] S. Almotairi, A. Clark, G. Mohay, and J. Zimmermann. *A Technique for Detecting New Attacks in Low-Interaction Honeypot Traffic*. Proceedings of IEEE 4th International Conference on Internet Monitoring and Protection, Washington DC, USA, pp. 7-13, May 2009. 23

[AJA11] S. H. Ahmadinejad, S. Jalili, and M. Abadi. *A Hybrid Model for Correlating Alerts of Known and Unknown Attack Scenarios and Updating Attack Graphs*. Computer Networks, 55(9):22212240, June 2011. 25

[AK12] A. AlEroud and G. Karabatis. *A Contextual Anomaly Detection Approach to Discover Zero-Day Attacks*. Proceedings of the IEEE International Conference on Cyber Security (CYBERSECURITY '12), Washington, DC, pp. 40-45, December 2012. 24

[AK13] A. AlEroud and G. Karabatis. *Toward Zero-Day Attack Identification Using Linear Data Transformation Techniques.* Proceedings of the 7th IEEE International Conference on Software Security and Reliability (SERE'13), Gaithersburg, MD, pp. 159-168, June 2013. 24

[AK14] A. AlEroud and G. Karabatis. *Detecting Zero-Day Attacks Using Contextual Relations.* Proceedings of the LNBIP Springer 9th International Conference on Knowledge Management in Organizations (KMO'14), Santiago, Chile, pp. 373-385, August 2014. 24

[Ald13] Aldeid. *PEid Online Manual.* http://www.aldeid.com/wiki/PEiD, 2013. 86

[AVWA11] M. Alazab, S. Venkatraman, P. Watters, and M. Alazab. *Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures.* Proceedings of the 9th Australasian Data Mining Conference (AusDM'11), Ballarat, Australia, pp. 171-182, December 2011. 13, 24

[B+04] R. Bajcsy et al. *Cyber Defense Technology Networking and Evaluation.* Communications of the ACM, 47(3):58-61, March 2004. 22

[B+09] U. Bayer et al. *Scalable, Behavior-based Malware Clustering.* Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS'09), San Diego, California, USA,, February 2009. 58

[Ban05] P. Bania. *TAPiON Polymorphic Decryptor Generator*. http://pb.specialised.info/all/tapion/, September 2005. 115

[BCT11] H. Bos, L. Cavallaro, and A. S. Tanenbaum. *Systems Security at VU University Amsterdam*. Proceedings of the First SysSec Workshop (SysSec), Amsterdam, pp. 111-114, July 2011. 14

[BD12] L. Bilge and T. Dumitras. *Before We Knew It: An Empirical Study of Zero-day Attacks in the Real World*. Proceedings of the ACM Conference on Computer and Communications Security (CCS'12), New Carolina, pp. 833-844, October 2012. 7, 13

[Bel05] F. Bellard. *QEMU, a fast and portable dynamic translator*. Proceedings of the Annual Technical Conference on USENIX, Anaheim, CA, USA, pp. 41-46, April 2005. 57

[BMKK06] U. Bayer, A. Moser, C. Kruegel, and E. Kirda. *Dynamic Analysis of Malicious Code*. Journal in Computer Virology, 2(1):67-77, May 2006. 55

[BNS+06] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha. *Towards Automatic Generation of Vulnerability-based Signatures*. Proceedings of the IEEE Security Privacy Symposium, Oakland, CA, pp. 2-16, May 2006. 19, 35

[BNS07] D. Brumley, J. Newsome, and D. Song. *Sting: An End-to-End Self-Healing System for Defending against Internet Worms*. Springer Science+Business Media LLC, Vol. 27, US, pp. 147-170, 2007. 35

[Boc15] Bochs. *Bochs: The open source IA-32 emulation project.* http://bochs.sourceforge.net/, January 2015. 55

[BS12] B. Bayoglu and I. Sogukpinar. *Graph based Signature Classes for Detecting Polymorphic Worms via Content Analysis.* Computer Networks: The International Journal of Computer and Telecommunications Networking, 56(2):832844, February 2012. 31

[C+05] M. Costa et al. *Vigilante: End-to-end Containment of Internet Worms.* Proceedings of the ACM 20th Symposium on Operating Systems Principles (SOSP'05), Brighton, UK, pp. 133-147, October 2005. 19, 35

[Cha13] H. Chaouchi. *The Internet of Things: Connecting Objects.* John Wiley & Sons, February 2013. 138

[Che07] S. Cheetancheri. *Collaborative defense against zero-day and polymorphic worms: detection, response and an evaluation framework.* University of California, Davis, August 2007. http://www.cs.ucdavis.edu/research/tech-reports/2007/CSE-2007-38.pdf. 22

[CJ03] M. Christodorescu and S. Jha. *Static Analysis of Executables to Detect Malicious Patterns.* Proceedings of 12th USENIX Security Symposium, CA, USA, pp. 1-12, August 2003. 50

[CJS+05] M. Christodorescu, S. Jha, S. A. Seshia, D. X. Song, and R. E. Bryant. *Semantics Aware Malware Detection.* Proceedings of IEEE

Symposium on Security and Privacy, CA, USA, pp. 3246, May 2005. 50

[CLMM08] L. Cavallaro, A. Lanzi, L. Mayer, and M. Monga. *LISABETH: Automated Content-based Signature Generator for Zero-day Polymorphic Worms*. Proceedings of the ACM 4th International Workshop on Software Engineering for Secure Systems, Leipzig, Germany, pp. 41-48, May 2008. 30

[CLS+13] P. M. Comar, L. Liu, S. Saha, P. N. Tan, and A. Nucci. *Combining supervised and unsupervised learning for zero-day malware detection*. Proceedings of the IEEE INFOCOM'13, Turin, pp. 2022-2030, April 2013. 25

[CSW05] J. R. Crandall, Z. Su, and S. F. Wu. *On Deriving Unknown Vulnerabilities from Zero-day Polymorphic and Metamorphic Worm Exploits*. Proceedings of the ACM 12th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, pp. 235-248, November 2005. 19

[Cuc13] Cuckoo. *Cuckoo Sandbox - Open Source Automated Malware Analysis*. https://media.blackhat.com/us-13/US-13-Bremer-Mo-Malware-Mo-Problems-Cuckoo-Sandbox-WP.pdf, August 2013. 58

[Dea12] T. Dube and et. al. *Malware Target Recognition via Static Heuristics*. Computers & Security, 31(1):137-147, February 2012. 50

[DUMU03] T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk. *Polymorphic Shellcode Engine Using Spectrum Analysis*. Phrack, vol. 11, no. 61, August 2003. 115

[Eag11] C. Eagle. *THE IDA PRO BOOK: The Unofficial Guide to the World's Most Popular Disassembler*. 2nd ed. No Starch Press, San Francisco, May 2011. 10, 49

[EH12] M. Eskandari and S. Hashemi. *A Graph Mining Approach for Detecting Unknown Malware*. Journal of Visual Languages and Computing, 23(1):154-162, March 2012. 51

[ESKK12] M. Egele, T. Scholte, E. Kirda, and C. Kruegel. *A Survey on Automated Dynamic Malware Analysis Techniques and Tools*. ACM Computing Surveys (CSUR), 44(2):1-49, February 2012. 49, 53, 58, 59

[Fla04] H. Flake. *Structural Comparison of Executable Objects*. Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'04), pp. 1-13, July 2004. 50

[FMC11] N. Falliere, L. O. Murchu, and E. Chien. *W32.Stuxnet Dossier*. http://www.h4ckr.us/library/Documents/ICS_Events/Stuxnet%20Dossier%20(Symantec)%20v1.4.pdf, February 2011. 6

[Fra14] Metasploit Framework. *Metasploit Penetration Testing Software*. http://www.metasploit.com/, April 2014. 115

[GCT13] C. Giuffrida, L. Cavallaro, and A. S. Tanenbaum. *Practical automated vulnerability monitoring using program state invariants*. Proceedings of the IEEE/IFIP 43rd Annual International Conference on Dependable Systems and Networks (DSN'13), Budapest, Hungary, pp. 1-12, June 2013. 14

[GSK10] S. Gupta, G. Sikka, and V. Katiyar. *Elliptic Curve Cryptography in Pervasive Computing*. International Conference on Biomedical Engineering and Assistive Technologies (BEATs-2010), pp. 1-6, December 2010. 115

[GW12] R. Goyal, S. Sharma , S. Bevinakoppa , and P. Watters. *Obfuscation of Stuxnet and Flame Malware*. Proceedings of 3rd International Conference on Applied Informatics and Computing Theory, Barcelona, pp. 150-154, October 2012. 6

[Hea14] VX Heavens. *VX Heavens Site*. http://vxheaven.org/, 2014. 116

[Hol09] T. Holz. *Tracking and Mitigation of Malicious Remote Control Networks*. PhD thesis, University of Mannheim, April 2009. 51

[HZXF14] Y. Hou, J.W. Zhuge, D. Xin, and W. Feng. *SBE - A Precise Shellcode Detection Engine Based on Emulation and Support Vector Machine*. Proceedings of the LNCS, Springer 10th International Conference on Information Security Practice and Experience (ISPEC'14), Fuzhou, China, pp. 159-171, May 2014. 37

[Inc14] FireEye Inc. *Less Than Zero: A Survey of Zero-day Attacks in 2013 and What They Say About the Traditional Security Model*.

https://www.fireeye.com/content/dam/legacy/resources/pdfs/fireeye-zero-day-attacks-in-2013.pdf, May 2014. 6

[Jac03] J. E. Jackson. *A User's Guide to Principal Components*. 1st ed. Wiley-Interscience, New York, September 2003. 24

[Joe14] Joesecurity. *Joe Sandbox Technology*. http://www.joesecurity.org/joe-sandbox-technology, February 2014. 59

[Jol02] I. T. Jolliffe. *Principal Component Analysis*. 2nd ed., ser. Springer Series in Statistics, Springer, New York, October 2002. 24

[JS11] P. Jain and A. Sardana. *A Hybrid Honeyfarm based Technique for Defense against Worm Attacks*. Proceedings World Congress on Information and Communication Technologies (WICT'11), Mumbai, India, pp. 1084-1089, December 2011. 39

[JS12] P. Jain and A. Sardana. *Defending Against Internet Worms using Honeyfarm*. Proceedings of the ACM CUBE International Information Technology Conference (CUBE'12), Pune, India, pp. 795-800, December 2012. 39

[JTA08] A. Jumratjaroenvanit and Y. Teng-Amnuay. *Probability of Attack based on System Vulnerability Life Cycle*. Proceedings of the IEEE International Symposium on Electronic Commerce and Security, Washington DC, pp. 531-535, August 2008. 8

[K+09a] I. Kim et al. *A Case Study of Unknown Attack Detection against Zero-day Worm in the Honeynet Environment*. Proceedings of the

IEEE 11th International Conference on Advanced Communication Technology (ICACT' 2009), Phoenix Park, pp. 1715-1720, April 2009. 31

[K+09b] I. Kim et al. *Validation Methods of Suspicious Network Flows for Unknown Attack Detection*. International Journal of Computers, 3(1):104-114, December 2009. 31

[K214] K2. *ADMmutate*. http://www.ktwo.ca/ADMmutate-0.8.4.tar.gz, August 2014. 115

[KK05] H. Kim and B. Karp. *Autograph: Toward Automated, Distributed Worm Signature Detection*. Proceedings of 13th USENIX Security Symposium, Berkeley, CA, USA, pp. 19-35, August 2005. 28

[KKB06] C. Kruegel, E. Kirda, and U. Bayer. *Ttanalyze: A Tool for Analyzing Malware*. Proceedings of the 15th Annual Conference European Institute for Computer Antivirus Research (EICAR'06), Hamburg, Germany, April 2006. 57

[KKM+05] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. *Polymorphic Worm Detection using Structural Information of Executables*. Proceedings of LNCS 8th International Symposium on Recent Advances in Intrusion Detection (RAID'05), Seattle, pp. 207-227, September 2005. 34

[KLK14] G. Kim, S. Lee, and S. Kim. *A Novel Hybrid Intrusion Detection Method Integrating Anomaly Detection with Misuse Detection*. Expert Systems with Applications, 41(4):16901700, March 2014. 25

[KM07] K. Kendall and C. McMillan. *Practical Malware Analysis.* Black Hat USA - Briefings and Training, Las Vegas, USA, August 2007. 47

[KS14a] R. Kaur and M. Singh. *Automatic Evaluation and Signature Generation Technique for Thwarting Zero-Day Attacks.* Proceedings of the CCIS Springer 2nd International Conference on Security in Computer Networks and Distributed Systems (SNDS'14), Trivandrum, India, pp. 298-309, March 2014. 66

[KS14b] R. Kaur and M. Singh. *A Survey on Zero-Day Polymorphic Worm Detection Techniques.* IEEE Communications Surveys & Tutorials, 16(3):1520-1549, March 2014. 18

[KS15] R. Kaur and M. Singh. *Two-Level Automated Approach for Defending Against Obfuscated Zero-day Attacks.* Proceedings of the LNCS Springer 9th International Conference on Risks and Security of Internet and Systems (CRiSIS'14), Trento, Italy, April 2015. 66

[Kul14] N. Kulwin. *Sony Hack.* http://recode.net/2014/12/23/sony-hack-what-you-missed/, December 2014. 7

[LD07] C. Leita and M. Dacier. *SGNET: A Distributed Infrastructure to Handle Zero-day Exploits.* Technical Report EURECOM+2164, EURECOM institute, France, February 2007. 37

[LD08] C. Leita and M. Dacier. *SGNET: Implementation Insights.* Proceedings of the IEEE Network Operations and Management Symposium, Salvador, Bahia, pp. 1075-1078, April 2008. 37

[Lel10] A. Lelli. *The Trojan. Hydraq incident: Analysis of the Aurora 0-day exploit.* http://www.symantec.com/connect/blogs/trojanhydraq-incident-analysis-aurora-0-day-exploit, January 2010. 6

[Lip13] H. Lipson. *Average Intruder Knowledge and Attack Sophistication as a Function of Time.* Carnegie Mellon University (CMU) Software Engineering Institute CERT, February 2013. xii, 3

[LS05] Z. Liang and R. Sekar. *Fast and Automated Generation of Attack Signatures: A Basis for Building Self-protecting Servers.* Proceedings of the ACM 12th ACM Conference on Computer and Communications Security, Alexandria, VA, USA, pp. 213-222, November 2005. 19, 35

[LSC+06] Z. Li, M. Sanghi, Y. Chen, M.Y. Kao, and B. Chavez. *Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience.* Proceedings of the IEEE Symposium on Security and Privacy (S&P'06), Berkeley/Oakland, CA, pp. 15-47, June 2006. 19, 29, 30

[LWZ+13] H. Lu, X. Wang, B. Zhao, F. Wang, and J. Su. *ENDMal: An anti-obfuscation and collaborative malware detection system using syscall sequences.* Mathematical and Computer Modelling, 58(5-6):11401154, September 2013. 38

[MAC+15] Ibrahim Mashal, Osama Alsaryrah, Tein-Yaw Chung, Cheng-Zen Yang, Wen-Hsing Kuo, and Dharma P. Agrawal. *Choices for In-*

teraction with Things on Internet and Underlying Issues. Ad Hoc Networks 28, pp. 6890, January 2015. 138

[Mar11] J. Markoff. *SecurID Company Suffers a Breach of Data Security.* http://www.nytimes.com/2011/03/18/technology/18secure.html, March 2011. 6

[MCV08] M. M. Z. E. Mohammed, H. A. Chan, and N. Ventura. *Honeycyber: Automated Signature Generation for Zero-day Polymorphic Worms.* Proceedings of the IEEE Military Communications Conference (MILCOM' 2008), San Diego, CA, pp. 1-8, November 2008. 30

[MCV+10] M. M. Z. E. Mohammed, H. A. Chan, N. Ventura, M. Hashim, I. Amin, and E. Bashier. *Detection of Zero-day Polymorphic Worms using Principal Component Analysis.* Proceedings of the IEEE 6th International Conference on Networking and Services, Cancun, pp. 277-281, March 2010. 31

[MKK07] A. Moser, C. Kruegel, and E. Kirda. *Limits of Static Analysis for Malware Detection.* Proceedings of IEEE 23rd Annual Computer Security Applications Conference, Florida, pp. 421-430, December 2007. 50

[MP13] M. Mohammed and A. S. K. Pathan. *Automatic Defense Against Zero-day Polymorphic Worms in Communication Networks.* CRC Press, Taylor & Francis Group, June 2013. 31

[MPB+13] C. Modi, D. Patel, B. Borisaniya, H. Patel, A. Patel, and M. Rajarajan. *A survey of intrusion detection techniques in Cloud.* Journal of Network and Computer Applications, 36(1):42-57, January 2013. 13

[MW11] P. K. Manadhata and J. M. Wing. *An Attack Surface Metric.* IEEE Transactions on Software Engineering, 37(3):371-386, June 2011. 9

[N+09] A.M. Nguyen et al. *MAVMM: Lightweight and Purpose Built VMM for Malware Analysis.* Proceedings of the Annual Conference on Computer Security Applications, Honolulu, HI, pp. 441-450, December 2009. 55

[NKS05] J. Newsome, B. Karp, and D. Song. *Polygraph: Automatically Generating Signatures for Polymorphic Worms.* Proceedings of the IEEE Symposium on Security and Privacy (S&P'05), Oakland, CA, pp. 226-241, May 2005. 28, 30

[NMT12] E. Nakashima, G. Miller, and J. Tate. *The Washington Post.* http://cyber-peace.org/wp-content/uploads/2013/06/U.S.pdf, June 2012. 6

[Nor14] Norman. *Norman Sandbox - Your Proactive IT Security Tool.* http://download01.norman.no/product_sheets/eng/SandBox_analyzer.pdf, February 2014. 58

[NS05] J. Newsome and D. Song. *Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Com-*

modity Software. Proceedings of the 12th Annual Network and Distributed System Security Symposium (NDSS'05), San Diego, California, pp. 1-17, February 2005. 19

[Oll13] OllyDbg. *The OllyDbg Debugger*. http://www.ollydbg.de/, 2013. 103

[PAM06] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. *Network-level Polymorphic Shellcode Detection using Emulation*. Journal in Computer Virology, 2(4):257-274, July 2006. 37

[PAM07] M. Polychronakis, K. G. Anagnostakis, and E. P. Markatos. *Emulation-based Detection of Non-self-contained Polymorphic Shellcode*. Proceedings of the LNCS Springer 10th International Conference on Recent Advances in Intrusion Detection (RAID'07), Gold Goast, Australia, pp. 87-106, September 2007. 37

[Par13] K. Parrish. *Facebook Hacked by Zero-Day Java Exploit*. http://www.tomsguide.com/us/Facebook-Twitter-zero-day-Oracle-Exploit,news-16786.html, February 2013. 7

[Pas12] P. Passeri. *A 0-Day Attack Lasts On Average 10 Months*. http://hackmageddon.com/2012/10/19/a-0-day-attack-lasts-on-average-10-months/, October 2012. xii, 8

[Pas15] P. Passeri. *2014 Cyber Attacks Statistics (Aggre-*

gated). http://hackmageddon.com/category/security/cyber-attacks-statistics/, January 2015. xii, 2

[PB07] G. Portokalidis and H. Bos. *SweetBait: Zero-hour Worm Detection and Containment using Low-and High-Interaction Honeypots.* Computer Networks: The International Journal of Computer and Telecommunications Networking, 51(5):1256-1274, April 2007. 30

[PCLW04] A. Pasupulati, J. Coit, K. Levitt, and F. Wu. *Buttercup: On Network-based Detection of Polymorphic Buffer Overflow Vulnerabilities.* Proceedings of IEEE Network Operation and Management Symposium, Seoul, Korea, pp. 235-248, May 2004. 34

[Pla99] J. C. Platt. *Fast training of support vector machines using sequential minimal optimization.* In Advances in Kernel Methods - Support Vector Learning, MIT Press, January 1999. 23

[Plu10] Anti-Anti-Debugger Plugins. *Google project.* https://code.google.com/p/aadp/, 2010. 103

[Pro03] The Honeynet Project. *Know your enemy: Sebek - A kernel based data capture tool.* http://old.honeynet.org/papers/sebek.pdf, November 2003. 60

[Pro05] The Honeynet Project. *Know your enemy: Honeywall Cdrom Roo.* http://old.honeynet.org/papers/cdrom/roo/, August 2005. 60

[Pro06] The Honeynet Project. *Know your enemy: Honeynets.* http://old.honeynet.org/papers/honeynet/index.html, May 2006. 59

[Rab81] M. Rabin. *Fingerprinting by Random Polynomials.* Harvard Aiken Computation Laboratory, Harvard University, TR-15-81, 1981. 29

[Ras12] F. Y. Rashid. *How To Detect Zero-Day Malware And Limit Its Impact.* http://www.darkreading.com/attacks-breaches/how-to-detect-zero-day-malware-and-limit/240062798, November 2012. 13

[Rie15] F. Rienhardt. *Kernel-based monitoring on Windows (32/64 bit).* http://www.bitnuts.de/KernelBasedMonitoring.pdf, February 2015. 92

[Roe99] M. Roesch. *Snort Lightweight Intrusion Detection for Networks.* Proceedings of the 13th Systems Administration Conference USENIX LISA'99, Seattle, Washington, USA, November 1999. 72

[SABS12] M. Szmit, S. Adamus, S. Bugala, and A. Szmit. *Anomaly Detection 3.0 for Snort.* SECURITATEA INFORMATIONALA, pp. 37-41, June 2012. 75

[SBGS15] S. Singla, D. Bansal, E. Gandotra, and S. Sofat. *A Novel Approach to Malware Detection using Static Classification.* International Journal of Computer Science and Information Security (IJCSIS), 13(3), pp. 1-5, March 2015. 51

[SBUPB13] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas. *Opcode sequences as representation of executables for data-mining-based unknown malware detection.* Information Sciences, 231:6482, May 2013. 26

[SC09] W. C. Sun and Y. M. Chen. *A Rough Set Approach for Automatic Key Attributes Identification of Zero-day Polymorphic Worms*. Expert Systems with Applications: An International Journal, 36(3):4672-4679, April 2009. 23

[SC13] N. Srivastav and R. K. Challa. *Novel Intrusion Detection System Integrating Layered Framework with Neural Network*. IEEE 3rd International Conference Advance Computing Conference (IACC), Ghaziabad, pp. 682-689, February 2013. 17

[Sea08] M. Sharif and et. al. *Eureka: A Framework for Enabling Static Malware Analysis*. Proceedings of 13th European Symposium on Research in Computer Security, Spain, pp. 481-500, October 2008. 50

[SEVS04] S. Singh, C. Estan, G. Varghese, and S. Savage. *Automated Worm Fingerprinting*. Proceedings of 6th USENIX Conference on Symposium on Operating Systems Design and Implementation, Berkeley, CA, USA, pp. 4-20, 2004. 28

[SH13] M. Sikorski and A. Honig. *Practical Malware Analysis - The Hands-On Guide to Dissecting Malicious Software*. No Starch Press; 1 edition, February 2013. 46

[SHT+07] J. Song, O. Hayato, H. Takakura, Y. Okabe, K. Ohira, and Y. Kwon. *A Comprehensive Approach to Detect Unknown Attacks via Intrusion Detection Alerts*. Proceedings of LNCS Springer Focusing on

Computer and Network Security, Doha Qatar, pp. 247-253, December 2007. 23

[Sop14] Sophos. *Security Threat Report: Smarter, Shadier, Stealthier Malware*. http://www.sophos.com/en-us/medialibrary/PDFs/other/sophos-security-threat-report-2014.pdf, December 2014. 6

[SPST+01a] B. Schlkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. *Estimating the support of a high-dimensional distribution*. Neural Computation, 13(7):1443-1471, July 2001. 78

[SPST+01b] B. Schoelkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson. *Estimating the Support of a High-dimensional Distribution*. Neural Computation, 13(7):1443-1471, July 2001. 23

[SSS03] G. Varghese S. Singh, C. Estan and S. Savage. *The Earlybird System for Realtime Detection of Unknown Worms*. Technical Report CS20030761, Deptartment of Computer Science, University of California, San Diego, 2003. 28

[Sta15] B. Stagnaro. *Survey Says Zero-Day Attacks and Evasive Malware are Biggest Risks*. http://researchcenter.paloaltonetworks.com/2015/04/survey-says-zero-day-attacks-and-evasive-malware-are-biggest-risks/, April 2015. xii, 5

[STK08] J. Song, H. Takakura, and Y. Kwon. *A Generalized Feature Extraction Scheme to Detect 0-day Attacks via IDS Alerts*. Proceedings of

IEEE International Symposium on Applications and the Internet, Washington, DC, USA, pp. 55-61, August 2008. 22

[STO08] J. Song, H. Takakura, and Y. Okabe. *Cooperation of Intelligent Honeypots to Detect Unknown Malicious Codes.* Proceedings of IEEE Workshop on Information Security Threats Data Collection and Sharing (WOMBAT'08), Amsterdam, pp. 31-39, September 2008. 39

[STOK09] J. Song, H. Takakura, Y. Okabe, and Y. Kwon. *Unsupervised Anomaly Detection Based on Clustering and Multiple One-class SVM.* IEICE Transactions on Communications, E92-B(06):19811990, June 2009. 25

[STON13] J. Song, H. Takakura, Y. Okabe, and K. Nakao. *Toward a More Practical Unsupervised Anomaly Detection System.* Information Sciences, 231:4-14, May 2013. 25

[SW00] N. T. Spring and D. Wetherall. *A Protocol-Independent Technique for Eliminating Redundant Network Traffic.* Proceedings of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Stockholm, Sweden, pp. 87-95, August 2000. 29

[Sym11] Symantec. *W32.Duqu The precursor to the next Stuxnet.* http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_duqu_the_precursor_to_the_next_stuxnet.pdf, November 2011. 6

[Sym14] Symantec. *Internet Security Threat Report of 2014*. http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf, April 2014. 6

[TC07] Y. Tang and S. Chen. *An Automated Signature-based Approach against Polymorphic Internet Worms*. IEEE Transactions on Parallel and Distributed Systems, 18(7):879-892, July 2007. 33

[TSPM11] Z. Tzermias, G. Sykiotakis, M. Polychronakis, and E. P. Markatos. *Combining static and dynamic analysis for the detection of malicious documents*. Proceedings of the 4th European Workshop on System Security, Austria, pp. 41-46, April 2011. 14

[TXL09] Y. Tang, B. Xiao, and X. Lu. *Using a Bioinformatics Approach to Generate Accurate Exploit-based Signatures for Polymorphic Worms*. Journal of Computers and Security, 28(8):827-842, November 2009. 31

[TXZ09] C. Ting, Z. Xiaosong, and L. Zhi. *A Hybrid Detection Approach for Zero-day Polymorphic Shellcodes*. Proceedings of the IEEE International Conference on E-Business and Information System Security, Wuhan, pp. 1-5, May 2009. 39

[Vap98] V. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, New York, 1998. 78

[Vap99] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, USA, 1999. 78

[Vir14] VirusTotal. *VirusTotal Community*. https://www.virustotal.com/, 2014. 84

[W+02] B. White et al. *An Integrated Experimental Environment for Distributed Systems and Networks*. Proceedings of USENIX 5th Symposium on Operating Systems Design and Implementation, Boston, MA, pp. 255-270, December 2002. 22

[W+06] X. Wang et al. *Packet Vaccine: Black-box Exploit Detection and Signature Generation*. Proceedings of the ACM 12th ACM 13th ACM Conference on Computer and Communications Security (CCS'06), Alexandria, VA, USA, pp. 37-46, November 2006. 19, 35

[WCS06] D. Wang, G. Cretu, and S. J. Stolfo. *Anomalous Payload-based Worm Detection and Signature Generation*. Proceedings of LNCS 8th International Symposium on Recent Advances in Intrusion Detection (RAID'06), Heidelberg, pp. 227-246, September 2006. 32

[WHF07] C. Willems, T. Holz, and F. Freiling. *Toward Automated Dynamic Malware Analysis Using CWSandbox*. Journal IEEE Security and Privacy, 5(2):32-39, March 2007. 58

[WHKM06] J. Wang, L. Hamadeh, G. Kesidis, and D. J. Miller. *Polymorphic Worm Detection and Defense: System Design, Experimental Methodology, and Data Resources*. Proceedings of ACM SIGCOMM workshop on Large-scale Attack Defense, New York, pp. 169-176, September 2006. 29

[WLC+10] L. Wang, Z. Li, Y. Chen, Z. Fu, and X. Li. *Thwarting Zero-day Polymorphic Worms with Network-level Length-based Signature Generation*. IEEE/ACM Transactions on Networking (TON), 18(1):53-66, February 2010. 35

[WLZ07] W. Wang, D. S. Luo, and J. Zhang. *Detect Polymorphic Worms based on Semantic Signature and Data Mining*. Proceedings of IEEE 1st Communications and Networking Conference, China, pp. 1-4, April 2007. 34

[WMK06] J. Wang, D. J. Miller, and G. Kesidis. *Efficient Mining of the Multidimensional Traffic Cluster Hierarchy for Digesting, Visualization, and Anomaly Identification*. IEEE Journal on Selected Areas in Communications (JSAC), 24(10):1929-1941, October 2006. 29

[WS04] K. Wang and S. J. Stolfo. *Anomalous Payload-based Network Intrusion Detection*. Proceedings of LNCS Springer 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04), Sophia Antipolis, France, pp. 203-222, September 2004. 32

[Y+07] H. Yin et al. *Panorama: Capturing systemwide information flow for malware detection and analysis*. Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07), Alexandria, VA, USA, pp. 116-127, October 2007. 54

[Zel10] L. Zeltser. *Introduction to Malware Analysis*. SANS Institute, 2010. http://zeltser.com/reverse-malware/intro-to-malware-analysis.pdf. 46

[ZH14] M. Zolotukhin and T. Hamalainen. *Detection of Zero-day Malware based on the Analysis of Opcode Sequences.* Proceedings of the IEEE 11th International Conference on Consumer Communications and Networking Conference (CCNC'14), Las Vegas, Nevada, USA, pp. 386-391, January 2014. 38

[ZW14] F. Zhu and J. Wei. *Static Analysis based Invariant Detection for Commodity Operating Systems.* Computers & Security, 43(1):49-63, June 2014. 50

Publications

1. Ratinder Kaur and Maninder Singh, A Survey on Zero-Day Polymorphic Worm Detection Techniques, IEEE Communications Surveys & Tutorials, Vol. 16, no. 3, March 2014, pp. 1520-1549. [SCI-indexed, Impact factor: 6.490]
2. Ratinder Kaur and Maninder Singh, A Hybrid Real-time Zero-day Attack Detection and Analysis System, IJCNIS, Vol. 7, No. 9, August 2015, pp. 19-31.
3. Ratinder Kaur and Maninder Singh, Hybrid Real-time Zero-day Malware Analysis and Reporting System, International Journal of Information Technology and Computer Science (IJITCS) [Accepted].
4. Ratinder Kaur and Maninder Singh, Two-Level Automated Approach for Defending Against Obfuscated Zero-day Attacks, 9th International Conference on Risks and Security of Internet and Systems (CRiSIS), LNCS Springer, Trento, Italy, April 2015.
5. Ratinder Kaur and Maninder Singh, Automatic Evaluation and Signature Generation Technique for Thwarting Zero-day Attacks, Second International Conference on Security in Computer Networks and Distributed Systems (SNDS), CCIS Springer, Kerala,

March 2014.

6. Ratinder Kaur and Maninder Singh, Efficient Hybrid Technique for Detecting Zero-Day Polymorphic Worms, 4th IEEE International Advance Computing Conference (IACC), IEEE Xplore, Gurgaon, February 2014.

7. Ratinder Kaur and Maninder Singh, Efficient Zero-day Polymorphic Worm Detection Technique, Grace Hopper Celebration of Women in Computing (GHC), Baltimore, USA, October 2012. [Poster]

8. Ratinder Kaur and Maninder Singh, Efficient Zero-day Attack Detection Technique, TCS Technical Architect's Global Conference (TACTiCS), New Delhi, India, May 2012. [Poster]