A Network Security Model
for
Attack Signature Generation, Tracking and Analysis

A Thesis submitted

In the fulfillment of the requirements for the award of degree of

DOCTOR of PHILOSOPHY

by

Sanmeet Kaur

(950803011)

Under the supervision of

Dr. Maninder Singh

Associate Professor

Computer Science and Engineering Department

Thapar University, Patiala

Computer Science and Engineering Department

Thapar University, Patiala-147004

May, 2015

Dedicated to

Two little angels of my life Rahat and Rishan

Certificate

I hereby certify that the work which is being submitted in this thesis entitled "A Network Security Model for Attack Signature Generation, Tracking and Analysis", in fulfillment of the requirements for the award of degree of DOCTOR OF PHILOSOPHY submitted in Department of Computer Science and Engineering, Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Maninder Singh and refers work of other researchers 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.

Sanmeet Kaur

(Sanmeet Kaur)
Regd. No. 950803011

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

(Maninder Singh)
Associate Professor,
Department of Computer Science and Engineering,
Thapar University, Patiala-147004 (INDIA)
Supervisor

Acknowledgements

First and foremost I want to thank almighty, thy super power, to provide me this opportunity to gain and share knowledge.

I feel blessed to pay my most valuable thanks to Dr. Maninder Singh, who not only has been a perfect supervisor, but also a great soul, a motivator, a true mentor and a positive inspiration. His faith in me has proved to be the biggest support in carrying out this work.

I am thankful to Thapar University for provisioning of opportunity to pursue PhD along with my job responsibilities. I pay my heartiest thanks to School of Mathematics and Computer Applications for providing me support in terms of resources, time and flexibility to accomplish my research work. I like to thank Dr. S. S. Bhatia, Dr. R.K. Sharma, Dr. Rajesh Kumar, my colleagues and students of School for their valuable support.

I owe my gratitude towards Department of Computer Science and Engineering for giving me this opportunity to accomplish my research. I am thankful to doctoral committee members and other faculty members of CSED for providing me valuable suggestions to make this research journey smooth. My special thanks to Dr. Seema Bawa who has always been a big motivator for me.

I am thankful to the anonymous reviewers for their useful remarks and the authors whose work has been referred for providing a research direction.

I am thankful to my parents, who always feel proud of their daughter. Their teachings, support, love and blessings are priceless. I pay my regards and thanks to my father S. Dalip Singh, mother Mrs. Joginder Kaur, my father-in-law Mr. Ved Bhatia and my mother-in-law Mrs. Jagdish Bhatia for their uncountable blessings. I am thankful to my

Didi and Veera for their love and affection. My regards and respect for Biji and Papa Ji. I am also thankful to my friends, especially Dr. Anupam Sharma for providing helpful and cheerful environment always.

My heartiest thank to my husband Dr. Parteek Bhatia, biggest inspiration of my life, who has always been with me to discuss technical as well as strategic details of this work. Very special thanks to my little kids who have sacrificed some of their priceless moments for the sake of mamma's research. My all love to you Rahat and Rishan. God Bless you.

Finally, it gives me an immense pleasure and feeling of gratitude while acknowledging every individual who has contributed by means of technical, moral, physical and emotional support towards successful completion of this research work.

Sanmeet Kaur

(Sanmeet Kaur)

Abstract

With the rapid growth of networks and Internet, security has become a big area of concern. The risk to the network resources is increasing day by day with the fast growing trends in attacks and intrusions. Despite current detection measures in place, timely discovery of novel attacks is still a critical issue. Intrusion Detection System (IDS) is a well known network security mechanism. IDS may use anomaly based or misuse based approach for detection of intrusions. Signature based technique is the most popular way of misuse detection. Most of IDS use signature based detection as it has low false alarm rate. But, on the contrary there is a big problem with this technique that it cannot detect unknown attacks whose signatures are not stored in their databases. Signature updating is generally a manual process and is a great overhead. Automated signature generation for Intrusion Detection Systems for proactive security of networks has been an emerging area of research. There are many solutions available in literature as proposed by various researchers. But still there is a need to address this problem as the intensity and sophistication of exploits and attacks are increasing exponentially by each passing day.

After an extensive review of literature about current security solutions and various mechanisms, a need for automatic signature generation of HTTP and SMTP attacks has been identified. The work carried out in this thesis is focused on attack detection and signature generation of unknown attacks. For the accomplishment of proposed objectives, a proactive hybrid framework, HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator), for attack detection and signature generation has been designed, implemented and tested. This thesis has been organized into six chapters. These chapters include Introduction; Review of Literature; HIDESIGN: A Proposed framework for Attack Detection and Signature Generation; Implementation details of HIDESIGN; Results and Discussion; Conclusions and Future Scope.

Introduction of thesis explores several concepts and terms used in the area of security; definition and need of network security; approaches to network security, namely reactive and proactive security; different available security solutions in terms of defense mechanisms such as Firewall, Antivirus, IDS, IPS, VPN and Honeypots. In the proposed system, two mechanisms, namely, IDS and honeypots are used. A detailed description about intrusion detection, Intrusion Detection Systems and their classification based on various factors has been given. A proactive security measure called honeypot, their classification and various available honeypots are elaborated.

In Review of Literature, history and evolution of IDS along with the architectural details of two popular IDSs, namely, *Snort* and *Bro* referred in this research work has been explored. The role of anomaly detection using machine learning, available hybrid systems using misuse and anomaly based approach are discussed in this work. Thereafter, the automatic signature generation and its techniques are briefly presented. In this thesis work, in-depth review of existing signature generation mechanisms is presented by elaborating their working detail. These systems are then summarized and compared on the basis of various important parameters. Some of these systems include *Honeycomb, Autograph, Earlybird, PAYL, COVERS, ARBOR, Polygraph, Nemean, Argos, Hamsa, Hancock, Nebula, Auto-Sign* and *F-Sign.* This chapter concludes by highlighting research motivation, objectives of proposed research, major achievements and contributions of this work.

The details of HIDESIGN for automatic generation and tracking of attack signatures in context with HTTP and SMTP traffic have been discussed. Here, an introduction followed by background and attack vector details of HTTP and SMTP protocols is presented. In this research work, we have focused on server side HTTP and SMTP attacks. Various types of HTTP attacks such as SQL injection, command injection, directory traversal, cross site scripting attacks *etc.* have been discussed in detail. SMTP attacks and their details are also discussed in this chapter. This chapter discusses the overview, detailed architectural design and work flow of proposed system and its components. Major components of HIDESIGN include honeypot server, Misuse

Detection Engine (MDE), Anomaly Detection Engine (ADE) and Signature Generation Engine (SGE). Incoming packet stream from network traffic is captured by honeypot server. Signature based detection is accomplished by MDE to detect and filter out known attacks from this traffic stream. The backbone of ADE is supervised machine learning. Any deviation from the normal behavior is detected by ADE and a malicious pool is populated. Finally, SGE generates signatures for an alert from malicious pool of data. These signatures are used to update signatures repository of IDS and may further be used to safeguard the network from intrusions.

In this thesis, the implementation details of HIDESIGN for automatic generation and tracking of attack signatures have been discussed. The deployment and experimental setup is done in a controlled environment of University's network. Elaboration of various steps like honeypot configuration, data capturing, misuse and anomaly detection along with the procedural details with the help of PseudoCodes have been provided. Signature generation procedure and overall working of the system has also been elaborated. The proposed work has been implemented using various modules developed in *Java*. It makes use of honeypot technology along with *Snort,* a signature based IDS and anomaly detection based on supervised machine learning using Naive Bayes classifier. ADE is trained using two datasets, namely, HTTP CSIC 2010 and a subset of NSL-KDD dataset for training of HTTP and SMTP detection models, respectively.

The performance of HIDESIGN has been evaluated on the basis of results obtained as outcome of the work carried out in this thesis. The evaluation of proposed system has been performed using various evaluation metrics. These metrics have been selected in such a way that the proposed system can be tested from several aspects of good IDS. The results of various metrics like Sensitivity, Specificity, False Positive Rate, Accuracy, Detection Rate, F-Measure and ROC for both HTTP and SMTP attack detection are very encouraging. The proposed system is capable of successfully detecting and generating attack signatures of various HTTP attacks with a sensitivity and specificity of 99.97% and 98.19% for HTTP attack vectors. The sensitivity and specificity for SMTP attacks is 96.83% and 93.3%, respectively. It has reported 1.8% false positives and 0.02% false

negatives for HTTP attacks whereas for SMTP attack detection 6.69% false positives and 3.17% false negatives have been reported. The auto generated signatures in *Snort* format have also been presented in this thesis. Further, HIDESIGN has been compared with existing recent systems proposed in literature and it has been observed that the proposed system is helpful in extending the detection capabilities of Intrusion Detection Systems by generating attack signatures in order to handle novel attacks.

Table of Contents

List of Figures

List of Tables

List of PseudoCodes

List of Snapshots

Chapter 1
Introduction

1.1 Introduction

Computer networks have brought the world together by bridging the information gap among people. Network technology has undergone a revolution with better and faster ways of sending information between computers. Security systems and policies to govern these networks are still in their progress phase. With the birth of the internetworking in 1969, computers and computer networks got interconnected. The Internet has grown at an exponential rate with innovations in information and communication technologies. The requirement for ease of communication has forced companies and government organizations to open their computer networks to the Internet. With e-commerce and online banking being more and more employed these days, security for systems offering these services has become necessary. While the Internet has made it easier to reach and communicate with people all over the globe, it has also made it easier to attack computer systems connected to it. In addition to this, many operating system flaws, application bugs and network protocol vulnerabilities are being discovered with each passing day [Pfl02].

The past few years have seen a radical evolution in the nature and requirements of network security. There are many factors contributing to these changes. In recent years, there is shift in focus from traditional network-level threat such as connection-oriented intrusions, to dynamic and content-based threats such as viruses, worms, Trojans, rootkits, spyware and phishing that can spread quickly and indiscriminately representing an extremely serious threat to the safety of the Internet [Sta02]. Data breaches, web-based attacks, spam, phishing, malware data, social media scams, android based malwares, cloud based vulnerabilities, spear phishing, bandwidth manipulation and delay attacks [Kar13], recent vulnerabilities like Heartbleed and Shellshock are some categories that require sophisticated levels of intelligence to get detected. Cyber crime is becoming more

professional with the emergence of increasingly powerful methods of intrusions and exploits [Jos13]. More and more zero-day vulnerabilities are being discovered in each passing year. Because of increasing level of threats targeting network resources and damage caused by them, it has become necessary to take appropriate measures to safeguard them. Network Security has become more a requirement than choice in the current scenario.

1.2 Network Security

As per NIST *Computer Security Handbook* [Gut95] the term Computer Security can be defined as follows.

> *"The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications)."*

Network security can be defined as set of mechanisms and policies to prevent unauthorized access, misuse, modification, or denial of a computer network and network accessible resources. While using the Internet, along with the convenience and speed of access to information new risks have arisen. Among them are the risks to computer systems, valuable and personal information's loss, theft, corruption and misuse. Intruders can create new electronic files, run their own programs, and even hide all evidence of their unauthorized activity without getting noticed or identified.

It is remarkably easy to gain unauthorized access to information in an insecure networked environment, and it is hard to catch the intruders. The computers can act as a weak link even if they have no production value. These systems may allow unauthorized access to the organization's systems and information. Intruders may be interested in information regarding hardware and software being used, system configuration, type of network connections, phone numbers, email ids, personal information like pictures, access and authentication procedures. Today's commercial off-the-shelf software components are

full of vulnerabilities that allow viruses, worms and other manual or automated attacks to cause damages. The targets can be banks, financial companies, individual persons, defense organizations, insurance companies, brokerage houses, consultants, government contractors, government agencies, hospitals, medical laboratories, network service providers, utility companies, textile industry, universities, wholesale and retail trades [Kab05].

Network Security is a step towards safeguarding resources such as data and information from vulnerabilities and threats. It comprises of all measures, services and mechanisms required to protect valuable information resources from being accessed, modified, deleted, corrupted and exploited by unauthorized person. These resources can be protected in a variety of ways including, via firewalls, antivirus software, antispyware software, data encryption, Intrusion Detection Systems, Intrusion Prevention systems, virtual private networks and automated security policy enforcement. Various such defense mechanisms are discussed briefly in section 1.4.

Security has three fundamental objectives, namely, Confidentiality, Integrity and Availability also commonly known as CIA triad [Wil06].

- **Confidentiality**

  Confidentiality refers to prevention of unauthorized disclosure of information to third parties. It emphasizes on preserving authorized restrictions on information access and disclosure, including information about resources, means for protecting personal privacy and proprietary information.

- **Integrity**

  Integrity is meant to prevent unauthorized modification of resources and maintain the status quo. It includes the integrity of information, personnel and system resources. Integrity includes guarding against improper information modification or destruction, including assurance of information non repudiation and authenticity.

- **Availability**

  Availability is required to prevent unauthorized withholding of system resources from those who need them. Availability ensures timely and reliable access to and use of information. Threat to the Internet availability is a big issue which is hampering growth and survival of e-business and other Internet based applications [Sac10].

1.3 Network Security Approaches

Network Security broadly employs two approaches, namely, Reactive Network Security and Proactive Network Security. These are described in following subsections.

1.3.1 Reactive Network Security

Reactive security approaches include procedures and mechanisms followed to take appropriate actions after discovering some breach or compromise by an intruder, attack program or script. Reactive Network Security services are designed to respond to requests for assistance, and any threats or attacks against systems. Example of reactive network security services are incident handling [Wes03], vulnerability handling, third-party notification, alerts and warnings *etc.* Reactive systems, such as Intrusion Detection System (IDS) gather and analyze information after detection of an attack, incident, or loss of information that ultimately drives to some action or reporting.

1.3.2 Proactive Network Security

Proactive approaches include preventative measures that are taken with the goal of safe guarding against host-based or network-based attacks from successfully compromising systems or network. A proactive system constantly tests the network for vulnerabilities and exposures in order to identify, prioritize and address these issues. All devices attached to the network are periodically scanned and observed for changes, violations to policy, vulnerabilities and exposures. The defect, if any, is then corrected before security can be compromised.

Proactive Network Security [Mil04] is the act of managing different components of network security, *e.g.,* firewall, Intrusion Prevention System (IPS), Virtual Private Networks (VPNs), antivirus software *etc.*, so as to get the most performance from them and to strengthen vulnerability management system. These services are designed to improve the infrastructure and security processes before any incident occurs or is detected. It may help to avoid incidents and to reduce their impact and scope when they occur.

1.4 Defense Mechanisms

Many methods have been developed to secure the network infrastructure and communication over the Internet. Among them are firewalls, encryption techniques, anti viruses, anti spyware, Intrusion Detection and Prevention systems, honeypots and Virtual Private Networks. These systems are expected to have high performance, fault tolerance, easy administration, rigorous security processing, real time response and high speed. These systems and mechanisms can be used to harden network security, as well as for legal purposes. Both commercial and open source solutions are available for this purpose. A comprehensive security system consists of multiple tools capable of providing a mix of reactive and proactive security solutions. These include the following defense mechanisms.

- **Firewalls**

  Firewalls are important at the boundary between the enterprise network and the Internet to avoid any unwanted ingress and egress traffic flow. A firewall has a set of rules that specifies which traffic should be allowed or denied. A static stateless packet filter firewall inspects individual packets to report any abnormality. A stateful firewall can track communication sessions and more intelligently allow or deny traffic. There are several open source and commercial range of firewall products available now days.

- **Antivirus**

  Antivirus is a software used to prevent, detect and remove malicious software. Antivirus can be used to detect and protect from computer viruses, malware, backdoors, rootkits, Trojan horses and worms *etc.* There are various commercial products available in the market to protect from such threats.

- **IDS**

  An Intrusion Detection System (IDS) detects malicious events and notifies about their occurrence. An IDS can perform statistical and anomaly analysis. IDS can report to a central server that correlates information from multiple sensors to ensure an overall real-time security of a network.

- **IPS**

  Intrusion Prevention Systems (IPS) are designed to stop or prevent attacks from entering the network environment in contrast to merely monitoring and reporting. Systems of this nature are designed to assess the characteristics of the traffic patterns, detect abnormal behavior and stop or prevent it from accessing the sensitive systems. An IPS can dynamically take action such as logging or blocking of traffic by adding rules to its repository.

- **Virtual Private Networks**

  Virtual Private Network (VPN) is a private network developed by using public infrastructure to connect nodes. It employs encryption and other security mechanisms such as tunneling, to ensure safe corporate communication through public Internet. A VPN connection across the Internet is similar to a Wide Area Network (WAN) link between websites. Users can access extended network resources in the same fashion as available within the private network, because VPN extends a private network across a public network, such as the Internet. VPNs securely connect geographically separated offices of an organization, creating one cohesive network.

- **Honeypot**

  A honeypot is a resource whose value is in being attacked or compromised. It is specifically designed to capture malicious behavior of attacker. A honeypot may consist of a computer, data, or a network site that appears to be part of a network and seems to be useful resource for attackers. But actually it is isolated and monitored to observe and analyze attackers' moves. It acts on "bait and trap" principle used to lure attackers and get compromised. A honeypot is expected to get probed, attacked and potentially exploited.

The details about two defense mechanisms, namely, Intrusion Detection Systems and Honeypots, have been given in the following sections.

1.5 Intrusion Detection

An intrusion is a deliberate unauthorized attempt to access information, manipulate information, and commit acts to render a system unreliable or unusable [Int04]. Intrusion detection refers to all efforts and processes which are applied to discover unauthorized attempt(s) to use network or computer devices and detect any unusual or abnormal activity.

Intrusion detection is a technique which may be used dynamically to secure the network. Using intrusion detection methods, information from known types of attacks can be collected and used to find out if someone is trying to attack the network or particular hosts. The information collected this way can be used to harden the network security. Both commercial and open source products are available for this purpose.

1.6 Intrusion Detection Systems

Intrusion Detection System (IDS) can be defined as an active process or device that analyzes system and network activity and reports any kind of unauthorized, unapproved or malicious activity. These systems attempt in discovering or detecting the presence of intrusive activities [Jon00]. Their purpose is to monitor and report on the presence of attacks, but not to prevent attacks. Intrusion detection is based upon an in-depth analysis

of prevalent attack profiles in the environment that would trigger an alert. Because new attack profiles continue to evolve in the environment that have not been defined, constant updates to profiles are critical. The overhead of performing real time analysis is a time consuming task. Incorporation of parallel processing in analysis may help in this case [Lin13].

1.7 Classification of IDS

Intrusion Detection Systems can be categorized based on several criteria. The various factors which play major role in the classification are enlisted below.

- Place of Deployment: Describes where the IDS will be physically deployed in the network topology.
- Scope: Decides the scope of coverage of intrusion detection.
- Architecture: Categorizes IDS based on their basic architecture.
- Response: Categorizes IDS based on whether it responds actively or passively.
- Detection Time: This classification is based on whether IDS is real-time or offline.
- Analysis Technique: Describes how the analysis of intrusion would be done, *e.g.,* anomaly detection or misuse detection.

Based on the above parameters, Intrusion Detection Systems can be classified as shown in Figure 1.1.

1.7.1 Classification based on Place of Deployment

On the basis of deployment IDSs can be categorized broadly into two types, namely, inline and sniffer IDS [Int04]. These are elaborated as below.

- **Inline IDS**

Inline IDS is placed in the line of packet transfer. It inspects all packets passing through it for intrusions and generates alerts if something malicious happens. It can block the

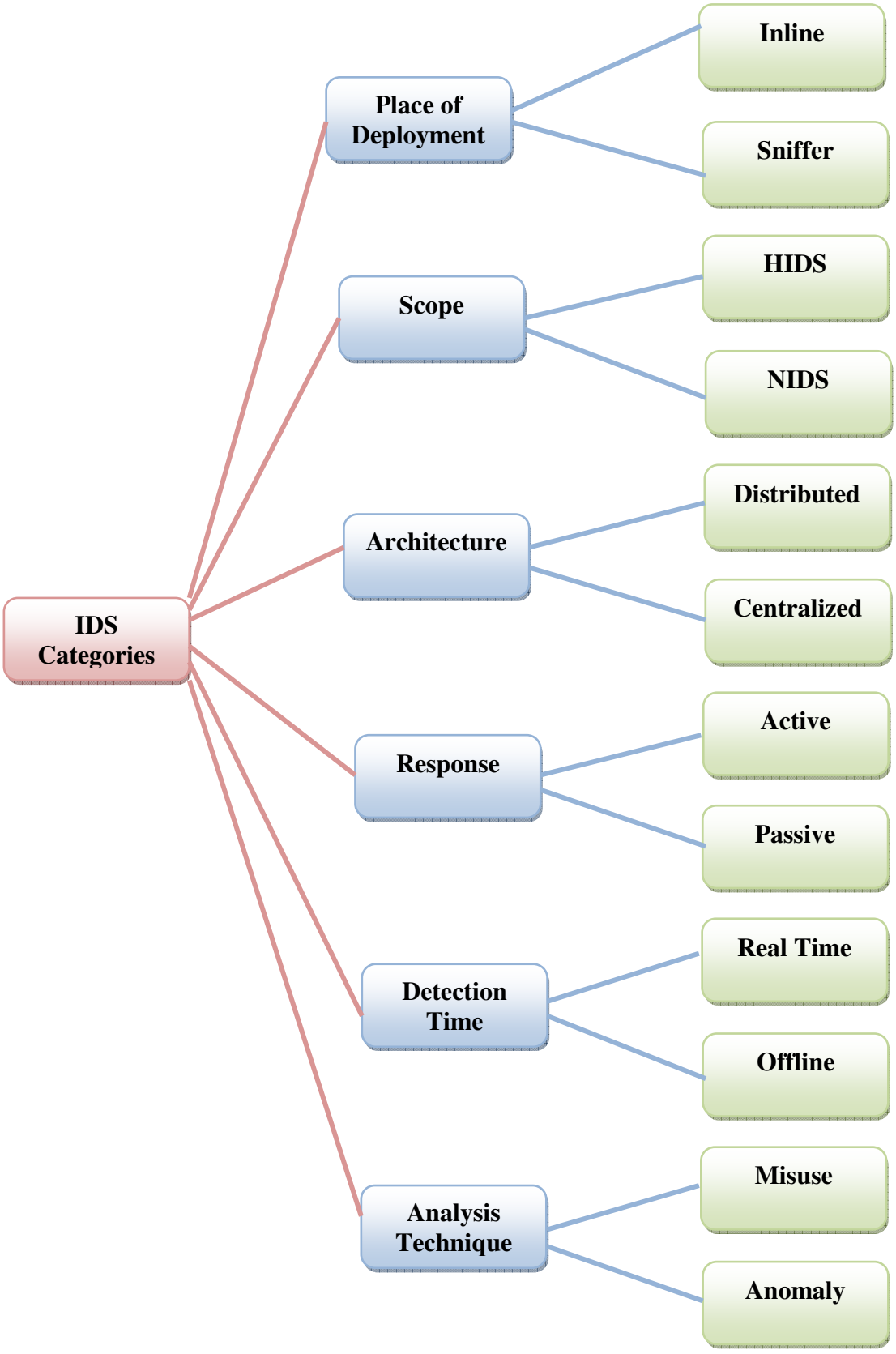


Figure 1.1: Classification of IDS based on various parameters

intrusion from going any further. Inline IDSs are placed at the perimeter of the network, behind the firewall or along with the firewall.

- **Sniffer IDS**

Sniffer IDSs examine packets within a network. These IDSs passively receive the packets from the network and analyze them. Sniffers are placed in a location on the network where they can receive all the packets flowing within the network.

1.7.2 Scope based classification

On the basis of scope, IDSs can be classified broadly as Host Based IDS and Network Based IDS [Muk94].

- **Host based IDS**

Host based Intrusion Detection Systems (HIDS) are installed as agents to monitor activity and behavior on a host. These Intrusion Detection Systems can look into system and application log files to detect any intruder activity. Some of these systems are reactive, meaning that they inform and alarm only when something happens. Some HIDS are proactive as they can sniff the network traffic coming to a particular host on which the HIDS is installed and generate alert in real time. Internal threats can be handled very efficiently by HIDS because they can monitor and respond to user actions and accesses on the host.

- **Network based IDS**

Network based Intrusion Detection System (NIDS) monitors and analyzes the data packets that are being transmitted through a network. They are also called packet sniffers [Kar02]. A NIDS detects and alerts about possible intrusions such as a malicious activity, computer attack, computer misuse, and spread of a virus. NIDS server can be set up on the links of a backbone network, to monitor all traffic or to monitor traffic directed to a particular server, switch, gateway, or router. NIDS can also be setup at a centralized server, which can scan the system files, looking for unauthorized activity and to maintain

data integrity. A typical NIDS has one or more sensors, and a console to aggregate and analyze data from the sensors.

1.7.3 Classification based on Architecture

On the basis of architecture, IDS can be classified as distributed or centralized [Laz05].

- **Distributed IDS**

Distributed IDS (DIDS) analyses information collected from multiple monitored systems in order to investigate distributed and coordinated attacks. There is a recent increasing trend towards distributed and coordinated attacks, where multiple machines are involved. In this kind of scenario, the data collected from single site is not enough to detect attacks. Distributed IDS is a solution in such case because the analysis of data is performed on various locations that are being monitored. DIDS can further be categorized as mobile agent IDS, grid based IDS and most recent cloud based IDS.

- **Centralized IDS**

Centralized IDS analyze the data collected only from a single monitored system. These systems are not able to detect distributed attacks like DDoS (Distributed Denial of Service). Analysis of data is performed on a fixed number of locations. The audit data is collected on the individual systems and then reported to some centralized location where the intrusion detection analysis is performed. This approach works well for smaller network situation but is inadequate for larger networks due to the high volume of audit data that must be analyzed by the central analysis component.

1.7.4 Classification based on Response

Based on the type of response IDS can be classified as active and passive IDS.

- **Active IDS**

An active IDS is configured to automatically respond to attack events in progress without any human intervention. These IDS may take action like blocking a port of

communication, configuration firewall and routers to protect from attacks and applying patch for vulnerabilities.

- **Passive IDS**

Passive IDSs are the ones which are responsible for analyzing and reporting any malicious activity to the security professional. These systems can log, alert or drop the suspected packets depending upon the analysis result, but they cannot actively change any configuration of the system, firewalls or routers so that these attacks can be prevented in future.

1.7.5 Classification based on Detection Time

Detection time is a very important parameter in IDS as it may decide the amount of loss occurred due to attack. If the detection time is less the loss may be lesser than the longer detection time. On the basis of detection time IDS can be classified as real time and offline IDS.

- **Real Time IDS**

Real time IDS works in real time or near real time by continuously monitoring the incoming packet streams and responding by taking action immediately. These are also called as online IDS.

- **Offline IDS**

Offline IDS works offline by performing analysis on the logged data. In these systems, the action is taken offline rather than immediately when the malicious event occurs.

1.7.6 Classification Based on Analysis Technique

There are mainly two techniques of analysis of events for the purpose of attack detection, namely, anomaly detection and misuse detection [Gho09].

- **Anomaly Detection**

Anomaly detection is based on the analysis of profiles that represent normal behavior of users, hosts, or network connections [Laz05]. Anomaly detectors characterize normal, legitimate computer activity using several techniques and construct profiles that represent normal usage. These profiles are used for the comparison of the current behavior data to detect a possible mismatch. The major benefit of anomaly detection algorithms is their ability to potentially recognize unforeseen attacks. However, the major limitation is high false alarm rate. Deviations detected by anomaly detection algorithms may not necessarily represent actual attacks, as these may be new or unusual, but still legitimate, network behavior. Anomaly Detection can be accomplished by using various techniques including Statistical Measures; Expert Systems; Artificial Neural Networks; Machine learning; Data mining techniques and User behavior patterning.

There are certain limitations of anomaly based detection technique as discussed below briefly.

- The false alarm rate is substantial.
- There is a requirement of constant update of the normal behavior profile database as user behaviors can vary with time.
- The necessity of training the system for changing behavior makes a system immune to anomalies detected during the training phase.

In contrast to anomaly detection another detection technique named as misuse based detection reports intrusions on the basis of patterns matching with the known signatures of malicious activities.

- **Misuse Detection**

Misuse detection, also commonly known as Signature detection, is considered complementary to anomaly detection. The underlying principle is that known attack patterns can be detected more effectively and efficiently by using explicit knowledge of them. Thus, misuse detection systems look for well-defined patterns of known attacks or

vulnerabilities. This approach attempts to match observed behavior against known intrusive behavioral patterns [Nin03]. A variety of techniques has been proposed in literature to model and recognize attack patterns including Rule Based systems; Signature analysis; Petri nets; State-transition analysis and Genetic algorithms.

A common element between these techniques is that they attempt to represent the essential nature of a known attack in such a way that variations on that attack can be distinguished from normal behavior. Anything that is not recognized as an attack is accepted as legal behavior. The most common misuse detection technique used by the IDS is Signature based analysis and detection. In signature based technique, the signatures or patterns of known attacks are stored in the database called as signature repository. These signatures are then matched with incoming packets. In case of a match, actions like logging, alerting or dropping of the packets can be taken. Various limitations of misuse detection approach are discussed below.

- It is difficult to update information on new types of attacks.
- Maintenance of NIDS is a time-consuming process as it is associated with analysis and patching of security vulnerabilities.
- They are not able to detect unknown and novel attacks whose signatures are not available in the library. A continuous update of the attack signature database is required for correlation.
- The attack signatures may be operating environment dependent, so misbehavior signature based Intrusion Detection Systems should be configured in accordance with the operating system details such as version, platform, and applications used *etc.*

IDS is a detection mechanism which is considered to be a part of reactive network security. But reaction is not always useful. In some cases proactive measure should be used to understand the behavior and policies of attackers, so that appropriate actions could be taken timely to safeguard from attacks. So, a proactive security measure called honeypot technology has been a choice of various researchers for effective defense. The

details about honeypots, their classification and various available honeypots are elaborated in the following sections.

1.8 Honeypot: A proactive technique for Intrusion Detection

A honeypot is specifically designed to capture malicious network traffic. It acts on "bait and trap" principle. It is used to lure attackers and get compromised. *L. Spitzner* (2003) defines the term honeypot as follows.

"A honeypot is a resource whose value is being in attacked or compromised. This means, that a honeypot is expected to get probed, attacked and potentially exploited. Honeypots do not fix anything. They provide us with additional, valuable information". [Spi03]

The logging capability of a honeypot is far greater than any other network security tool and captures raw packet level data even including the keystrokes and mistakes made by hackers. The captured information is highly valuable as it contains only malicious traffic with little to no false positives. Honeypots are one of the leading proactive security tools used to monitor the latest attacks and exploits of hackers by recording their every move so that the security community can more quickly respond to new exploits. Honeypots do not help directly in increasing a computer network's security. But, tracking and analyzing activities on honeypot can help security administrator to look and interpret attackers' moves which is further helpful to harden the security so that actual production systems does not get compromised in the same way.

History and evolution of honeypots through past years, classification of honeypots and existing honeypot systems are discussed in the forthcoming subsections.

1.8.1 History and evolution of Honeypots

The concept of honeypots became public in early 1990s. In 1991 two publications, namely, "*The Cuckoos Egg*" a book written by Clifford Stoll and "*An Evening with Berferd*" a whitepaper by Bill Cheswick were published. In these two publications, the events monitored while tracking the hacker in different scenarios have been presented

[Spi03]. In 1997 Fred Cohen's *Deception Toolkit* [Coh98], one of the first original and landmark honeypot solutions available to the security community was released. It was a collection of *Perl* scripts written for UNIX systems to emulate various vulnerabilities. In 1998, one of the first commercial honeypots called *CyberCop Sting3* was developed that introduced the concept of multiple, virtual systems bound to a single honeypot. In the same year 1998, a honeypot solution called *NetFacade* was developed by Marty Roesch and GTE Internetworking which could emulate seven different operating systems with different services. *BackOfficer Friendly*, released in 1998, was a free, easy to use, Windows based honeypot. In 1999, Honeynet project was initiated which is considered to be a major contributor to the honeypots research. In early 2000s, honeypots were started to be used to capture and study worm behavior and activity. In 2002, a Solaris honeypot was used to detect unknown Solaris *dtspcd* exploit. In 2003, Honeynet project and Honeynet research alliance came up with various tools like *Snort Inline*, *Sebek* and advanced virtual honeynets [Spi03]. Many of these are discussed in subsection 1.8.3. Since then honeypots have been in use in research and development of new security solutions. Today honeypots are in use as a proactive security measure for prevention of novel worms and attacks in both wired as well as wireless network. For instance, blackhole attacks can be detected in a wireless mesh network using intelligent honeypot agents [Pra13].

1.8.2 Honeypots Classification

Honeypots can be broadly categorized into two categories, namely, production honeypots and research honeypots. A production honeypot is helpful in migration of security risk from actual production servers to fake servers in an organization. While the second category, research honeypot, is meant to gather as much information as possible for further analysis and research on intruder's activity. These honeypots do not add any security value to an organization, but they can help to understand the behavior of attackers and their attacks as well as to build some better defenses against security threats.

Honeypots can further be classified into three categories based on their level of interaction, namely, low interaction, medium interaction and high interaction honeypot. A low interaction honeypot provides only certain fake services. The purpose of such a honeypot system lies in detection of simple attacks. The risk of such honeypots is quite low as there is no real operating system for interaction. A high interaction honeypot on the other side does provide some real services with real vulnerabilities. The risk is much higher due to a real underlying operating system exposed to the attacker to get root access. A medium interaction honeypot offers more ability to interact than low interaction honeypots but less functionality than high interaction solutions. Hence, they are safer than high interaction ones and at more risk than low interaction honeypots. High interaction honeypots are more informative for security analysts as they capture more activities of hackers by exposing maximum applications and vulnerabilities to them.

Honeynets are complex systems consisting of multiple honeypots to look more like productive systems. These systems allow the simulation of realistic productive environments. Honeynets comes under the category of high interaction honeypots. Here, the risk is significantly higher than the risk of a single honeypot. But these are more informative than single hoenypots [Bal05].

1.8.3 Available Honeypots

There are various free and commercial honeypots available these days. They differ in functionality, complexity and ease of use. *ManTrap* is a medium interaction honeypot running on Solaris operating systems [Spi03]. *Honeyd* [Pro03b], a low interaction honeypot is able emulate the IP stacks of various target operating systems with a variety of services at the same time. *Specter* is a low involvement honeypot running on different operating systems [Spi03]. *Nepenthes* [Bae06], is a low-interaction honeypot which emulates common Windows known vulnerabilities and downloads the payloads when an attacker attempts to exploit them. *Honeytrap* [Wer06] is another low-interaction network services honeypot that provides basic service emulation. *Kojoney* [Cor06] is a low-interaction honeypot that emulates an SSH server process and records the usernames and passwords of attackers attempting to log in to the honeypot. *Sebek* is a tool for monitoring

high interaction honeypots. *Hflow* [Pro03a] is a data coalescing tool for honeynet analysis that combines data from *Snort*, *p0f*, and *Sebekd* into a unified cross related data structure to be storage in a relational database. *Honeywall* [Cha04] is a bootable CDROM which can be used to quickly build a high-interaction honeynet. *Capture-HPC* [Sei06] is a high-interaction client honeypot framework.

Various web application honeypots have also been developed including *Google Hack Honeypot* [McG07] and *HIHAT* [Pro07]. *Google Hack Honeypot (GHH),* written in *PHP,* is a system designed to be vulnerable to sophisticated search engine queries for the purpose of attracting search engine hackers. *HIHAT (High Interaction Honeypot Analysis Toolkit)* is a high-interaction web application honeypot and data reporting interface. It allows easy conversion of many existing *PHP* applications to a honeypot and has been successful in monitoring web application attacks such as SQL injection and remote file includes. There are many honeypot solutions available and many more are being proposed. HoneyNet project is a big contributor to the research of honeypots.

In this research work, the major objective is to propose a network security model for attack signature generation, tracking and analysis. For the accomplishment of this objective various methods, technologies and tools have been used. These are briefly introduced in the previous sections. In this thesis, a hybrid framework for attack detection and signature generation named HIDESIGN (Hybrid Intrusion Detector and SIGNature generator) has been proposed, implemented and tested. The rest of thesis is organised as described in the following section.

1.9 Organization of Thesis

The work carried out in this thesis has been organized into six chapters. These chapters include Introduction; Review of Literature; HIDESIGN: A Proposed framework for Attack Detection and Signature Generation; Implementation details of HIDESIGN; Results and Discussions; Conclusions and Future Scope. A brief detail on these chapters has been presented below.

First Chapter: Introduction

In this chapter introduction to network security, its need in current era of information technology and challenges in development of a network security system have been discussed. The details of different approaches of network security, namely, reactive network security and proactive network security have also been discussed. Various defense mechanisms like Firewalls, anti-virus, IDS, IPS, VPNs and Honeypots are also explored. It elaborates IDS and their classification in detail. There are two major techniques for detecting intrusions, namely, anomaly detection and misuse detection. In this chapter, insight about honeypots, history and evolution of honeypots, classification of honeypots and a brief discussion about existing honeypots have been provided. This chapter provides introduction to these concepts and concludes by providing organization of thesis.

Second Chapter: Review of Literature

Second chapter of thesis provides details of reported literature in the area of network security. Review of literature has been performed by tracking historic developments in this field. Literature reported in this work includes extensive details about history and evolution of IDS through decades. Detailed architecture of Intrusion Detection Systems namely *Snort* and *Bro* has been explained. In this chapter, the role of machine learning in intrusion detection has also been elaborated. As the proposed work is a framework for automatic generation of attack signature for IDS, literature review includes various existing systems for automatic attack detection and signature generation. This chapter also provides a detailed study of architecture of these systems, their working and various parameters for evaluation. The chapter concludes by presenting the research motivation, objectives of the proposed work and major contributions of thesis**.**

Third Chapter: HIDESIGN: A Proposed framework for Attack Detection and Signature Generation

In this chapter the details of proposed framework, named, HIDESIGN for automatic generation and tracking of attack signatures in context with HTTP and SMTP traffic have been discussed. Here, an introduction to HTTP and SMTP protocols background has been

provided. It also discusses HTTP and SMTP attack vectors in details. Various types of HTTP attacks such as SQL injection, command injection, directory traversal and cross site scripting attacks have been discussed in detail. SMTP attacks and their details are also discussed in this chapter. Followed by detailed discussion of HTTP and SMTP attacks, the proposed framework for attack signature generation and tracking is presented. This chapter discusses the detailed architectural design and work flow of proposed system and its components.

Fourth Chapter: Implementation details of HIDESIGN

This chapter contains the details about deployment details of HIDESIGN in actual network environment and experimental set up. Here, configuration of honeypot, implementation details of misuse detection engine and anomaly detection engine are also provided. This chapter also explains how signature generation engine has been implemented. It elaborates various experiments performed and their outcome. This chapter provides overall working details of HIDESIGN by jointly presenting all modules.

Fifth Chapter: Results and Discussions

This chapter discusses various evaluation criteria used for measuring performance of HIDESIGN. This chapter also presents the results and discussion in detail.

Sixth Chapter: Conclusions and Future Scope

This chapter concludes the thesis by highlighting the major contributions of this work. To enhance this work further, research directions are discussed in future scope.

Chapter Summary

In this chapter, introduction to several concepts and terms used in the area of security has been provided. The definition and need of network security, various approaches to network security, namely reactive and proactive security have been elaborated. Different available security solutions in terms of defense mechanisms such as Firewall, Antivirus, IDS, IPS, VPN and Honeypots are briefly described. In this research work two of these mechanisms, namely, IDS and honeypots are used. A detailed description about Intrusion Detection, Intrusion Detection Systems and their classification based on various factors has been given. A proactive security measure called honeypot, their classification and various available honeypots are elaborated in this chapter. The chapter has concluded by describing organization of thesis.

Chapter 2
Review of Literature

2.1 Introduction

Networks have grown too complex these days that it has become difficult to secure them from intruders. It has become the need of the hour to develop robust techniques those can protect the networks and their resources from being damaged. There are various reactive and proactive security approaches available to safeguard the resources from unauthorized persons. In this work, enhancement of detection and automatic signature generation capability of Intrusion Detection Systems has been done by using a hybrid approach. This chapter discusses the background of intrusion detection and related work carried out by researchers in this area of research.

In this chapter, evolution of Intrusion Detection Systems (IDS) through past decades, along with architecture of two IDSs named *Snort* and *Bro* has been elaborated. Also, role of machine learning in intrusion detection has been presented by highlighting work of various researchers in this field. Further, this chapter describes the concept of attack detection, signature generation and common techniques used for signature generation. The contribution of various researchers in the area of automatic signature generation has been reviewed and analyzed in a comparative manner. The chapter concludes by presenting the motivation of research, objectives of proposed work and major contributions of thesis.

2.2 Evolution of IDS

Intrusion Detection has been the area of research for various network security professionals for more than three decades. It started in 1970's and with the advancement in technology and levels of threats, there is still a lot of scope to contribute to its research.

The evolution of IDS through the years and decades is presented in the following subsections.

2.2.1 Beginning of the era (1970 to1980)

In 1970, Willis H.Ware of RAND Corporation issued a report on Security Controls for Computer Systems [War70] for Defense Science Board's Task Force on Computer Security that acted as the foundation stone for the research in security and development of computer systems for the processing of multi level data. In 1972, Anderson developed two volume report [And72a], [And72b] on Computer Security Technology Planning study developed for US Air Force. Anderson raised very important questions about IDS which are still important in today's IDS research. The questions raised in this study addressed what should be detected, how to analyze it, and how to protect the IDS system and its data from attack.

2.2.2 IDS Research Decade (1980 to 1990)

In 1980, Anderson [And80] had proposed a computer security threat monitoring and surveillance system. This system aimed at the improvement of security auditing and surveillance capability of a computer system. This acted as a very big inspiration in the research and development on intrusion detection. In 1985, Denning and Neumann proposed a hybrid model for real time Intrusion Detection System named *IDES (Intrusion Detection Expert System)* [Den85], [Den87] which incorporated both signature analysis and anomaly detection. It was one of the most significant and persistent of the early IDS research efforts. *NIDES (Next-Generation Intrusion Detection Expert System)* [And95], the extended version of *IDES,* was further extended to latest system *EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances)* [Por97].

Smaha in 1988, proposed *Haystack* [Sma88], an Intrusion Detection System for the US Air Force to analyze audit data by comparing it with defined patterns. It reduced the larger volumes of audit trails into short summaries of anomalous events. *MIDAS (Multics Intrusion Detection and Alerting System)* [Seb88] was developed by the National Computer Security Center to provide intrusion detection for its Multics system,

Dockmaster. It was based on a hybrid expert system with an idea of heuristic intrusion detection that used statistical analysis approach and examined audit log data from the Multics. *Wisdom & Sense (W&S)* [Vac89], developed at Los Alamos and Oak Ridge, was another hybrid statistical and expert system based on a unique approach to anomaly detection. It used to study historic audit data to make a set of rules describing normal behavior, forming the '*wisdom*' of the system. These rules were then fed to an expert system for evaluating recent audit data for violations of the rules, and generating alerts whenever anomalous behavior was observed, thus forming the '*sense*'. It used machine learning technique in which statistical techniques were used to derive its rule base from historical audit data.

2.2.3 Decade of Commercial Development (1990 to 2000)

Commercial development of intrusion detection technologies began in the early 1990s. *Haystack* Labs was the first commercial vendor of IDS tools, with its *Stalker* product line of host based IDS [Sma94]. *Stalker* was a host based pattern matching system which could search and query the audit data both manually and automatically. *NSM (Network System Monitor)* [Her90, Muk94], developed in 1990 at the University of California, used network traffic as the primary source of data for analysis. It was the time when the concept of Network Intrusion System was introduced. It was first system of this kind which used to listen passively to all network traffic that pass through a broadcast LAN, and report intrusive behavior from this input. Most of the NIDS of today use the live network traffic for the audit data.

In 1991, Heberlein along with the *Haystack* team introduced *Distributed Intrusion Detection System (DIDS)* [Sna91a, Sna91b] project which was first idea of hybrid intrusion detection. *DIDS* incorporated *Haystack* and *NSM* in its architectural framework. *NADIR (Network Anomaly Detection and Intrusion Reporter)* [Hoc93] was a misuse detection system developed at Los Alamos National Laboratory's Integrated Computing Network (ICN). This was an automated expert system which helped security auditors by automating the auditing and reviewing process which was earlier done manually. *USTAT* [Ilg93] was a real time intrusion detection tool based on state transition analysis approach

to intrusion detection that used to detect intrusions by observing the state changes from initial secure state to the compromised target state as a result of a number of penetrations. The commercial market of intrusion detection began to gain more popularity and generate revenues in mid 90's. In 1995, WheelGroup, later acquired by Cisco, commercialized a security product called *NetRanger* [Gle96, Cis99] which was initially developed for the US Air Force. It was designed to monitor network links and their traffic to identify misuse as well as suspicious and malicious activity. In 1996, *GrIDS(Graph based IDS)* [Sta96] came up as a distributed IDS that used event graphs to model network activity. *GrIDS* used to collect data about activity on computers and network traffic between them. This information is aggregated into activity graphs which reveal the causal structure of network activity. This helped large scale automated or coordinated attacks to be detected in near real time. Internet Security Systems (ISS) launched a tool for network security with real time attack detection called as *RealSecure* [Rea97]. The *EMERALD* [Por97] was distributed approach to network surveillance, attack isolation, and automated response. This model used combined approach based on signature analysis with statistical profiling to provide near real time protection to network resources.

NetSTAT [Vig98], a network-based intrusion detection approach, was aimed at real-time network based intrusion detection in order to represent attack scenarios in a networked environment. *Network Flight Recorder (NFR)* [Ran98] was a combination of content based network monitoring and an efficient filtering mechanism, with some support for distributed review using a centralized concentration point. In 1999, Paxson proposed a NIDS called *Bro* [Pax99] for detecting network intrusions in real time by passively monitoring a network link. Today, *Bro* is widely used open source NIDS which is used by many people including researchers. This system has been discussed in detail in section 2.4.2. In 1999, Roesch had proposed *Snort* [Roe99] which is a very popular open source and lightweight NIDS. Initially it was started with a support to a wide variety of UNIX platforms. But today it is available for Windows platform also. It features rules based logging and can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes. *Snort* has been used in this research work and is discussed in more detail in section 2.4.1.

2.2.4 Research Since 2000

AAFID (Architecture for Intrusion Detection using Autonomous Agents), a distributed intrusion detection architecture and system, developed in CERIAS at Purdue University, was the first architecture that proposed the use of autonomous agents for doing intrusion detection [Bal98]. *NFIDS (Neuro-Fuzzy Intrusion Detection System)* [Moh03] is an anomaly based Intrusion Detection System that uses fuzzy logic and neural network to detect if malicious activity is taking place on a network. *MINDS (Minnesota Intrusion Detection System)* [Ert04] uses a suite of data mining techniques to automatically detect attacks against computer networks and systems. The major techniques used in *MINDS* are statistical analysis, pattern matching, data mining and outlier detection. *N@G (Network at Guard)* [Ann05] developed by CDAC is a hybrid Intrusion Detection System (IDS) having capabilities of both misuse and anomaly detection. *Anagram* [Wan06], a content anomaly detector that uses payload modeling through *N*-grams, was designed to detect anomalous and suspicious network packet payloads. *GIDRE* [Olg09] is mechanism for early detection and response to attacks, as well as distribution of information about attack characteristics through the network using grid architecture. It uses stochastic modeling and pattern matching for intrusion detection.

Usmani *et al.* (2013) had proposed a layered Internet traffic monitoring framework for the Internet service providers so as to provide a reliable and systematic way of incident handling by monitoring the Internet traffic to combat cyber crimes, cyber terrorism, and cyber disasters. This framework also provides a foundation for the overall security management at Computer Emergency Response Team (CERT), Mauritius [Usm13]. Joshi *et al.* (2013) had proposed a framework for automatic extraction of relevant information about new security vulnerabilities, attacks and events from text. This framework also represents and integrates this information along with data extracted from the National Security Vulnerability Database as a RDF (Resource Description Framework) linked data representation. The collaboration and representation of these information sources in a structured, semantic, machine-understandable format helps in automated handling of possible zero day attacks [Jos13]. Many other systems using signature based, anomaly based and hybrid techniques have also been proposed since 2000. Although, misuse

detection is a very commonly used technique for intrusion detection but it is not capable of detecting novel attacks and vulnerabilities. So, anomaly detection is also a choice of various researchers in the field of intrusion detection. For the purpose of anomaly detection, various approaches can be used as discussed in Section 1.7.6. In this thesis work, we have used machine learning based anomaly detection. Some of the major related contributions in the area of intrusion detection using machine learning approach are discussed in following section. In this section, several standalone and hybrid anomaly based contributions in the field of intrusion detection using machine learning have been reviewed.

2.3 Machine Learning based Intrusion Detection

The major approaches to anomaly detection [Thu10] include machine learning, statistical methods and knowledge based [Chh09] techniques. In literature, numbers of anomaly detection systems are developed based on many different machine learning techniques [Tsa09]. Machine learning techniques are based on establishing an explicit or implicit model that enables the categorization of patterns to be analyzed. Machine learning based network intrusion detection is a valuable technology to protect target systems and networks against malicious activities [Bhu14]. The major machine learning based schemes include Bayesian networks [Fri97], [Ben07], Markov models, neural networks, expert systems, fuzzy logic [Cha12], genetic algorithms, clustering, classification [Dom97], outlier detection [Bar02] and non-stationary models [Mah02]. Machine learning is very effective approach to train Intrusion Detection Systems and for building rule sets. It is heavily based on statistical analysis of data and pattern recognition in previous data to make decisions about new data. Many researchers including [Amo04], [Gre07], [Lee98], [Lee99], [Ore09] and [Sco04] have proposed and analyzed machine learning techniques for intrusion detection in both wired and wireless communications. Despite the variety of such methods described in the literature in recent years, security tools incorporating anomaly detection functionalities are just starting to appear [Gar09].

Barbara *et al.* (2001) had proposed *ADAM (Automated Data Analysis and Mining)* [Bar01] that provides a test bed for detecting anomalous instances. *ADAM* uses a combination of classification techniques and association rule mining to discover attacks

in a *tcpdump* audit trail. *ADAM* uses a classifier which has been trained to classify suspicious connections as a known type of attack, an unknown type or a false alarm. Estévez-Tapiador *et al.* (2004) found in a study that certain features extracted from HTTP requests could be used to distinguish anomalous traffic from normal connections. They also proposed an anomaly based approach to detect attacks carried out over HTTP traffic. They had applied statistical technique using Markov chains to model HTTP network traffic. Packet payload was used as a parameter to evaluate incoming HTTP traffic [Est04]. Chebrolu *et al.* (2005) presented an ensemble approach by combining two classifiers, Bayesian networks (BN) and Classification And Regression Trees (CART). They incorporated hybrid architecture for combining different feature selection algorithms for real world intrusion detection for better results [Che05].

Panda and Patra (2007) had proposed network intrusion detection using Naive Bayes. They concluded that on the KDD Cup'99 data set Naive Bayes performed better in terms of false positive rate, cost and computational time compared to a back propagation neural network based approach [Pan07], [Pan09]. Lee *et al.* (2007) proposed a hybrid approach for real time Intrusion Detection System using Random Forest (RF) for feature selection and Minimax Probability Machine (MPM) for intrusion detection [Lee07]. Hwang *et al.* (2007) proposed hybrid Intrusion Detection System that combined the advantages of low false positive rate of signature based Intrusion Detection System and the ability of anomaly detection system to detect unknown attacks [Hwa07]. *Snort* was modified by [Tor07] to operate as hybrid detector. This system could operate as hybrid detector or only as an anomaly based detector. Grégio *et al.* (2007) applied and evaluated data mining techniques including k-Nearest Neighbors (k-NN), Artificial Neural Networks (ANN) and Decision Trees on log data sets acquired from a real network and a honeypot, in order to classify traffic logs as normal or suspicious. This approach proved to be helpful in reducing amount of logs to the network administrator, improving the analysis task and aiding in discovering new kinds of attacks against networks [Gré07]. Xiang *et al.* (2008) proposed a multiple level hybrid classifier for intrusion detection which combines the supervised tree classifiers and unsupervised Bayesian clustering. Performance of this approach reported high detection and low false alarm rates when

tested with KDD Cup'99 dataset [Xia08]. Zhang *et al.* (2008) described a systematic framework that applies random forests in misuse, anomaly, and hybrid network based IDSs. In signature detection, patterns of intrusions are built automatically over training data using random forest algorithm thereby detecting intrusions by matching network activities against the patterns. In anomaly detection, the outlier detection mechanism of the random forests algorithm is used to detect the intrusions [Zha08].

Aydin *et al.* (2009) combines the anomaly detection approach and the signature detection approach to form hybrid IDS. The scheme uses *PHAD* (*Packet Header Anomaly Detection*) [Mah01] and *NETAD* (*Network Traffic Anomaly Detection*) [Mah03], both anomaly based IDSs with the misuse based IDS *Snort* [Ayd09]. Gómez *et al.* (2009) extended a new anomaly preprocessor to *Snort* IDS. This hybrid IDS is named as *H-Snort* [Gom09]. Danev *et al.* (2010) used k-NN and Support Vector Machine (SVM) classifiers to study the feasibility of performing impersonation attacks on the modulation-based and transient-based fingerprinting techniques in wireless communication. They had assessed the feasibility of performing impersonation attacks by simulations using collected data from wireless devices and extensive measurements [Dan10]. Farid *et al.* (2010) had proposed a system by combining Naive Bayes and decision tree for adaptive intrusion detection. They validated it resulting in balanced detections and acceptable level of false positives rates for different types of network attacks. Their proposed algorithm also addressed some difficulties of data mining such as handling continuous attribute, dealing with missing attribute values, and reducing noise in training data [Far10]. Zhao *et al.* (2010) proposed a hybrid IDS network and host based IDS that combined anomaly and misuse detection approach. Auditing was used to extract an extensive set of features to describe each connection as network or host session. Data mining programs are applied to capture the behavior of normal activities and attacks [Zha10]. Yang *et al.* (2010) suggests a hybrid Intrusion Detection System using both misuse detection and anomaly detection techniques [Yan10].

Amza *et al.* (2011) proposed a hybrid Intrusion Detection System, combining pattern matching with a neural network detection component to detect anomalies in the network

traffic [Amz11]. Nguyen *et al.* (2011) had explored application of the Generic Feature Selection Measure (GeFS) in detection of web attacks. They conducted experiments on the publicly available ECML/PKDD-2007 and self created CSIC 2010 dataset which target real web applications. They applied two instances of GeFS measure for intrusion detection, namely, the correlation feature selection (CFS) measure and the minimal Redundancy Maximal Relevance (mRMR) measure [Ngu11]. Muda *et al.*(2011) proposed hybrid learning approach for better Intrusion Detection through a combination of classification and clustering. They used *k*-means clustering to cluster all data into the corresponding group of samples that behave similarly and dissimilarly such as malicious and non-malicious before applying Naive Bayes classifier to classify all data into correct class category. The experiments were carried out using KDD Cup'99 dataset to evaluate the performance of their approach [Mud11].

Palmer (2011) had proposed Naive Bayes classification for intrusion detection using live captured packets. KDD Cup'99 data set has been used as a valid source for training data of machine learning algorithms for intrusion detection [Pal11]. Hussein *et al.* (2012) proposed a hybrid IDS by integrating signature based system *Snort* with Naive Bayes based anomaly detection to enhance system security [Hus12]. Qazanfari *et al.* (2012) used Support Vector Machine (SVM) and Multi Layer Perceptron (MLP) as anomaly detection approaches to develop a hybrid intrusion detection approach [Qaz12]. Yerima *et al.* (2014) had analyzed Bayesian classification based approaches for android malware detection. They had developed and analyzed proactive machine learning approaches based on Bayesian classification aimed at uncovering unknown android malware via static analysis [Yer14]. Abdulla *et al.* (2014) applied *k*-Nearest Neighbors and Naive Bayes, for detection of unknown scanning and email worms [Abd14].

There are several commercial and open source Intrusion Detection Systems available. The details about two popular IDSs have been provided in the following section.

2.4 De Facto IDSs

There are various existing IDS which are proposed in literature. The most popular de facto open source IDSs are *Snort* and *Bro*. These two systems along with their architectural details are discussed below.

2.4.1 Snort

Snort [Roe99] is an open source Network based Intrusion Detection System (NIDS). It is very powerful Intrusion Detection System that performs real time traffic analysis and packet logging. It uses *libpcap* [Jac09] to sniff and filter packets. *Snort* is primarily a rule based IDS, however input plug-ins (or pre-processors) are present to detect anomalies in protocol headers. *Snort* has a rich set of pre-defined rules to detect intrusions. The rule files are customizable, *i.e.*, rules can be added or removed to avoid false alarms. *Snort* stores rules grouped in categories in separate updateable files. These files are then included in *Snort.conf*, the main configuration file. It reads these rules at the start up time and builds internal data structures or chains to apply these rules to captured data.

2.4.1.1 Architecture of Snort

Snort is logically divided into five components. These components work together to detect particular attacks and to generate output in a required format from the detection system. Major components of *Snort* are Packet Decoder, Preprocessors, Detection Engine, Logging and Alerting System, and Output Modules. Figure 2.1 depicts the architecture of *Snort*. Incoming traffic packet travels through various components of *Snort* and is either dropped, logged or generates alert during this journey. The description of each individual module described in the following discussion.

- **Packet Decoder**

The packet decoder intercepts the packets from different types of network interfaces like Ethernet, PPP (Point-to-Point Protocol) *etc.* and prepares the packets to be preprocessed or to be sent to the detection engine.

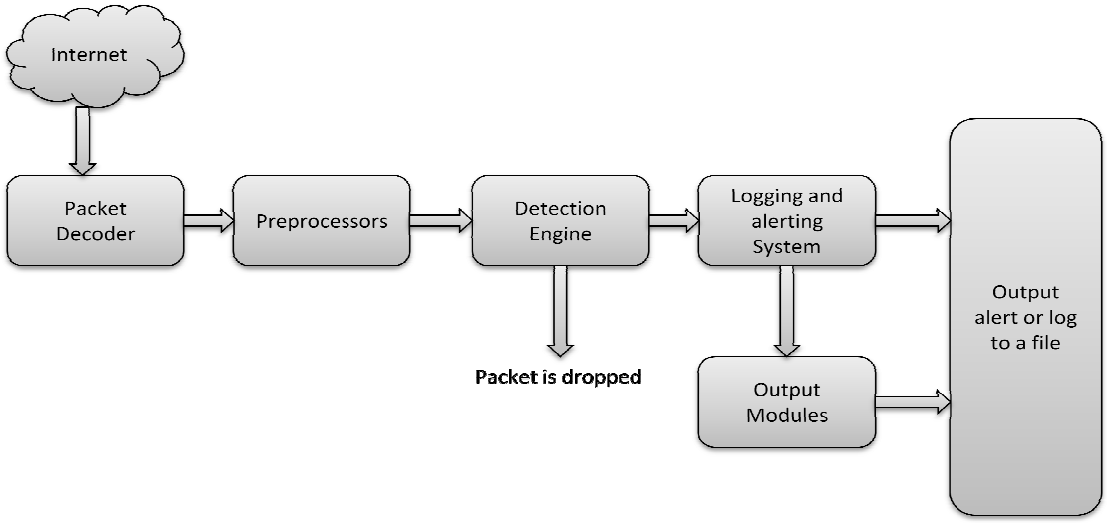


Figure 2.1: Architecture of *Snort* [Roe99]

- **Preprocessors**

These are the optional plug-in components that can be used in connection with *Snort* to arrange or modify data packets before forwarding it to detection engine. For instance, some pre-processors may be used to detect anomalies in packet headers, generate alerts, rearrange the string to make it detectable by the IDS, packet defragmentation, re-assemble TCP streams and so on.

- **The Detection Engine**

The detection engine is the most important and time critical part of *Snort* responsible to detect any malicious packet in the ingress traffic. The detection engine employs *Snort* rules for this purpose. The rules are read into internal data structures or chains where they are matched against all packets. If a packet matches any rule, appropriate action like logging, alerting or dropping that packet can be taken.

- **Logging and Alerting System**

Logging and Alerting System logs the activities or generates alerts depending upon what the detection engine finds inside a packet. Logs are stored in text or *tcpdump* format. By default, logs are stored under */var/log/Snort* folder.

- **Output Modules**

These modules control the type of output generated by the logging and alerting system. Depending on the configuration, output modules can perform variety of actions like simply logging to alerts file (inbuilt or user created), sending messages to syslog, logging to a database like MySQL or Oracle, generating eXtensible Markup Language (XML) output, modifying configuration on routers and firewalls *etc.*[Reh03].

2.4.1.2 Structure of Snort Rule

Snort has a simple, flexible, lightweight but quite powerful rules description language. A *Snort* rule has two parts, namely, Rule header and Rule options.

- **Rule header**

The rule header contains the information about rule's action, protocol, source IP, destination IP, net masks, and source and destination ports. This information defines what part of a packet to be inspected and action to be taken if a packet with all the attributes indicated in the rule match [Sno10]. All portions contained in *Snort* rule header are compulsory. Various parts of rule header are described below.

- **Rule Actions**: The rule action guides *Snort* what to do when it finds a packet that matches the rule criteria. There are five available default actions in *Snort*. These are *alert, log, pass, activate, and dynamic* [Sno10]. *Alert* is used to generate an alert using the selected alert method, and then log the packet; *log* is used to log the packet; *pass* is used to ignore the packet; *activate* is used to alert and then turn on another dynamic rule; *dynamic* instructs *Snort* to remain idle until activated by an activate rule thereafter acting as a log rule. In addition, if *Snort* runs in inline mode, additional options for rule actions are available which include *drop*, *reject*, and *sdrop*.
- **Protocol**: There are four protocols that *Snort* analyzes for suspicious behavior, namely, TCP, UDP, ICMP and IP.

- **IP addresses:** This portion is used to specify source IP address and destination IP address of the packet. The keyword *any* may be used to define any address. The addresses are formed by a straight numeric IP address and a CIDR block.
- **Port Numbers:** Port numbers specifies the source and destination port. Here also, *any* may be used in the same way as in IP address. Port numbers may be specified using static port definitions, ranges, and by negation.
- **Direction operator**: The direction operator → indicates the orientation, or direction, of the traffic to which the rule applies. The IP address and port numbers on the left side of the direction operator is considered to be the traffic coming from the source host whereas the address and port information on the right side of the operator is the destination host. There is also a bidirectional operator, which is indicated with a <> symbol. This guides *Snort* to consider the address/port pairs in either the source or destination orientation.

- **Rule options**

The rule option section contains alert messages and information about which parts of the packet should be inspected to determine if the rule action should be taken. All *Snort* rule options are separated from each other using the semicolon. Rule option keywords are separated from their arguments with a colon. There are four major categories of rule options, namely, general rule options, payload detection rule options, non-payload detection rule options and post-detection rule options [Sno10]. General rule options provide information about the rule but do not have any affect during detection. Payload detection rule options look for data inside the packet payload and can be inter-related whereas non-payload detection rule options look for non-payload data. Post-detection rule options are rule specific triggers that happen after a rule has fired.

Snort rules can be classified into three categories depending upon criteria to perform the analysis on network data. These categories are signature detection rule, protocol detection rule and anomaly detection rule. The example *Snort* rule [Sen06] is shown in (2.1).

```
alert tcp $HOME_NET any -> any 80 (msg: “Match Found”;content: “|47 45 54|” ;)
```

(2.1)

Rule Header Rule Option

Here, elements before parentheses comprise ‘rule header’, whereas elements in parentheses are ‘rule options’.

2.4.2 Bro

Bro [Pax99] is a passive, open source network traffic analyzer. It is primarily a security monitor that inspects all traffic on a link in depth for signs of suspicious activity. *Bro* originated as a research system, designed and developed by Vern Paxson of *ICSI’s Center for Internet Research (ICIR), Berkeley*. The project started in 1995 at *Lawrence Berkeley National Laboratory (LBNL)*, and *Bro* was first published in 1998.

Bro’s fundamental design goal is to separate mechanism and policy. *Bro* is neither fundamentally anomaly based nor misuse based. It supports both approaches and default policy scripts shipped with *Bro* implement examples of both anomaly as well as misuse detection. Its may use *Snort*’s signatures by means of a converter called as *Snort2Bro*.

2.4.2.1 Architecture of Bro

Bro is conceptually divided into an event engine that reduces a stream of packets to a stream of higher level network events and an interpreter for a specialized language that is used to express a site's security policy. The architecture of *Bro* is structured in layers, as shown in Figure 2.2. The lower most layers process the greatest volume of data whereas in higher layers, the data stream diminishes, allowing for more processing per data item.

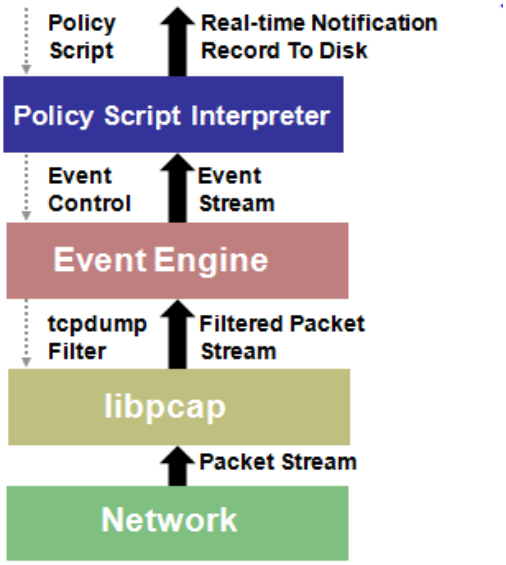


Figure 2.2: Architecture of *Bro* [Pax99]

It has three major components, namely, *libpcap*, event engine and policy script interpreter. The brief description about these components is discussed in the following part of this subsection.

- **libpcap**

Libpcap [Jac09], the packet capture library used by *tcpdump*, helps *Bro* to capture and filter network traffic to analyze it. The filtering of traffic is important as it reduces overhead of looking inside the packets which are of no interest. Both online as well as offline analysis are possible because it can save the captured packets in *tcpdump* format for later use.

- **Event engine**

The filtered packet stream is then fed to the next layer called event engine. This layer first performs several integrity checks to assure that the packet headers are well formed failing which the packets will be discarded. After reassembling IP fragments for the packets which passed the integrity check, connection state is checked. After processing each event, the event engine checks for new events kept in a FIFO queue for further processing.

- **Policy script interpreter**

The policy script interpreter executes scripts written in the specialized *Bro* language. These scripts specify event handlers similar to *Bro* functions except that they do not return a value. For each event interpreter retrieves compiled code for the corresponding handler and executes arbitrary *Bro* scripting commands. These include generating new events, logging real time notifications, recording data to disk, or modifying internal state for access by subsequently invoked event handlers. *Bro*'s major feature is that it separates policy from mechanism.

This research work is focused on attack detection and signature generation. So far, many attack detection and signature generation techniques and systems have been proposed in literature. The following sections elaborate the taxonomy, techniques and available systems related to attack detection and signature generation.

2.5 Attack Detection and Signature Generation

Zero day discoveries need the defenders to be on guard always but this is not feasible for manually interfaced systems. Automatic attack detection and signature generation systems assist Intrusion Detection Systems to capture and report zero day attacks. This section describes the general taxonomy and various techniques proposed in literature used for signature generation.

- **Signature**: A signature for a specific activity is a pattern recipe which is chosen for uniquely identifying an associated activity on a system with a property that it should match only the activity it describes [All07].

- **Attack Detection**: Attack detection is the process of observing a system in order to separate normal activities from those representing an attack. The process defines normal activities either explicitly or implicitly [Res06].
- **Signature Generation:** Signature generation is the process of creating a signature of virus, worms, Trojans, malware and attacks.

2.5.1 Techniques Used for Signature Generation

There are various information extraction techniques for signature generation proposed in literature. The following enlisted and briefly explained techniques have been used in one or more signature generation systems.

- **Byte Frequency Distribution (BFD)**

  The byte frequency distribution is a function *fp(b),* which describes the probability of *b* to appear at position *p*. The sum of *fp(b)* for all possible bytes is 1. If *X* is the width in bytes of the signature, $(f_1, f_2, .. , f_X)$ represents a simple signature based on BFDs [McD03], [Kau02].

- **Longest Common Substring (LCS)**

  Let string *X* of length *m* represented as *X(1…m),* and string *Y* of length *n* represented as *Y(1..n),* longest common substring finds the longest string (or strings) that is a substring (or are substrings) of two or more strings.

- **Longest Common Subsequence (LCSeq)**

  The problem of the longest common subsequence (LCSeq) is to find a longest sequence which is a subsequence of all sequences in a set of sequences. The problem is NP hard for the general case of an arbitrary number of input sequences. When the number of sequences is constant, the problem is solvable in polynomial time by dynamic programming.

- ***N*-Gram Analysis**

  N-Grams [Cav94] are used to describe objects as vectors. Let *X* be an alphabet, |*X*| is the cardinal number of the alphabet and *n* be an integer. An *N*-gram is a word of length *n.* If number of occurrences of an *N*-gram can be determined within a possible sequence of characters of an alphabet then it is possible to get *N*-gram frequency vector for that sequence.

- **Rabin Fingerprint**

  Fingerprints are just like hash tags. These are shorter bit strings for larger data. Rabin fingerprinting is a method for implementing public key fingerprints using polynomials over a finite number of elements [Rab81]. Rabin fingerprints are the basis of the Rabin-Karp algorithm [Kar87], one of the fastest known string searching algorithms. Rabin fingerprints are very efficient over a sliding window, as they allow for a new sliding window hash to be computed based on the previous hash computation. These are used by many known signature generation systems.

- **Content based Payload Partitioning (COPP)**

  The COPP [Kim04] algorithm is based on Rabin fingerprints that search for repeated byte sequences by partitioning the payload into content blocks. This enables the description of payload content that is tolerant of payload variability to some degree.

- **CMENW Algorithm (Contiguous Matches Encourages Needleman-Wunsch)**

  CMENW algorithm [Tan06] is a pair wise alignment algorithm based on global alignment. It is improved on Needleman-Wunsch algorithm [Nee70], which is the typical pair wise alignment algorithm.

- **HMSA Algorithm (Hierarchical Multi Sequence Alignment)**

  HMSA algorithm [Tan06] is a type of hierarchical multi sequence alignment algorithm based on pair-wise alignment CMENW algorithm, suitable for attack signatures generation. This algorithm has three main features, namely, hierarchical pair wise alignment, supporting wildcard characters and with a pruning function. HMSA algorithm has accelerated convergence and enhanced noise resisting ability because of pruning function.

- **Smith-Waterman Algorithm**

  Smith-Waterman algorithm is a pair wise local alignment algorithm put forward by Smith and Waterman [Smi81], which is used to find and compare the

similarity in local regions in an overall view. Smith-Waterman algorithm is used to find the biggest similarity value and alignment based on the principle of dynamic programming. Its process includes two major steps. In the first step, similarity value of two given sequences is calculated and a similarity matrix is generated. In second step, the best results of sequence alignment are obtained through dynamic programming and backtracking algorithm, according to the similarity matrix.

- **GASBSLA Algorithm (Generation of Attack Signatures Based on Sequence Local Alignment)**
  It is a pair wise local alignment algorithm. GASBSLA [Nan08] algorithm uses the idea of local alignment and affine empty penalty model to improve the generality of attack signatures, so that it can detect polymorphic attack more effectively.

- **TGMSA Algorithm (Tri-stage Gradual Multi-Sequence Alignment)**
  It is a tri-stage gradual multi-Sequence alignment algorithm. TGMSA algorithm proposed by [Nan08] presents a new pruning policy to make the algorithm more insensitive to noises in the generation of attack signatures.

Signatures based intrusion detection is the most effective solution for the attack reporting, but the continuous emergence of new types of attacks and polymorphic worms is a great challenge to the existing intrusion detection technologies. To solve this problem, automatic generation of attack signatures has been concerned by most researchers and has become a new hotspot in intrusion detection. There are various algorithms which have been devised to automate the generation of attack signatures. The following section discusses the various existing automated defense systems which are capable of attack detection and signature generation.

2.6 Existing Automatic Attack Signature Generation Systems

Detection and prevention of network attacks has always been a topic of interest of various researchers. There are many manual and automated systems available in the literature

[War05]. Various reactive and proactive methods have been proposed by researchers for detection and signature generation of variety of attacks and worms, including polymorphic worms. These systems with architectural and implementation details are discussed in the following subsections.

2.6.1 Honeycomb

Honeycomb [Kre04] is considered to be the first approach aiming at the automated generation of attack signatures. This system is built as an extension to *honeyd*, an open source, low interaction honeypot. This system generates signatures for malicious network traffic automatically by using pattern detection techniques. It uses Longest Common Substring (LCS) algorithm to spot similarities in packet payloads. The system leads to large number of false positives when long identical byte sequences are in use. *Honeycomb* produces signatures compatible with *Bro* and *Snort* NIDS. Honeycomb creates signatures for Slammer and CodeRed II worms but is less successful in producing signature of polymorphic worms which change their signatures frequently.

2.6.2 Autograph

Autograph [Kim04] is a system capable of network level attack detection with automatic generation of signatures for novel Internet worms propagating through TCP transport. *Autograph* uses a port scan based flow classifier to differentiate between innocuous (harmless) and suspicious flow. This technique reduces the volume of traffic on which it performs content prevalence analysis to generate signatures. If some external host attempts to connect unreachable targets in the network more than *n* number of times, it is marked by the *Autograph* system as a scanner and stored in a scanner list. All incoming traffic from this host will be considered as suspicious flow pool. The scanner list is refreshed periodically so that most recent flow pool can be considered for signature generation.

2.6.3 Early Bird

Earlybird [Sin04], [Sin03] is an automated approach for realtime detection of unknown worms and viruses. This system could automatically extract unique content signatures using content sifting approach in which Rabin fingerprints for all possible substrings of a certain length in each packet is computed. This system uses address dispersion in order to detect the actual worm traffic from the benign one to reduce false alarms. The system has low memory and CPU requirements. It generates *Snort* compatible signatures. *Earlybird* could successfully detect and extracted signatures for unknown worms like Blaster, My-Doom and Kibuv.B, variants of CodeRed and Sasser worms along with some of the known worms. This system is not suitable for detection of polymorphic worms. It had some false positives but no false negatives.

2.6.4 PAYL

PAYL (Payload-Based Anomaly Detector) [Wan04], is a payload-based anomaly detection sensor that detects inbound anomalous payloads, and correlates them with outgoing traffic on same ports. It uses unsupervised machine learning techniques to model normal traffic profile of a host. The system first goes through a training phase in which profile byte frequency distribution is computed. Then, standard deviation of the application payload flowing to a single host and port is calculated. Thereafter in the detection phase, Mahalanobis distance is calculated to judge the similarity of new data against the pre-computed profile. If the distance of the new input exceeds a threshold value, an alert is generated. The system was tested on the 1999 DARPA IDS dataset and a live dataset collected on the Columbia CS department network. The authors claimed nearly 100% accuracy with 0.1% false positive rate for port 80 traffic. The system could also successfully detect the Code Red II and a buffer overflow attack from the unlabeled CUCS dataset. This system was further improved [Wan05] by adding some new features to anomalous payload detection sensor to accurately detect and generate signatures for zero-day worms.

2.6.5 COVERS

COVERS (COntext-based, VulnERability-oriented Signature) [Lia05a], uses a forensic analysis of a victim server's memory to develop correlation between attacks and inputs received over the network. It automatically develops a signature that characterizes inputs which carry attacks. The architecture of *COVERS* is shown in Figure 2.3. This approach provides effective protection against majority of attacks that exploit memory errors in C/C++ programs. It also does not require access to source code of the protected server. These signatures can then be used to filter out future occurrences of these attacks in order to preserve the integrity and availability of the service.

COVERS has an attack detection module and a signature generation module. The attack detection part employs the Address Space Randomization (ASR) technique. Signature generation module generates signatures by comparing malicious data with data appearing in the same field in benign messages. The correlation step identifies the specific network packet or flow involved in an attack, and the bytes within this packet that were responsible for triggering the alert. Thereafter, the input context is identified. An input context is a particular field of a specific type of message.

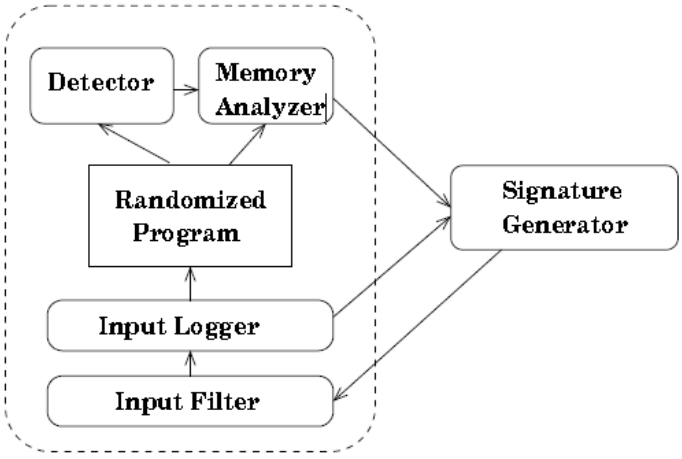


Figure 2.3: Architecture of *COVERS* [Lia05a]

After the field has been identified, the characteristics of the underlying vulnerability are extracted. The reference values are continuously updated using benign input traffic. The

generated signature consists of message format specifier, message field carrying the exploit and thresholds for characteristics. Authors claim that this approach had successfully generated signatures for the attacks with very low overheads and without producing false positives.

2.6.6 ARBOR

ARBOR [Lia05b] is a system for automatic generation of buffer overflow attack signatures. This approach is based on program behavior model. They argued that most existing techniques of buffer overflows detection leads to repeated restarts of the victim application causing the unavailability of their services. *ARBOR* filters out attacks before they compromise the integrity of a server, thereby allowing the server to continue to run without interruption. This significantly increases the availability of servers subjected to repeated attacks. Figure 2.4 shows the architecture of *ARBOR*. It is implemented using in-line and off-line components. In line components are responsible for input filtering and logging purpose whereas the off line components perform tasks such as detection, analysis and signature generation. The input filter intercepts all input actions of the protected process and compares them with the list of signatures currently deployed in the filter. Inputs matching any of these signatures are discarded.

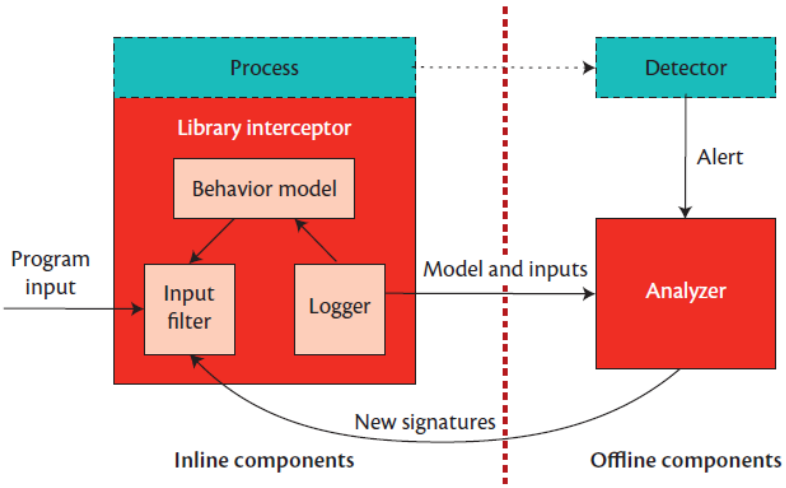


Figure 2.4: Architecture of *ARBOR* [Lia05b]

The behavior model is a central component of *ARBOR* used for making automatic filtering decisions based upon the knowledge gathered from the program itself. Library interception is used to learn the behavior model of a protected process. The logger records inputs for offline analysis. It also saves the entire behavior model periodically to the disk, so that the model does not have to be rebuilt from scratch on process restarts. The off-line components include a detector and an analyzer. The detector is responsible for attack detection. It promptly notifies the analyzer, which begins the process of generating an attack signature. The generated signature is then deployed in the input filter. This enables future instances of the attacks to be dropped before they compromise the integrity or availability of the protected process.

The system does not have any false positives. However, the system reported some issues of false negatives, like attacks delivered through multiple packets, concurrent servers, message field overflows, Denial of Service (DoS) attacks aimed at evading character distribution signatures and addressing limitations.

2.6.7 Polygraph

Polygraph [New05b] is a system which generates signatures for polymorphic worms. *Polygraph* generates signatures that consist of multiple disjoint content substrings corresponding to various parameters like protocol framing, return addresses *etc.* *Polygraph* defines three different signature types, namely, conjunction signature, token subsequence and Bayes signature. All these signatures are built from substrings, called tokens. All network traffic monitored by *Polygraph* is passed through a flow classifier. After reassembly of incoming flow in contiguous byte flows, the traffic is classified as innocuous or suspicious by analysing parameters like IP protocol number and port as shown in Figure 2.5.

The false positive and false negative rate of the system depends upon the number of worm samples. If the number of samples is less the false negatives are high for conjunction and token subsequence signatures. However, number of samples does not have any effect on false negatives in the case of Bayes signatures.

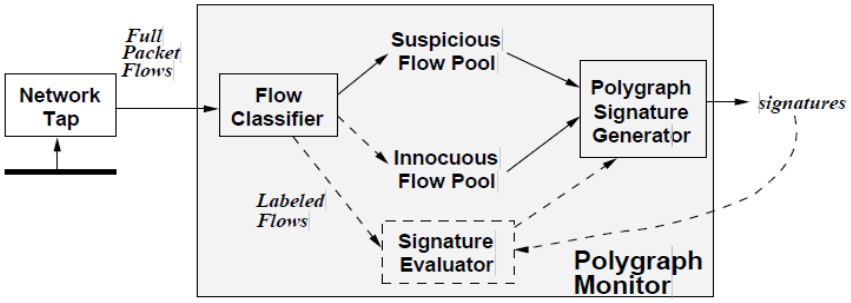


Figure 2.5: Architecture of *Polygraph* [New05b]

The false positive rate also increases with the introduction of noise in the innocuous flow pool. This approach is not well suited for network zero day attacks, for which multiple sample data is not available. The system could generate the signatures for Apache-Knacker, ATPhttpd and BIND-TSIG exploits.

2.6.8 Nemean

Nemean [Yeg05] is a system for automatic generation of intrusion signatures from honeynet packet traces. *Nemean* uses automata learning approach to create protocol semantic aware signatures. There are two major components in this system, namely, Data Abstraction Component and Signature Generation Component (Figure 2.6). The Data Abstraction Component (DAC) aggregates and transforms the packet trace into a well defined data structure suitable for clustering. The signature generation component is responsible for connection and session clustering with similar attack profiles. These signatures can be made compatible with *Bro* and *Snort* IDS.

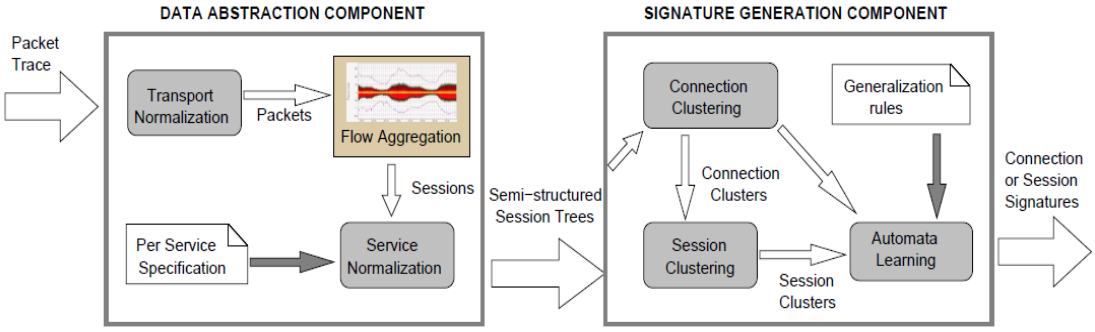


Figure 2.6: Architecture of *Nemean* [Yeg05]

As claimed by authors, *Nemean* generated accurate signatures with extremely low false alarm rates for a wide range of attack types, including buffer overflows attack called Welchia, attacks with large diversity like Nimda, and attacks for complicated protocols like NetBIOS/SMB. Signatures generated by *Nemean* for NetBIOS exploits had zero false positive rate and a 0.04% false negative rate.

2.6.9 PADS

PADS (Position-Aware Distribution Signatures) [Tan05] is a signature generation system for the detection of polymorphic worms. It implements a double honeypot system, which is able to automatically isolate the attack traffic and detect new worms. It uses two algorithms based on Expectation-Maximization (EM) [Law90] and Gibbs Sampling [Law93] for efficient computation of signatures from polymorphic worm samples. These algorithms could accurately separate new variants of the MSBlaster worm from the normal background traffic. The authors claimed 100% accuracy in detection of variants of MSBlaster worm with no false positives.

2.6.10 TaintCheck

TaintCheck [New05a] is a mechanism that uses dynamic taint analysis, for the automatic detection, analysis, and signature generation of exploits on commodity software. It does so by performing binary rewriting at run time without a need of source code or specially compiled binaries. Its two tiered hybrid detector approach provides automatic analysis and input correlation of exploit attacks with low performance overhead. It enables automatic semantic analysis based signature generation resilient against polymorphic and metamorphic worm attacks. It also provides a feature of signature or alert verification. *TaintCheck* is implemented using *Valgrind* [Net03], an open source x86 emulator that supports extensions which can instrument a program while it is running. It can reliably detect most overwrite attacks with no known false positives.

2.6.11 Vigilante

Vigilante [Cos05] is an end-to-end approach to contain fast spreading worms using collaborative worm detection at end hosts. It includes an automatic filter generation

component to protect the end hosts from subsequent attacks and an overlay network for fast alert distribution. *Vigilante* introduced the concept of Self Certifying Alerts (SCAs). An SCA can be used as a proof for the existence of vulnerability in a service that can be verified in an automated way. If any host is vulnerable, the host can generate a filter to protect itself from the attack that triggered SCA generation. Otherwise, this SCA could be safely ignored. A *Vigilante* host consists of components, namely, detector, SCA verifier, a filter generator and an alert distributor as shown in Figure 2.7.

Attack detector is responsible for generation of SCAs using Non-eXecutable (NX) pages and dynamic data flow analysis. SCA verifier verifies SCAs using virtual machines to avoid any side effects. Filter generation is used for generation of a general and specific host based filter for every new SCA that was successfully verified. Alert distributor enables fast, resilient and secure distribution of SCAs to all other hosts using flooding. *Vigilante* could contain real worms like Slammer, Blaster, CodeRed, and polymorphic variants of these worms.

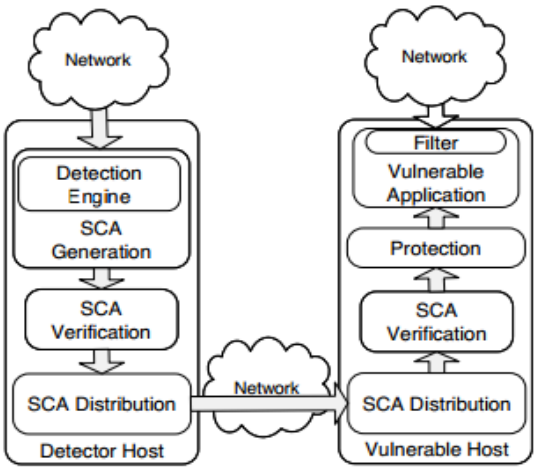


Figure 2.7: Automatic worm containment in *Vigilante* [Cos05]

2.6.12 Argos

Argos [Por06] is an emulator for fingerprinting zero day attacks. *Argos* is a containment environment that can handle worms as well as human launched attacks. *Argos* is built upon a fast x86 emulator which tracks network data throughout execution to identify

invalid jump targets, function addresses and instructions. Figure 2.8, depicts the architecture of *Argos*. Incoming traffic is logged in a trace database, and fed to the unmodified application running on emulator. In the emulator a dynamic taint analysis is deployed to detect when vulnerability is exploited to alter an application's control flow.

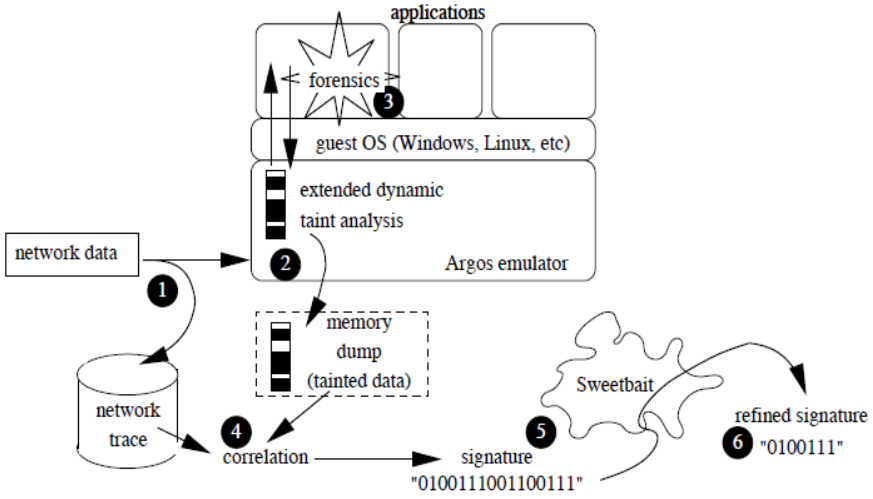


Figure 2.8: Architecture of *Argos* [Por06]

Argos traces physical addresses rather than virtual addresses minimizing the memory mapping problems. When a violation is detected, an alarm is raised which leads to a signature generation phase. To aid signature generation, *Argos* first dumps all tainted blocks and some additional information to file, with markers specifying the address that triggered the violation, the memory area it was pointing to, *etc.* To know about additional information about the application like process identifier, executable name, open files and sockets *etc.*, the system inject its own shell code to perform forensics. *Argos* submits the signature to a subsystem known as *SweetBait* [Por07], which correlates signatures from different sites, and refines signatures based on similarity. The final step is the automated use of the refined signature. *Snort* is attached to *SweetBait* to provide the signatures of traffic. The system has used to *Aho-Corasick* pattern matching algorithm [Aho75] to match network signatures. Argos was used by NoAH [Koh09] test bed project as virtual machine emulator.

2.6.13 Hamsa

Hamsa [Li06] is a network based automated signature generation system for polymorphic worms. *Hamsa* is the Sanskrit word for the swan bird which is believed to be able to separate the milk from a mixture of milk and water. *Hamsa* is fast, noise tolerant and attack resilient. This is a model which analyzes the invariant content of polymorphic worms. Architecture of *Hamsa* is similar to the basic frameworks of *Autograph* [Kim04] and *Polygraph* [New05b]. Figure 2.9, shows the architecture of *Hamsa*.

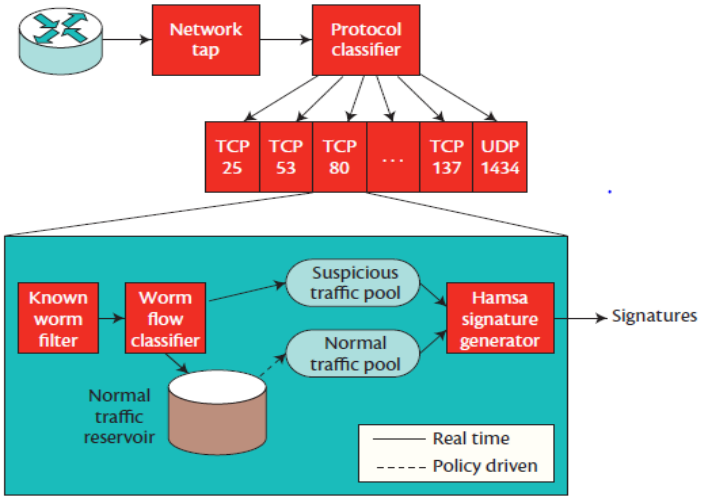


Figure 2.9: Architecture of *Hamsa* [Li06]

It sniffs the traffic from networks, assemble the packets to flows, and classify the flows based on different protocols like TCP, UDP, ICMP and port numbers. Then, for each protocol and port pair, *Hamsa* filters out the known worm samples and then separates the flows into the suspicious pool and the normal traffic reservoir using a worm flow classifier. Then based on a normal traffic selection policy some part of the normal traffic reservoir is selected to be the normal traffic pool. The signature generator then generates the signatures given suspicious and normal traffic pool. *Hamsa* focuses on content based signatures because they treat the worms as strings of bytes and does not depend upon any protocol or server information. The signature generated by *Hamsa* can be easily deployed at IDSs such as *Snort* or *Bro*. *Hamsa* significantly outperforms *Polygraph* in terms of efficiency, accuracy, and attack resilience.

2.6.14 Honeycyber

Honeycyber [Moh08] is an automated signature generation for zero day polymorphic worms. *Honeycyber* has double honeynet system. Figure 2.10 shows the architecture of *Honeycyber*. The packets containing worms come to first honeynet and make outbound connections. These are redirected to the second honeynet by the internal translator. These worms make outbound connections from second honeynet too and are redirected to the first honeynet. The legitimate packets do not make outbound connections. So, the packets that make outbound connections are considered malicious.

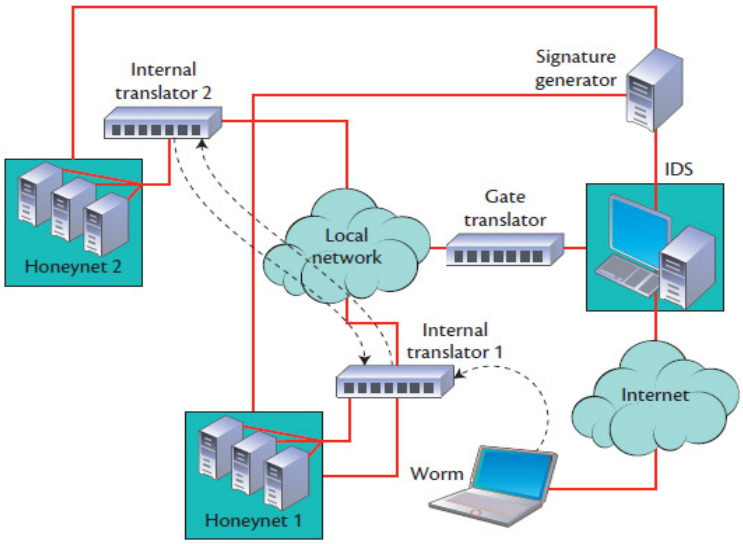


Figure 2.10: Architecture of *Honeycyber* [Moh08]

This system is able to detect polymorphic worms also. This procedure captures different payloads of the same worm, which helps to generate high quality signatures for a worm. The signature generation is based on longest common substring method applied on multiple invariant substrings. *Honeycyber* is able to automatically detect new worms and isolate the attack traffic from innocuous traffic. The signatures generated by *Honeycyber* are both *Snort* and *Bro* based. The system is able to generate signatures to match most polymorphic worm instances with low false positives and low false negatives.

Mohammed *et al.* (2010) have extended this architecture using Principal Component Analysis (PCA) [Jol05] to determine the most significant substrings that are shared between polymorphic worm instances so as to use them as signatures [Moh10].

2.6.15 Eudeamon

Eudaemon [Por08], is an on-demand emulation against zero day exploits that uses dynamic taint analysis for attack detection. *Eudaemon* is able to attach to any running process, and redirect execution to a user space emulator that will dynamically instrument the binary by means of taint analysis. Any attempts to subvert control flow, or to inject malicious code will be detected and averted. *Eudaemon* is able to automatically generate signatures for NIDSs, similarly to Argos. When employed in a significant number of nodes, the timely generation of signatures can protect entire communities of systems.

2.6.16 Hancock

Hancock [Gri09] is a system for automatic generation of string signatures for malware detection. *Hancock* is the first automatic string signature generation system developed in Symantec Research Labs that automatically generates high quality string signatures with minimal false positives and maximal malware coverage.

Hancock uses string signatures, each of which is a contiguous byte sequence that can match many variants of a malware family. It uses three types of heuristics to test a candidate signature's false positive rate. These are probability based, disassembly based, and diversity based. Probability and disassembly based heuristic is used to filter candidate signatures extracted from malware files and the diversity based heuristic is used to select good signatures from among these candidates. *Hancock* begins by recursively unpacking malware files and then examining every 48-byte code sequence in unpacked malware files. It filters out byte sequences whose estimated occurrence probability in good ware programs, according to a pre-computed good ware model, is above a certain threshold. These byte sequences can be a part of library functions or whose assembly instructions are not sufficiently unique. *Hancock* further applies a set of selection rules based on the diversity principle on the candidate signatures that have passed the initial filtering step. *Hancock* finally generates string signatures that consist of multiple disjoint byte sequences rather than only one contiguous byte sequence. *Hancock* is able to automatically generate string signatures with a false positive rate below 0.1%.

2.6.17 ZASMIN

ZASMIN (*Zeroday-Attack Signature Management Infrastructure*) [Kim09] is a system for novel network attack detection. This system provides early warning at the moment the attacks start to spread on the network and to block the spread of the cyber attacks by automatically generating a signature that could be used by the network security appliance such as IPS. This system has adopted various technologies for unknown network attack detection. These are suspicious traffic monitoring, attack validation, polymorphic worm recognition and signature generation. In order to check the feasibility of the validation functions in *ZASMIN*, it was installed and verified on real honeynet environment.

2.6.18 Nebula

Nebula [Wer09] is a system for generation of intrusion signatures from syntactical information in attack traces. This system works in a two step approach. In the first step, similar inputs are identified and clustered to represent groups of similar attacks as resulting clusters. Thereafter, common features are extracted from the elements of a cluster. These features are finally assembled to a signature. It uses generalized suffix tree in conjunction with longest common substring for information retrieval. Positional and ordering information is included for the individual parts of a signature. *Nebula* is able to compute signatures for previously unknown attacks by requiring no other knowledge than syntactical structure. The authors claimed that nebula could successfully generate signatures automatically for novel threats, *e.g.*, the exploit used by *Conficker* worm. It is an open source tool and is available in the package repository of Fedora Linux.

2.6.19 Auto-Sign

Auto-Sign [Tah10] is a real time system for extracting unique signatures of malware executables to be used by high speed malware filtering devices based on deep packet inspection. It enables analysis at the binary level and does not require a semantic interpretation of code making this technology generic. It can be used on large size malware by disregarding signature candidates which appear in benign executables. *Auto-Sign* has two phases, namely, setup phase and signature generation phase. In setup phase two data structures, namely, Common Function Library (CFL) and Common Threat

Library (CTL) are created. In signature generation phase signatures are generated, trimmed, ranked and final signature is chosen on the basis of entropy. It uses 3-gram representation for the signatures. The main benefit of this method is that it enables analysis at the binary level and does not require a semantic interpretation of code into function blocks. *Auto-Sign* need to follow a more exhaustive and systematic methodology for building CFL repository in generating signatures for high throughput network security appliances.

2.6.20 F-Sign

F-Sign [Sha11], an automatic function based signature generation for malware files, is designed to generate simple, byte string, signatures that can be used by network based Intrusion Detection Systems for filtering malware in real time. *F-Sign* employs an exhaustive and structured technique, which first extracts the malware's unique code from other segments of common and usually benign code, such as shared libraries. This system generates signatures from malware such as worms, spyware, Trojan horses, and viruses. The suspected files are classified as benign or malicious by a human expert or by an automated detection tool. In this system, a Common Function Library (CFL) is created first, that contains representation of functions from standard libraries used by higher level languages. After this the signatures are generated for the entire malware corpus. The CFL creation and signature generation process of *F-Sign* is shown in Figure 2.11. The CFL can either be created by extracting functions using file disassembly by using specialized state machine. Signature generation process begins with the identification of the internal functions thereby matching these functions against the database of existing functions stored as CFL. Those which do not exist in the CFL are the candidates for generating unique malware signatures.

The best candidate is chosen by entropy based on the length of candidate in bytes and frequency of appearances of a specific byte in the candidate function. The one with the highest entropy is selected. *F-Sign* was used as the automatic signature generation module of the *eDare,* an early detection, alert, and response framework, which provides malware filtering service to network service providers, Internet Service Providers and

large enterprises. This system has low false positives for longer signature candidates and for larger CFL.

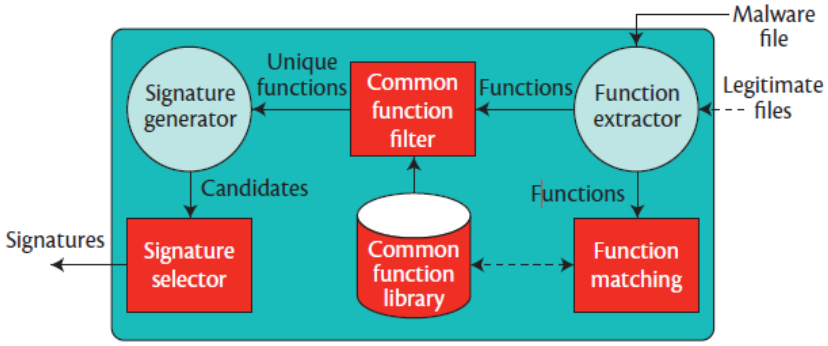


Figure 2.11: Architecture of *F-Sign* [Sha11]

2.6.21 Honeyfarm based defense against Internet worms

Jain and Sardana [Jai12] proposed a hybrid approach that integrates anomaly and signature detection with honeypots. The architecture of the system is shown in Figure 2.12. Signature based detection is used as the first level filter to detect known worm attacks. At the second level an anomaly detector is placed to detect any deviation from the normal behavior. In the last level honeypots are deployed which help in detecting zero day attacks. The controller is responsible for the traffic redirection among various honeypots which are deployed in honeyfarm. In this approach longest common subsequence algorithm is used to generate signatures. The detection rate of hybrid honeyfarm based approach is 81% with a false alarm rate of 4%.

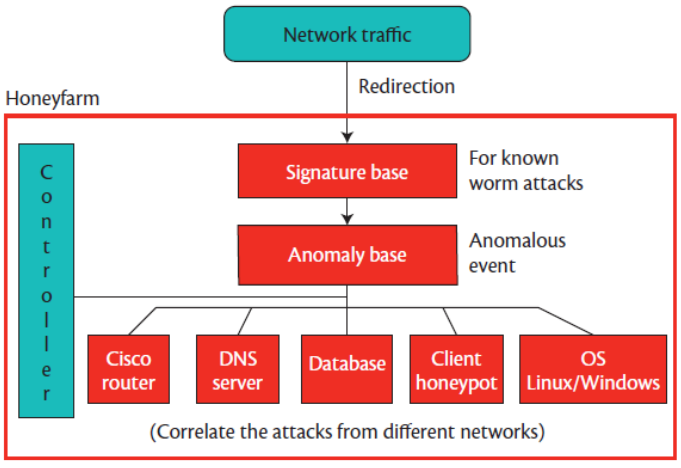


Figure 2.12: Architecture of Hybrid Honeyfarm based approach [Jai12]

Many other researchers have contributed to this area of research by proposing systems based upon variety of contexts and techniques for attack detection and signature generation. Some more systems falling in this category include *DOME* [Rab03], *HoneyStat* [Dag04], *DIRA* [Smi05], *HoneyAnalyzer* [Tha05], *LESG (Length-based Signature Generator)* [Li07], *ShieldGen* [Cui07], vulnerability based signature generation system [Bru08], *PolyTree* [Tan11] and regular expressions based signature generation system proposed by [Wan12]. These systems have also been proposed as a way to automatically detect attacks to generate attack signatures.

2.7 Comparative analysis of existing attack Signature Generation Systems

The comparative analysis of existing attack detection and signature generation systems has been carried out on the basis of important parameters, namely, methodology for signature generation, type of signatures generated, attacks /worms detected and false alarm rates. The analysis has been summarized in Table 2.1.

Table 2.1: Comparative analysis of automated signature generation systems under study

Sr. No	Parameter → System↓	Year of Publication	Authors	Methodology for Signature Generation	Type of Signatures generated	Attacks/Worms covered	False Alarms Rate
1	*Honeycomb*	2004	Kreibich *et al.*	LCS	*Bro* and *Snort*	Code Red II and Slammer	Low
2	*Autograph*	2004	Kim *et al.*	Rabin Fingerprints, content-based payload partitioning (COPP)	*Snort* supported	Code-RedIv2, Code-RedII and Nimda,	Low
3	*Earlybird*	2004	Singh *et al.*	Invariant Bytes	*Snort* Supported	Blaster, My-Doom and Kibuv.B, variants of CodeRed and Sasser worms	Low false positives zero false negatives
4	*PAYL*	2004	Wang and Stolfo	Machine learning of byte frequency distributions	Continuous and dis-continuous byte pattern	Code Red I & II, Nimda, crashIIS, back, apache2 and buffer overflow attack	0.1% false positive rate
5	*COVERS*	2005	Liang and Sekar	Address Space Randomization	Message type/field characteristics, *Snort* compatible	Ntpd, Samba, passlogd, epic4 and gtkftpd Stack Overflow, wu-ftpd format string, apache mod SSL Heap Overflow	No false positives
6	*ARBOR*	2005	Liang and Sekar	Address Space Randomization	Buffer overflow related signatures	10 real world vulnerabilities including wu-ftpd, apache SSL, ntpd, ircd, samba, Passlogd	No FPs, FNs in case of Fragmented attacks, Concurrent servers, Message Fields overflows

7	*Polygraph*	2005	Newsome *et al.*	Token Subsequence (multiple invariant bytes)	Bayes, Token Subsequence and Conjunction	Polymorphic worms	Depends upon Number of worm samples
8	*Nemean*	2005	Yegneswaran *et al.*	Clustering technique, Automata learning	*Snort* and *Bro*	Buffer Overflows (Welchia), Nimda, NetBIOS/SMB exploits	Zero false positive rate and a 0.04% false negative rate
9	*PADS*	2005	Tang and Chen	Statistical anomaly based, Redirecting outgoing connections using double honeypot	Byte frequency distribution based signatures	Polymorphic malware variants of MSBlaster	No false positives
10	*TaintCheck*	2005	Newsome and Song	Dynamic Taint Analysis	Can be used for automatic signature generation based upon semantic analysis	ATPhttpd Buffer Overflow, cfingerd, wu-ftpd	Low
11	*Vigilante*	2005	Costa *et al.*	Non-eXecutable (NX) pages and dynamic data flow analysis	Self Certifying Alerts (SCAs)	Slammer, Blaster, CodeRed, and polymorphic variants of these worms	Low
12	*Argos*	2006	Portokalidis *et al.*	Dynamic Taint Analysis, Longest Common Sub string, and Critical Exploit String Detection	*Snort* based signatures	Scalper, sadminD/IIS, Welchia, Poxdar, Sasser, Gaobot.ali, Zotob, Blaster, Mytob-CF, Dopbot-A	No false alarms

13	*Hamsa*	2006	Li *et al.*	Content Based Token Extraction	*Snort* and *Bro* based signatures	Code-Red II, Apache-Knacker, ATPhttpd, CLET, TAPiON	Low and Bounded FPs and FNs
14	*Honeycyber*	2008	Mohammad *et al.*	Colored Set Size for String Matching	*Snort* and *Bro* based signatures	Polymorphic worms	Low false positives
15	*Eudeamon*	2008	Portokalidis and Bos	Taint Analysis, Longest Common Substring, and Critical Exploit String Detection	*Snort*, *SweetBait*	Scalper, sadminD/IIS, Welchia, Poxdar, Blaster, Zotob, Sdbot	Low
16	*Hancock*	2009	Griffin *et al.*	Uses probability based, disassembly based, and diversity based string signatures with contiguous byte sequence	Single component and Multi component signatures for Anti Viruses	Malware detection	Sufficiently low false positives (Below 0.1%)
17	*Zasmin*	2009	Kim *et al.*	Longest Common Substring	*Snort* signatures	MS06-040 10, MS05-039, MS04-011 16, MS04-007 9, MS04-026 3, MS03-039 3. LSASS Vulnerablities related attacks	High
18	*Nebula*	2009	Werner *et al.*	Clustering and Generalized Suffix Tree based Longest Common Substring	*Snort* Compatible	Novel worms such as Conficker	Low false positives
19	*Auto-Sign*	2010	Tahan *et al.*	By ranking the candidate signature based on entropy, probability and distance	Signatures compatible with NIDS/NIPS operating as malware filtering devices	Malware executables including worm emails, virus, Trojans, Email flooder, DOS attack, exploits, worms, P2P attacks	Low false positives

20	*F-Sign*	2011	Shabtai *et al.*	Entropy based selection after creating CFL (Common Functions Library)	Compatible with eDare (early detection, alert, and response) framework	Malware Detection	Low FPs Provided CFL size is large (Below 0.4% with 1675 MB CFL)
21	*Hybrid Honeyfarm based approach*	2012	Jain *et al.*	Longest Common Subsequence along with protocol based packet header anomaly detection technique	Local signature detection and generation engine	Metasploit generated attack patterns	4% false alarms rate

2.8 Research Motivation

Many methods and tools have been developed to secure network resources and communication over Internet. Intrusion detection is one such method. Intrusion Detection System (IDS) monitors the state of a system or network, recognize and report any malicious activity or improper behavior. IDS can use either anomaly based or misuse/signature based technique to detect intrusions. Most network based Intrusion Detection Systems use signature detection technique which has a drawback that it does not secure the system fully. The problem with signature based technique is that it cannot detect attacks whose signatures are not available in repository. Novel attacks which are not detected by these IDSs can lead to severe damage for network and its resources. These are called zero day attacks. These attacks are difficult to detect both by proactive as well as reactive security approaches. Anomaly based detection technique is able to report about these attacks, but it results in high number of false alarms. Zero day discoveries need real time attack handling and response, but this is not feasible for manually interfaced systems. So, some automated defence mechanisms are required to save against these attacks. A technique is needed for the online detection and signature generation of such attacks so that valuable information on resources like web and email servers can be secured well in advance before these attacks can hit the network. There is a need of a defense mechanism to protect from HTTP and SMTP related attacks. This mechanism needs to incorporate automatic attack detection and signature generation process. The focus of this research is to propose and validate a network security model that works in a proactive manner and generates the attack signatures for these novel attacks.

2.9 Objectives of the Proposed Work

The main objective of this study is to design and develop a network security model for attack signature generation. In order to perform this task, following objectives were proposed to be carried out.

i) To propose a network security model for attack signature generation in context with HTTP and SMTP network traffic.
ii) To design and implement the proposed model to minimize zero day attacks.
iii) To verify the proposed model for tracking and analysis of attacks.

2.10 Major Contributions of Thesis

Major contributions of this research work have been enlisted below.

- In this thesis, extensive study of existing tools and techniques for intrusion detection and automated signature generation systems along with their comparative analysis has been done under literature review.
- The work carried out in this thesis is focused on automatic attack signature generation, tracking and analysis. For the accomplishment of proposed objectives, a proactive hybrid framework for attack detection and signature generation has been designed, implemented and tested.
- A major outcome of this research work is a proposed framework named as HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator) that uses a hybrid approach for detection using both misuse detection approach and anomaly based detection approach. It is designed to detect attacks and generate signatures so as to update signature repository of Intrusion Detection Systems proactively.
- The combination of signature and anomaly based detection in conjunction with honeypots makes it an effective defense mechanism that detects new attacks within minimum time of their launch and with minimum or zero damage to information.
- Major components of HIDESIGN include honeypot server, Misuse Detection Engine (MDE), Anomaly Detection Engine (ADE) and Signature Generation Engine (SGE). Incoming packet stream from network traffic is captured by honeypot server. Signature based detection is accomplished by MDE to detect and filter out known attacks from this traffic stream. Any deviation from the normal behavior is detected by ADE and a malicious pool is populated. Finally, SGE generates signatures for an alert from malicious pool of data. These signatures may further be used to safeguard the network from intrusions.
- This solution has been implemented using various modules developed in *Java*. It makes use of honeypot technology along with *Snort,* a signature based NIDS and anomaly detection based on supervised machine learning using Naive Bayes classifier.
- ADE is trained using two datasets, namely, HTTP CSIC 2010 and a subset of KDD CUP'99 dataset for training of HTTP and SMTP detection models, respectively. The

evaluation is performed using standard IDS evaluation metrics like Sensitivity, Specificity, False Positive Rate, Accuracy, Detection Rate, F-Measure and ROC Area. The proposed system is capable of successfully detecting and generating attack signatures of various HTTP attacks with a sensitivity and specificity of 99.97% and 98.19% for HTTP attack detection training model. The sensitivity and specificity for SMTP attacks is 96.83% and 93.3% respectively. It has reported 1.8% false positives and 0.02% false negatives for HTTP attacks whereas for SMTP attack detection 6.69% false positives and 3.17% false negatives have been reported.

Chapter Summary

In this chapter, history and evolution of IDS has been discussed in detail. This chapter has elaborated the architectural details of two popular IDSs referred in this research work. The role of anomaly detection using machine learning and other approaches have been presented. Also, available hybrid systems using misuse and anomaly based approach are discussed in this chapter. Thereafter, the automatic signature generation and its techniques are briefly presented. In-depth review of existing signature generation mechanisms has been performed by elaborating their working detail. These systems are then summarized and compared in the tabular form on the basis of various important parameters. Some of these systems include *Honeycomb, Autograph, Earlybird, PAYL, COVERS, ARBOR, Polygraph, Nemean, Argos, Hamsa, Hancock, Nebula, Auto-Sign* and *F-Sign.* The chapter concludes by highlighting research motivation, objectives of proposed research, major achievements and contributions of this work.

Chapter 3

HIDESIGN: A Proposed Framework for Attack Detection and Signature Generation

3.1 Introduction

Intrusion detection is highly desired and challenging task in today's network environment. Detection approaches like signature based and anomaly based approach have their own pros and cons. None of them can be considered as a complete solution to this problem. The proactive approach is always beneficial for safeguarding the network and its resources from unknown vulnerabilities by detecting them well in advance. So, a combination of signature and anomaly based detection along with the proactive feature of honeypot technology is a reliable option. In this thesis work, a hybrid intrusion detection and signature generation mechanism named HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator) is proposed. HIDESIGN is a hybrid approach for intrusion detection and signature generation which relies on three layers of defense. The combination of signature and anomaly based detection in conjunction with honeypots makes it an effective defense mechanism that detects new attacks in a proactive manner to protect the system from any damage targeted by novel attacks. It generates the signatures for unknown attacks and updates IDS repository in, so as to make them available for detection of such attacks in future.

In this work, the major focus is on detection and signature generation of HTTP and SMTP related attacks. The following sections in this chapter describe basics of HTTP and SMTP protocols along with their attack vectors, overview of HIDESIGN, architecture and working of HIDESIGN.

3.2 HTTP Background

Hypertext Transfer Protocol (HTTP) is light and fast application-level protocol necessary for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data

communication for the World Wide Web. HTTP is the protocol to exchange or transfer hypertext. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods. HTTP standards development was coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C) published as series of Requests for Comments (RFCs). RFC 2616 published in June 1999 [Fie09] defines HTTP/1.1, the version of HTTP in common use. HTTP is used as a generic protocol for communication between user agents and proxies/gateways to other Internet protocols, such as SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), Gopher, and WAIS (Wide Area Information Servers) allowing basic hypermedia access to resources available from diverse applications and simplifying the implementation of user agents. It is also one of the few protocols that bridge the gap between networking and application development groups, containing information that is used by both in the delivery and development of web-based applications. The working of HTTP, particularly the headers used by the client and the server to exchange information regarding state and capabilities, often have an impact on the performance of web-based applications. This protocol is based on a request/response paradigm in the client-server computing model. A web browser, for example, may be the client and an application running on a computer hosting a web site may be the server. The client submits an HTTP request message to the server. The server returns a response message to the client. Server provides resources such as HTML files and other content, or performs certain functions on behalf of the client. The response contains completion status information about the request and may also contain requested content in its message body. HTTP is a stateless protocol in a way that the current request does not know what has been done in the previous requests.

A typical client server communication sequence is shown in Figure 3.1. A client establishes a connection with a server and sends a request to the server in the form of a request method, URI (Uniform Resource Identifier), and protocol version, followed by a MIME (Multi-Purpose Internet Mail Extensions) like message containing request modifiers, client information, and possible body content.

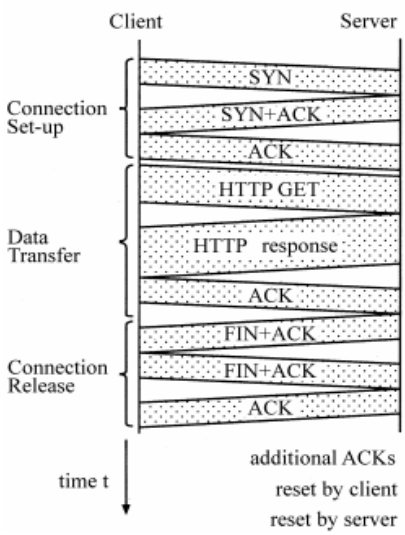


Figure 3.1: Sequence of messages exchanged during client server communication in HTTP

The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta-information, and possible body content. Once the communication is complete the connection is released. An example request response communication process [HTT09] is illustrated in Figure 3.2.

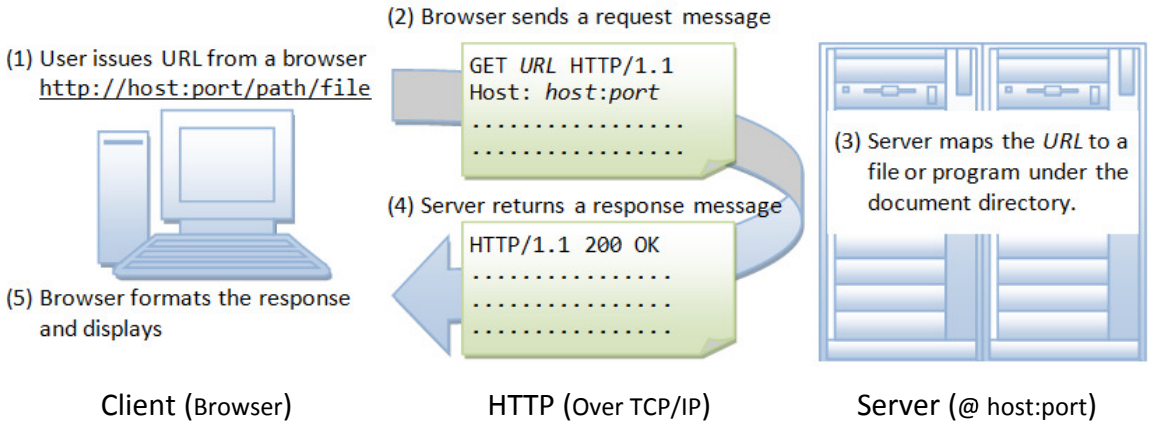


Figure 3.2: Example scenario of a typical request response process [HTT09]

The browser translates URL into a request message according to specified protocol and sends request message to server. For instance, browser translates the URL *http://www.test.com/doc/index.html* into following request message.

```
GET /docs/index.html HTTP/1.1
Host: www.test.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

When this request message reaches the server, it can interpret the request received, map request into a file under server's document directory, and return file requested to client. It may also map the request into a program kept in server, execute the program, and returns the output of program to client. If request cannot be satisfied, server returns an error message. An example of the HTTP response message is as shown below.

```
HTTP/1.1 200 OK
Date: Wed, 04 Sep 2013 09:36:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
ETag: "10000000365a5-2e-3c96b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug
<html><body><h1>It works!</h1></body></html>
```

The browser receives response message, interprets message and displays the contents of message on browser's window according to media type of the response. Common media type include "text/plain", "text/html", "image/gif", "image/jpeg", "audio/mpeg", "video/mpeg", "application/msword", and "application/pdf" [HTT09].

3.3 HTTP attack vectors

With increasing shift towards web applications, many avenues for new attack vectors have open. HTTP is a very commonly used protocol for web communication and its provision in the

network is generally considered as safe and necessary. However, this belief can be exploited by hackers. There is a variety of attacks which can be launched onto HTTP traffic [Gal08]. As per Open Web Application Security Project (OWASP), major attacks are Cross Site Scripting attack, Directory Traversal attack, SQL Injection attack, Command Injection attack, Cross-Site Request Forgery, Insecure Direct Object References and Xpath Injection [Ope13]. Brief description about these attacks has been given below.

i) Cross Site Scripting (XSS)

Cross Site Scripting attacks work by embedding script tags in HTTP requests and inviting victims to click on them. The malicious script then gets executed on victim's machine. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to end user. XSS could potentially impact any site that allows users to enter data. This vulnerability is commonly seen on search engines, error messages, input forms and web message boards. The expressions which may trigger *JavaScript* or other active code may be contained in HTML tags such as *JavaScript, VbScript, expression, applet, meta, xml, blink, link, style, script, iframe, frame, frameset, embed, object, bgsound, title, base, ilayer, layer etc.* XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites.

ii) Directory Traversal attack

Directory Traversal, also known as Path Traversal, is an HTTP exploit which allows attackers to access restricted directories and execute commands outside of web server's root directory. The attacker targets to search absolute links to files stored on web server by browsing the application. It may be possible to access arbitrary files and directories stored on file system, including application source code, configuration and critical system files. Attacker may use "dot-dot-slash *../*" sequences and its variations like "\.." to move up to root directory. This permits navigation through file system. There may be many variations for these sequences. For example, "/" is used for representing root directory in UNIX, whereas in Windows "\" represent root directory. The characters "/" or "\" may also represent directory separator. Even "*../*" can be represented in various ways using different kind of encoding. For example, "*%2e%2e%2f*", "*%2e%2e/*", "*..%2f*", "*..%c0%af*" are all different representations of string "*../*".

iii) SQL Injection attack

SQL injection is an attack that involves injection of malicious SQL queries into user input forms. The malicious code is inserted into strings passed to SQL server for parsing and execution. These malicious queries can directly manipulate the database. The control can be shifted out of the original SQL statement by single quote (') or double dash(--). Another method is by using keyword 'OR' in SQL statement. This makes 'Where' condition to become true always leading to execution of query without authentication. The following example in (3.1) is an HTTP request containg a SQL injection attack to capture passwords against the *PostNuke* which is a Content Management System (CMS).

```
http://[target]/[postnuke_dir]/modules.php?op=modload&name=Messages&file=readpmsg&sta
rt=0%20UNION%20SELECT%20pn_name,null,pn_uname,pn_pass,pn_p                    (3.1)
```

This attack returns database table entries upon successful completion.

iv) Command Injection attack

The purpose of command injection attack is to inject and execute commands specified by attacker in vulnerable application. The vulnerable application acts as a pseudo system shell, which executes unwanted system commands and attacker may use it as any authorized system user. Command injection attacks occur because of lack of correct input data validation, which can be manipulated by attacker using input forms, cookies, HTTP headers *etc.*

v) Cross-Site Request Forgery (CSRF)

A CSRF attack forces a logged-on victim's browser to send a forged HTTP request, including the victim's session cookie and any other automatically included authentication information, to a vulnerable web application. This allows the attacker to force the victim's browser to generate malicious requests which seems legitimate to vulnerable application. The application allows a user to submit a state changing request that does not include anything secret. Consider the following example as given in (3.2).

```
http://example.com/app/transferFunds?amount=1500 & destinationAccount=7346244332
                                                                        (3.2)
```

The attacker constructs a request that will transfer money from the victim's account to the attacker's account, and then embeds this attack in an image request as in (3.3).

```
<img src="http://example.com/app/transferFunds?
amount=1500&destinationAccount=attackersAcct#“ width="0" height="0" />          (3.3)
```

If the victim visits any of the attacker's sites while already authenticated to any legitimate site, these forged requests will automatically include the user's session information, authorizing the attacker's request.

vi) Insecure Direct Object References

A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, or database key. Without an access control check or other protection, attackers can manipulate these references to access unauthorized data. Consider the following example of insecure direct object reference in (3.4), where the application uses unverified data in a SQL call that is accessing account information.

```
String query = "SELECT * FROM accounts WHERE account_no = ?"; PreparedStatement pstmt
=connection.prepareStatement(query,…);pstmt.setString(1,request.getParameter("acct"))
; ResultSet results = pstmt.executeQuery( );                                    (3.4)
```

Here, attacker simply modifies the '*account_no*' parameter in browser to send whatever account number he/she wants. If not properly verified, the attacker can access any user's account, instead of only the intended customer's account as in (3.5).

```
http://example.com/app/accountInfo?account_no=notmyacct                        (3.5)
```

vii) XPath Injection

XPath is used to query XML database. XML database is represented using an XML document. XPath queries are used for search requests, for login processing, for data retrieval, and other lightweight database tasks. XPath injection takes place in web site, which constructs XPath from user input. For example, consider an XML document with three elements, namely, user name,

password and account number. The XPath expression to retrieve account number requires user to give user name and password as shown in (3.6).

```
string(//user[name/text()='xyz' and password/text()='1234']/account/text())        (3.6)
```

If the attacker enters string *' or 1=1 or ' '='* in username field, the Xpath expression would become (3.7).

```
string(//user[name/text()='' or 1=1 or ' '=' ' and
password/text()='xyz']/account/text())                                            (3.7)
```

Since, 1=1 is a tautology, it makes this statement true and allows the attacker to login as first user listed in XML document.

As per a web applications vulnerability report by IBM Managed Security Services group, SQL injection attacks are one of the continued top attacks experienced by client networks [IBM13].

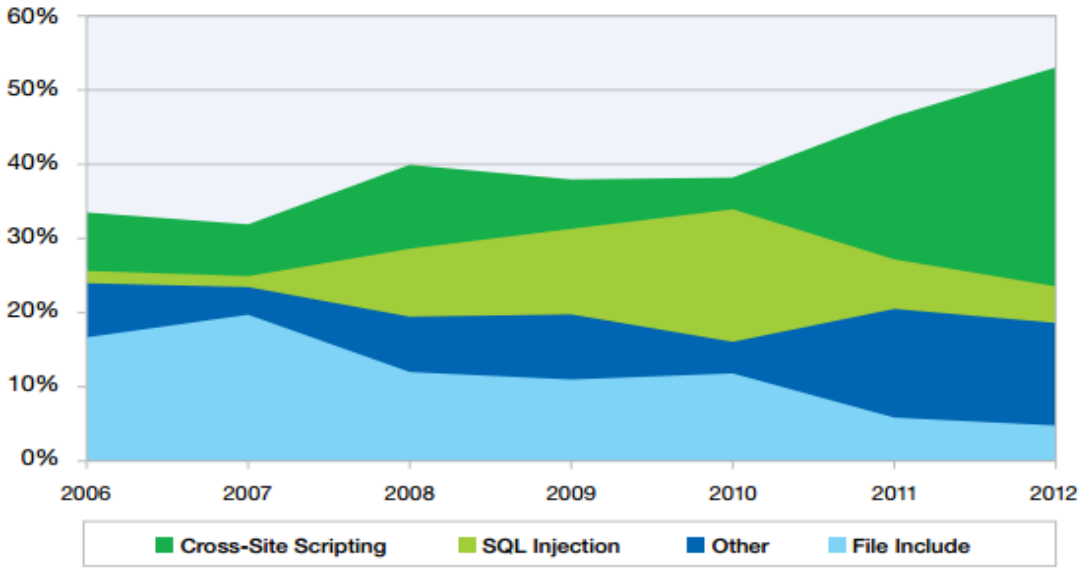


Figure 3.3: Web application vulnerabilities trend from 2006 to 2012 [IBM13]

Figure 3.3 summarizes the web applications vulnerability data from year 2006 to 2012 based upon attack category. Cross-site scripting dominated the web vulnerability disclosures as they

were reported to be 53% percent of all publicly released web application vulnerabilities. The vulnerabilities related to file inclusion occur when the application retrieves code from an invalidated remote source, to be executed in the local application. This allows an attacker to use the web application to remotely execute malicious code. Other category includes denial-of-service attacks, miscellaneous techniques that allow attackers to view or obtain unauthorized information, and to change files, directories, user information or other components of web applications.

3.4 SMTP Background

SMTP (Simple Mail Transfer Protocol) is a standard host-to-host communication protocol operating on TCP port 25. It is a set of communication guidelines that allow software to transmit email over the Internet. SMTP started becoming popular in 1980's. Most email software is designed to use SMTP for communication purposes when sending email. The objective of the Simple Mail Transfer Protocol (SMTP) is to transfer mail reliably and efficiently. SMTP is independent of the particular transmission subsystem and requires only a reliable ordered data stream channel. An important feature of SMTP is its capability to transport mail across networks, usually referred to as "SMTP mail relaying". Using SMTP, a process can transfer mail to another process on the same network or to some other network via a relay or gateway process accessible to both networks. In this way, a mail message may pass through a number of intermediate relay or gateway hosts on its path from sender to ultimate recipient.

The Mail eXchanger (MX) mechanisms of the domain name system are used to identify the appropriate next-hop destination for a message being transported. SMTP transports a mail object. A mail object contains an envelope and content. The SMTP envelope consists of an originator address (to which error reports should be directed); one or more recipient addresses; and optional protocol extension material. The SMTP content has two parts, namely, headers and body. The content is textual in nature, expressed using the US-ASCII. The basic structure of SMTP model is discussed in following subsection.

3.4.1 Basic Structure of SMTP Model

Figure 3.4 depicts SMTP model as described in RFC 5321 [Kle08]. When an SMTP client has a message to transmit, it establishes a two way transmission channel to an SMTP server. The responsibility of an SMTP client is to transfer mail messages to one or more SMTP servers, or report its failure to do so. A client computer that wants to connect to an SMTP server must establish a connection to the server port number 25 and use TCP to its low-level communication.

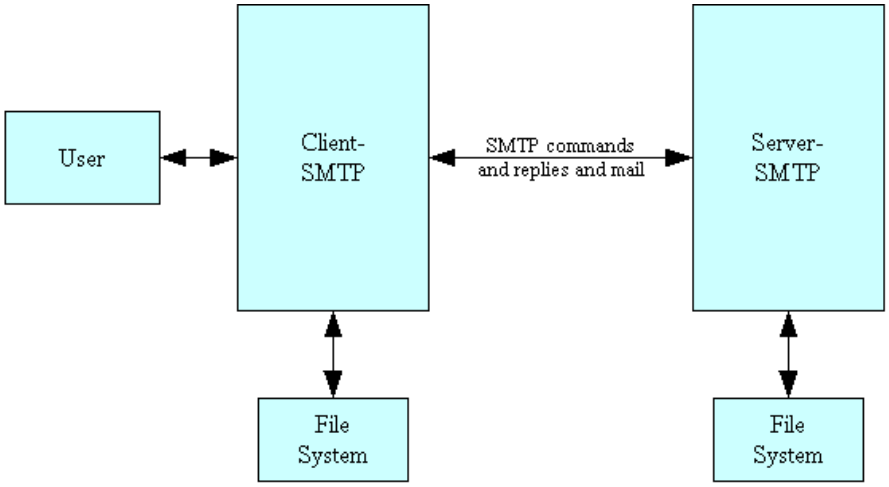


Figure 3.4 SMTP Model [Kle08]

SMTP has a limited ability to queue messages at the receiving end, it is usually used with one of two other protocols, POP3 (Post Office Protocol) or IMAP (Internet Message Access Protocol), which let the user save messages in a server mailbox and download them periodically from the server. SMTP is used for sending e-mail and either POP3 or IMAP for receiving e-mail as seen in Figure 3.5. The major components taking part in SMTP communication are described below.

- *Mail User Agent (MUA)*

MUA, also known as email client, is a program used by user to send and receive emails. Examples of MUAs include *Outlook Express, KMail, Eudora, Mail* and *Thunderbird*. In addition, there are web based email services such as *SquirrelMail*, *AIM Mail, Yahoo! Mail*, *Gmail*, and *Hotmail*. MUA reads the incoming emails that have been delivered to the recipient's mailbox, and passes the outgoing emails to the mail transmission agent for dispatching.

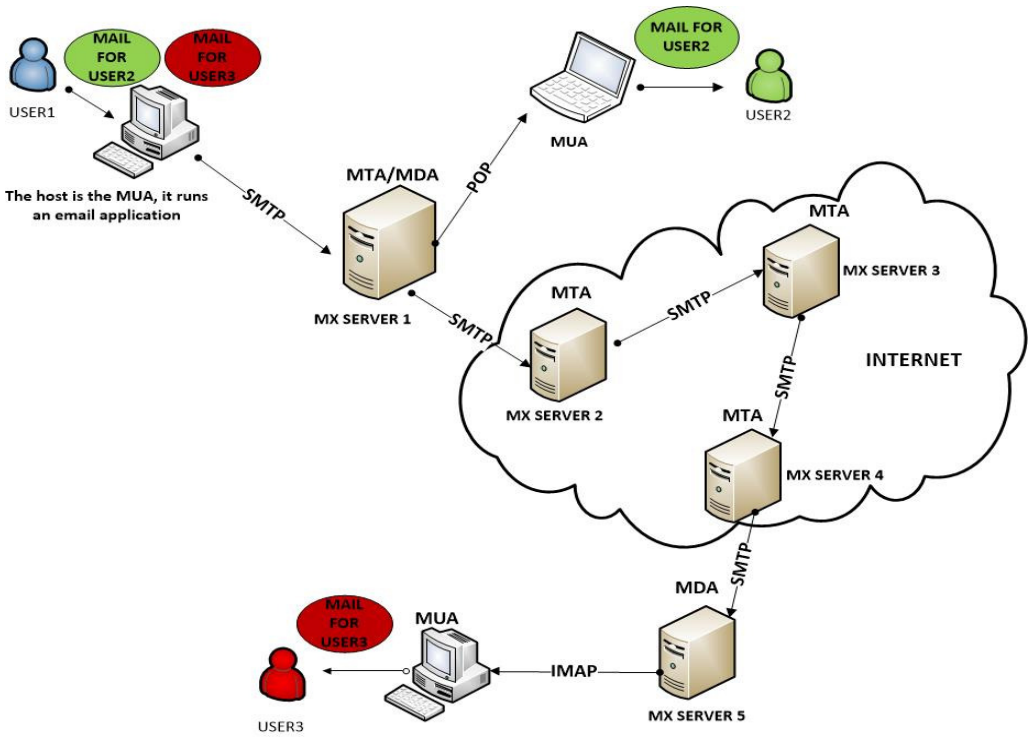


Figure 3.5: Communication between client and server to send and receive mails

- *Mail Transmission Agent (MTA)*

The MTA acts as a "mail router" that accepts emails passed to it either by mail user agent or another MTA, and passes them to the appropriate mail delivery agent for delivery. MTA is the mail server software like *Sendmail*, *Postfix*, *Exim*, and *Qmail*. MTA is the most important of the three agents. It is responsible for making intelligent decisions on email transfer.

- *Mail Delivery Agent (MDA)*

The MDA accepts emails from an MTA and is responsible for delivering incoming email messages to recipients' individual mailboxes. Examples of MDA are *mail.local*, *Maildrop*, and -*/rocmail*.

The email communication flow between Sender and Recipient as shown in Figure 3.5 is described in following steps.

- Sender composes message and MUA transfers the message to local MTA using SMTP.

- Sending MTA performs lookup for the Mail eXchanger records of the recipient domain through the Domain Name System (DNS). The sending MTA determines the recipient domain through the recipient email address.
- Sender/Forwarding MTA uses SMTP to send the message to the recipient's MTA. The recipient MTA delivers the message to the recipient's email box using its MDA.
- Recipient uses his MUA to retrieve the message by using POP3 or IMAP. Alternatively, this email can be retrieved by directly logging into a webmail service such as Hotmail or Yahoo! Mail.

3.5 SMTP Attack Vectors

SMTP is a simple protocol and contains only a few basic commands. There are several security threats that are associated with these commands. By its very nature, email servers have a public face. SMTP traffic is generally allowed through firewall in order to accept email messages from other users on the Internet. This high visibility makes an SMTP server potentially more susceptible to attack compared to other services. Prior to the actual email data being sent, a few messages are exchanged back and forth between the client and server. Both parties use these messages to identify each other and to provide a set of instructions that are mutually understandable. This conversation can be exploited by a malicious user to find out potential vulnerabilities in SMTP server. Before malicious user attempts to attack a system, these weaknesses are discovered by sending unexpected messages and/or inspecting the data returned by the server. In most cases an actual email is never sent. Consequently, it is very difficult to find out if such activity is taking place against server. Such events can be proactively detected and any future attacks can be diffused by inspecting and monitoring all clients connecting to SMTP server who do not send any email. Possible attacks which can compromise SMTP servers include email based virus and worms, mail command injection, buffer overflows, SMTP authentication attacks and Denial of Service attacks [Sti11]. Details about attacks and worms targeting SMTP are described in the following subsections.

i) Email Based Viruses

Another important threat of SMTP is email-based viruses, and worms [Zou04]. These have become one of the major Internet security threats today. An email virus is a malicious program,

which hides in an email attachment, and becomes active when the attachment is opened. A principal goal of email virus attacks such as *Melissa* is to generate a large volume of email traffic over time, so that email servers and clients are eventually overwhelmed with this traffic, which effectively disrupting the usage of the email service. Modern email viruses are more damaging, taking actions such as creating hidden backdoors on the infected machines that can be used to control these machines in a subsequent coordinated attack [Luo06] .

ii) Mail Command Injection

Mail command injection is an attack technique used to exploit mail servers and webmail applications that construct IMAP/SMTP statements from user-supplied input that is not properly sanitized. There are two types of injections based upon the type of statement exploited, namely, IMAP and SMTP injection. An IMAP/SMTP injection may make it possible to access a mail server. In typical structure of an IMAP/SMTP injection header contains ending of the expected command, injection of the new command(s) is done in body and footer contains beginning of the expected command. In order to execute the IMAP/SMTP command, the previous command must have been terminated with the CRLF (%0d%0a) sequence [OWA14]. An IMAP/SMTP injection makes it possible to access a mail server which otherwise would not be directly accessible from the Internet. Because of lack of infrastructure security and hardening applied to the front-end web servers, mail server become more vulnerable to attacks by end users as presented in Figure 3.6.

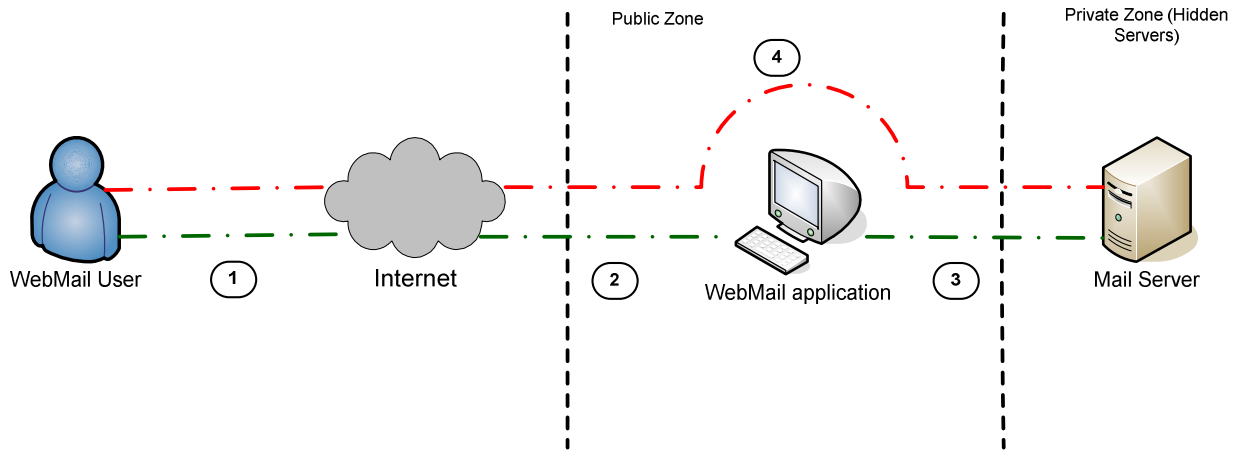


Figure 3.6: Communication with the mail servers using the IMAP/SMTP injection technique [OWA14]

It depicts the flow of traffic generally seen when using webmail technologies. Here, steps 1, 2 and 3 depicts normal path of user interacting with the webmail client, whereas step 4 is the attacker bypassing the webmail client and interacting with the back-end mail servers directly. This technique allows a wide variety of actions and attacks possibilities depending on the type and scope of injection. After vulnerable parameters are identified and analyzed as per the context in which they are executed, the functionality may be exploited.

iii) Buffer Overflows

Buffer overflows send a larger quantity of data to the server than is anticipated. Depending on how the overflow is executed, it could cause the server to stop working or it might run malicious code from the attacker. Various variants of buffer overflow have been explained below.

- **SMTP HELO/ EHLO command buffer overflow**

This problem occurs when malicious SMTP HELO/EHLO messages are sent to the affected server. During the HELO/EHLO introduction process the affected server copies strings provided by external SMTP servers or clients into process buffers without sufficient bounds checking, facilitating the buffer overflow vulnerability. A remote attacker may exploit this vulnerability to execute arbitrary code on a computer running the affected software. This will allow unauthorized access and privilege escalation. *Microsoft Exchange Server 2003* could allow a remote attacker to execute arbitrary code on the system.

- **SMTP RCPT TO Buffer Overflow**

This type of buffer overflow vulnerability comes into picture while handling RCPT TO commands. It happens due to a failure of the application to properly check bounds of user supplied data prior to copying it to fixed size memory buffers. These vulnerabilities allow remote attackers to execute arbitrary machine code with system level privileges in the context of the affected application.

iv) SMTP Auth Attacks

To prevent unauthorized users from relaying mail via the SMTP server, it is required to use proper authentication. SMTP authentication is only as strong as the password used. A weak password on a known account can make the SMTP server to be exposed as an open relay with a poor sense of security.

Generally, HTTP and SMTP attacks discussed in the previous sections can be detected by two methods, namely, pattern detection and anomaly detection. Pattern detection [Dha06] techniques seek to find patterns in requests, and then determine if those patterns are associated with legitimate requests. Often these systems have predefined lists of signatures which indicate a common attack. This technique is widely deployed in the form of many Intrusion Detection Systems (IDS) such as *Snort*. In anomaly detection, a base line for normal traffic is generated and then used to identify possible attacks. These anomalies may be in the form of unusual traffic flows or an unexpected behavior. This is hard to do on real networks, as traffic flows can be highly variable, but still not malicious always. However, this approach may prove to be promising for analysis of HTTP and SMTP traffic as anomalies would present themselves as unusual traffic flows.

To encounter attacks in HTTP and SMTP traffic routed to web and mail servers, a network security model named HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator) has been proposed in this thesis. HIDESIGN is a combination of various techniques to proactively handle these threats. The following sections discuss overview, detailed architecture and working of HIDESIGN.

3.6 Overview of HIDESIGN

The proposed system HIDESIGN first detects the attack and then generates the signatures for attack so that these can be stored in IDS repository. This is a hybrid architecture which relies on three layers of defense. The combination of signature and anomaly based detection in conjunction with honeypots makes it an effective defense mechanism that detects new attacks within minimum time of their launch and with minimum or zero damage to information. At the first level, misuse detection is used which detects the known attacks from incoming network

traffic. At the second level, any deviation from the normal behavior can be easily detected by anomaly detector by making use of profile. Finally, the system generates signatures for an alert when it detects any intrusion. Incoming packet stream from network traffic is captured by honeypot server. This packet stream is analyzed by signature based IDS placed in Misuse Detection Engine (MDE). After filtering known attacks, rest of packet stream is analyzed by Anomaly Detection Engine (ADE). Once the malicious candidate is detected by Detection Engine (DE), it is sent to Signature Generation Engine (SGE) that is responsible for signature generation. As shown in Figure 3.7, HIDESIGN aims to detect attack using misuse and anomaly based detection thereby generating the signatures for attack so that these can be stored in IDS repository.

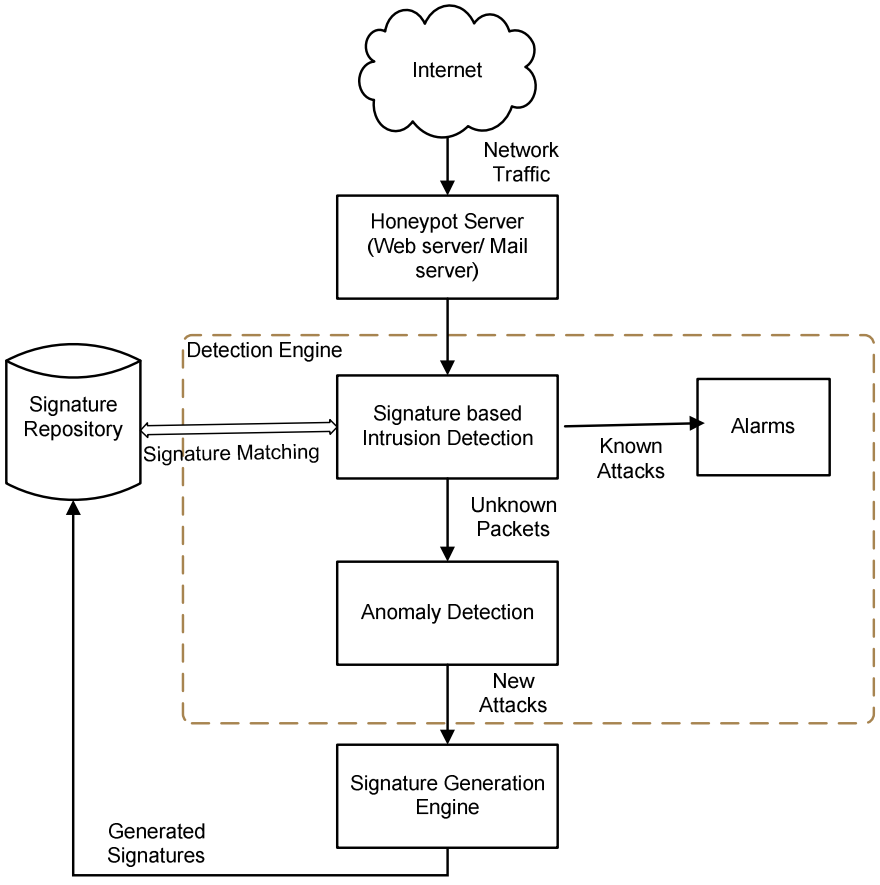


Figure 3.7: Overview of HIDESIGN

The overall objective of HIDESIGN is to generate a set of signature *S = {Sig1, Sig2, Sig3…}* for malicious traffic packets from whole incoming traffic containing normal traffic pool *N* =

{N1,N2,…} and suspicious traffic pool *S = {S1,S2,…}* with high sensitivity and specificity minimizing false positives.

3.7 Architecture of HIDESIGN

The architecture of proposed HIDESIGN system has been given in Figure 3.8.

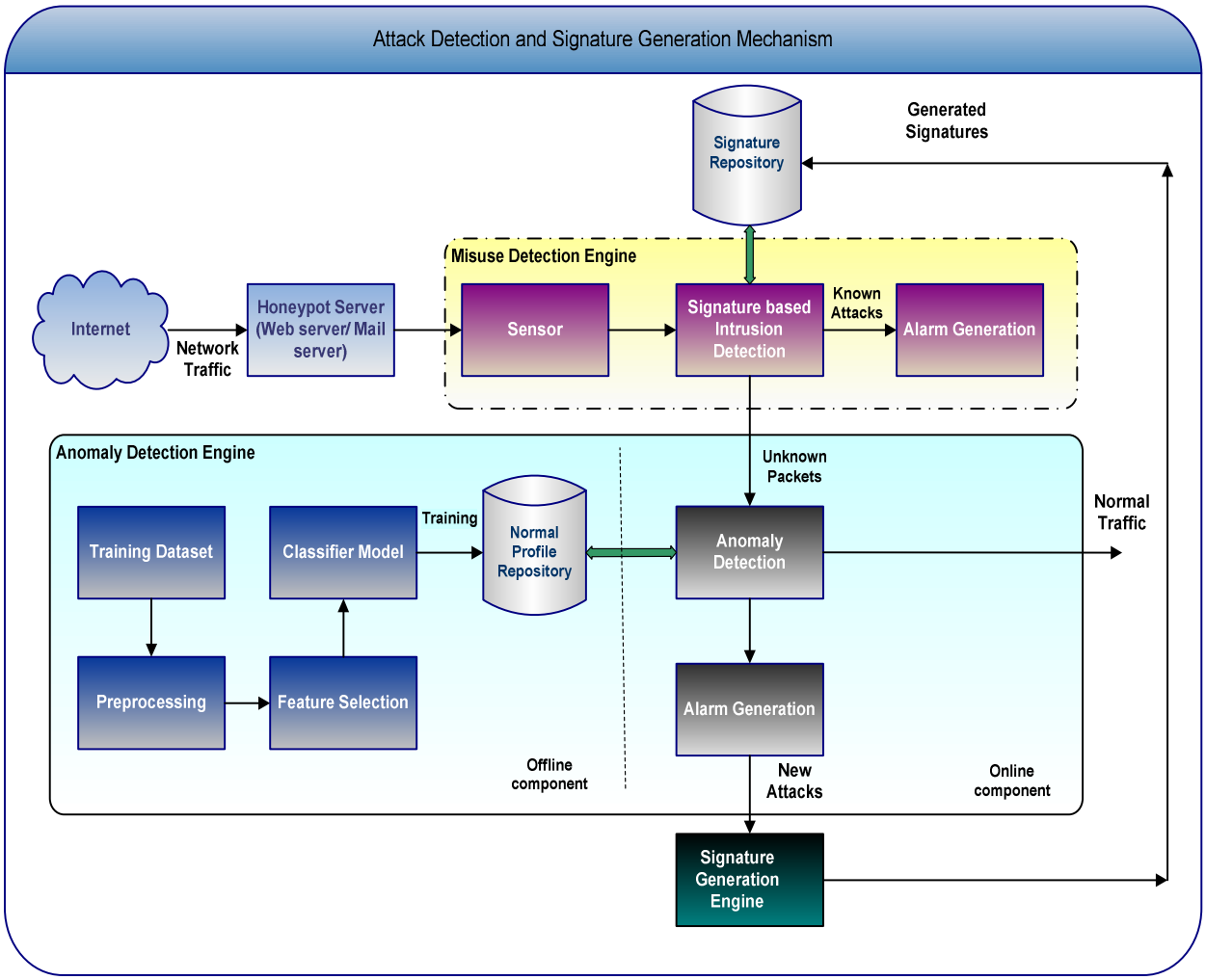


Figure 3.8: Detailed architecture of HIDESIGN

The main components of this architecture have been discussed below.

- Honeypot Server: Honeypot acts as bait and trap sensor to attract malicious traffic. It emulates vulnerable web and mail server so that abnormal HTTP and SMTP traffic can

be routed to these fake servers instead of production servers in the network. This abnormal traffic is captured in honeypot's log and analyzed further to extract information about attacks.

- Detection Engine (DE): This module uses a hybrid approach for detection of attacks or abnormality in the honeypots' logged packets. It comprises of two sub modules, namely, Misuse Detection Engine (MDE) and Anomaly Detection Engine (ADE). Traffic packets are first passed through misuse detection engine and then through anomaly detection engine.

    - MDE is the first filter to detect known attacks. All incoming packets captured by honeypot are analysed by MDE. It uses signature based detection technique to filter out known attacks. The remaining packets which are declared non-malicious can still contain attacks whose signatures are not stored in the signature repository of MDE. So these are forwarded to ADE.

    - ADE is responsible to find any further abnormalities in the packets directed to it via MDE. The normal profile repository is created and updated in an offline mode. Here, machine learning is used to train the system by using training dataset. The online component uses normal profile repository for comparison of packets and to observe any deviations. If some deviation is found an alarm is generated and packet is added to malicious pool. These malicious pool entries are considered as candidates for signature generation.

- Signature Generation Engine (SGE): Once an attack is detected by DE in the form of suspected candidate, it is forwarded to SGE. SGE then extracts patterns from malicious pool, generate signatures and store them in signature repository of IDS.

The details of these components have been discussed in the following sections.

3.8 Honeypot Server

These are decoy servers set up for the purpose of attracting the attacker, monitoring and logging the activities of entities that probe, attack or compromise them. In general, any network activities observed at honeypots are considered suspicious, and it is possible to capture the latest intrusions based on analysis of these activities. Honeypot server is the first layer to separate suspicious traffic pool $S = \{S1,S2,....\}$ from normal traffic pool $N = \{N1,N2,.....\}$. Honeypot is a proven technology to capture malicious traffic, specifically new attacks and exploits. It will narrow down the domain of traffic in which attack is to be detected. Otherwise it will be an overhead to consider the whole traffic stream including normal traffic for the process of attack detection. Honeypot Server in HIDESIGN emulates vulnerable web and mail server so that abnormal HTTP and SMTP traffic can be captured in honeypot's log and analyzed further to extract information about attacks.

3.9 Detection Engine

Attack detection is the process of observing a system in order to separate normal activities from those representing an attack. The process defines normal activities either explicitly or implicitly. Detection Engine is a very important component of HIDESIGN. It analyzes the traffic captured by honeypot and detects attacks. As reported in the literature, detection approaches include misuse and anomaly detection. DE follows hybrid approach in which both misuse and anomaly detection is used. Details of its two components, *i.e.*, MDE and ADE are given in following subsections.

3.9.1 Misuse Detection Engine (MDE)

Misuse detection has been a preferred choice of researchers and security professionals since long because of its various advantages. It is capable of detecting known worms/viruses/attack instances with very low false positives. In HIDESIGN also, it does the same by employing signature based detection. Architecture of MDE is shown in Figure 3.9. It is deployed after honeypot so as to filter out known attacks from suspicious traffic pool $S = \{S1,S2,....\}$. Honeypot traffic is captured by the sensors and fed to the signature based detector after being preprocessed. The misuse detector raises an alarm to the alarm generation module if any connection matches an

intrusion pattern. If the connection does not match any intrusion pattern, it is sent to the anomaly detection engine that stores data for the anomaly detection component.

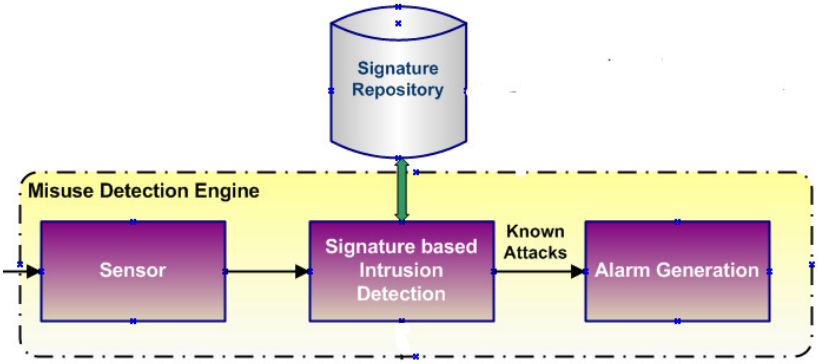


Figure 3.9: Misuse Detection Engine

For the purpose of signature detection, *Snort* has been used. MDE sensor sniffs the honeypot's traffic and generates alerts for the packets whose signatures match with those stored in *Snort's* signature repository.

3.9.2 Anomaly Detection Engine (ADE)

Anomaly detection can be accomplished by following various approaches including machine learning, statistical methods and knowledge based techniques. In literature, numbers of anomaly detection systems are developed based on many different machine learning techniques. Machine learning based network intrusion detection is a valuable technology to protect target systems and networks against malicious activities. Despite variety of such methods described in the literature in recent years, security tools incorporating anomaly detection functionalities are just starting to appear. Machine learning techniques are based on establishing an explicit or implicit model that enables the patterns analyzed to be categorized. The major machine learning based schemes include Bayesian networks, Markov models, neural networks, expert systems, fuzzy logic, genetic algorithms, clustering and outlier detection.

Machine learning is very effective approach to train Intrusion Detection Systems and for building rule sets. It is heavily based on statistical analysis of data and pattern recognition in previous data to make decisions about new data. A model for both normal traffic and suspicious traffic is built to make predictions about current and future traffic. This approach helps in

detecting new types of attacks based on the model of previous suspicious network activity. The data used to build this model is typically called the training data. For effective learning, the choice of a proper training set is very important. In this work, Bayesian classifier algorithm has been used to model normal and suspicious network activity in HTTP traffic. The classifier model has been built using Naive Bayes approach to identify the unknown packet as malicious or normal.

ADE comprises of two major components, namely, offline component and online component as depicted in Figure 3.10. The details of these components are given in following subsections.

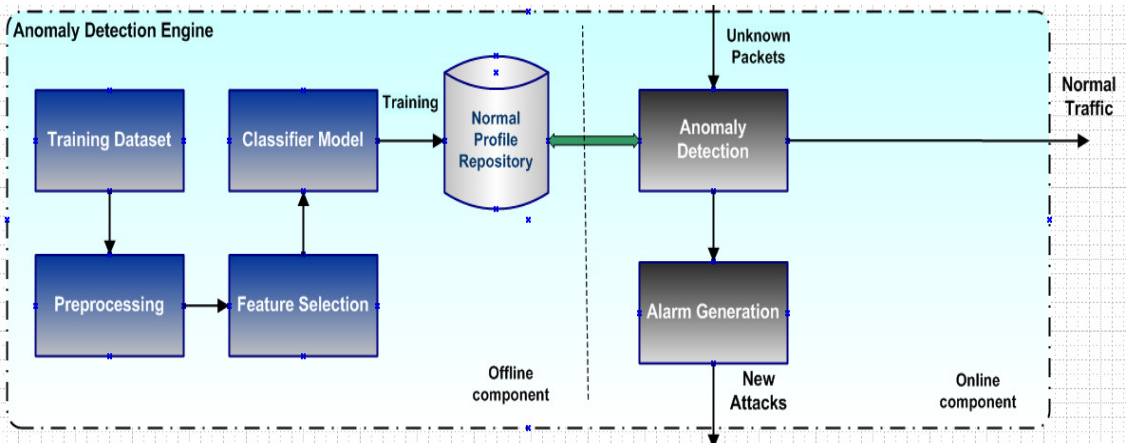


Figure 3.10: Anomaly Detection Engine

3.9.2.1 Offline Component

Offline component is an initial training phase that builds the profile of normal user behavior, by mining rules from network traffic known to be attack free. During training rule classifier learn attack classes through network traffic that has been labeled with known attacks. Offline component uses training dataset to build and train classifier model.

- **Training Dataset**

  A training set is a set of data used in various areas of information science to discover potentially predictive relationships. The data set used for learning consists of a list of m training examples $\{(x(i), y(i)); i = 1, \ldots, m\}$ is called a training set.

Training sets are used in artificial intelligence, machine learning, genetic programming, intelligent systems, and statistics. In all these fields, a training set has much the same role and is often used in conjunction with a test set. The training data consist of pairs of input objects and desired outputs. The output of the function can predict a class label of the input object. The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples, *i.e.,* pairs of input and target output. To achieve this, the learner has to generalize from the presented data to unseen situations in a reasonable way. After a model is learned or built from the training data by a learning algorithm, it is evaluated using a set of test data or unseen data to assess the model accuracy. The training dataset used in ADE consists of various packets that are already classified as normal and anomalous.

- **Preprocessing**

  Preprocessing of data is very significant phase in data mining that involves transforming raw data into an understandable format. Data-gathering methods are often loosely controlled, resulting in out-of-range values, impossible data combinations, missing values, *etc.* Preprocessing is a proven method of resolving various issues of real world data such as incompleteness, inconsistency, lacking in certain behaviors or trends, and likelihood of containing errors. Analyzing data that has not been carefully screened for such problems can produce misleading results. Thus, the representation and quality of data is first and foremost before running an analysis. Data preparation and filtering steps can take considerable amount of processing time.

  Data pre-processing includes cleaning, normalization and transformation *etc.* The outcome of data pre-processing is the final training set. Preprocessing module extracts information from packet headers and builds records which are subsequently mined and classified after storing the preprocessed data in the training database.

- **Feature selection**

  Feature selection is the process of selecting a subset of the terms occurring in the training set and using only this subset as features in text classification. It is an important step to

select only relevant features. Theoretically, having more features should result in more discriminating power. However, practical experience with machine learning algorithms has shown that this is not always the case. Sometimes learning algorithms can give a biased estimate of the probability of the class label given a set of features. Feature selection is the process of identifying and removing as much redundant and irrelevant information as possible. Redundant features are those which provide no more information than the currently selected features, and irrelevant features provide no useful information in any context. Feature selection techniques are a subset of the more general field of feature extraction. This reduces the dimensionality of the data and may allow learning algorithms to operate faster and more effectively. Feature selection serves two main purposes. First, it makes training and applying a classifier more efficient by decreasing the size of the effective vocabulary. This is of particular importance for classifiers that are expensive to train. Second, feature selection often increases classification accuracy by eliminating noise features. A noise feature is one that, when added to the document representation, increases the classification error on new data. Various feature selection methods are reported in literature including ranking methods, search based feature selection and embedded feature selection algorithms used as filters. In the offline component, Ranker's algorithm for feature selection has been used.

- **Classifier Model for Attack Classification**

  Building a classifier model generally means learning from examples. A classifier is a systematic approach to build classification models from an input dataset. It is possible to learn a classifier that predicts class labels of future for previously unknown records. Various classifier examples include decision tree classifiers, rule-based classifiers, neural networks, support vector machines, and probabilistic classifiers such as Naive Bayes classifier. Each of these employs a learning algorithm to identify a model that best fits the relationship between attribute set and class label of the input data. The model generated by a learning algorithm should both fit the input data well and correctly predict the class labels of records it has never seen before. Probabilistic classifier is a classifier that is able to predict a probability distribution over a set of classes for a given sample input.

ADE relies upon a probabilistic model based on Bayes Theorem. The Bayesian classification represents a supervised learning method as well as a statistical method for classification. This classification assumes an underlying probabilistic model and captures uncertainty about the model in a principled way by determining probabilities of the outcomes. Bayesian classifiers have an ability to detect new attacks without having a prior knowledge. It does so by estimating the prior and posterior probabilities of new attacks while keeping the false alarms rate low. In the proposed work, Naive Bayes classifier has been used. Naive Bayes is one of the most popular machine learning algorithms based on a very strong independence assumption that classifies high dimensional input data. The method strongly assumes independence of attributes to each other and henceforth is named "Naive". It analyzes the relationship between independent variable and the dependent variable to derive a conditional probability for each relationship. Once the system is trained, it is possible to classify any new object based on its attributes using the Bayes rule as given in (3.8).

$$P(Ci \mid X) = [P(X \mid Ci) \cdot P(Ci)] / P(X) \quad (3.8)$$

This probability is known as posterior probability. Here,

- P(Ci | X) is the probability of the object X belonging to class Ci;
- P(X | Ci) is the probability of obtaining attribute values X if we know that it belongs to class Ci;
- P(Ci) is the probability of any object belonging to class Ci without any other information;
- P(X) is the probability of obtaining attribute values X whatever class the object belongs to.

Let X is an object to be classified and Ci's are the possible classes in which object X can be classified, the classifier calculates P(Ci | X) for all the classes and will assign X to the class that has the highest conditional probability.

- **Normal Profile Repository**

  ADE is responsible for building a profile of the normal behavior. Here, normal profile can be defined as patterns or summary statistics for the non-malicious population of network traffic. This normal profile will further be used to detect anomalies. Anomalies are observations whose characteristics differ significantly from the normal profile. The classifier model is used to build normal profile repository which is referred by online component for detection of any anomalous behavior of incoming packet stream.

3.9.2.2 Online Component

Online component performs dynamic analysis of the packets which are unidentified by MDE. It matches the behavior of these packets with normal profile behavior, detects any deviation and generates alarms. Those packets for which alarms have been raised are considered to be malicious packets and are candidates for signature generation. There are two phases of online component, namely, Anomaly Detection phase and Alarm generation phase as can be seen in Figure 3.11.

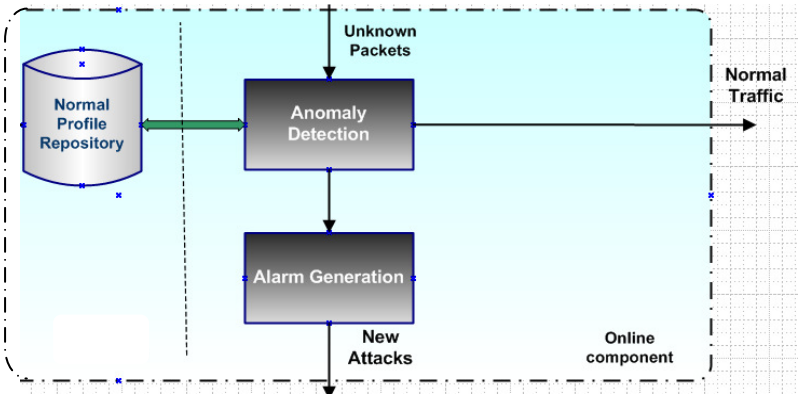


Figure 3.11: Online component of ADE

The details of these two phases have been given below.

- **Anomaly Detection**

  In the anomaly detection phase, the system can detect novel intrusions using the anomaly detection component. This phase is same as testing phase in supervised machine learning in which first the machine is trained using labeled data and then testing is done on

unknown and unlabeled data. Here Naive Bayes classifier, which was built in offline component for the training of system, is used for classification of unknown packets. Given an instance of packet object *A* and possible classes *Ci* to which *A* can be assigned, the classifier calculates *P(Ci | A)* for all the classes and will assign *A* to the class that has the highest conditional probability.

The patterns are retrieved from the normal profile database and classifier model is used to detect any deviation from this normal profile. This deviation is considered to be anomalous and the corresponding packet is appended to malicious pool *M = {M1, M2, M3,…}*. If any attack is detected, it raises alarms to the anomaly alarm generation module. These newly detected intrusions are forwarded to SGE so that the new intrusion patterns can be built for misuse detection.

- **Alarm Generation**

  Alarm generation phase in anomaly detection engine is responsible for generating alarms for anomalous connections. These alarms are significant because these are for possible new attacks which were not reported by signature based detection module of MDE. These alarms can be used for cross verification of automatically generated signatures.

3.10 Signature Generation Engine

Attacks detected by Detection Engine are further processed by Signature Generation Engine for *Snort* compatible signature generation. Signature Generation Engine is the component responsible for generating content based attack signatures automatically from malicious pool of packets. Here, malicious pool consists of multiple strings *M1, M2……,Mn,* the problem is to find common substrings of minimum threshold length and which are present in at least a minimal percentage of overall strings. In particular, a byte sequence *str* is considered to be a candidate signature string if its size is at least *l* bytes long and it occurs in at least λ percent of malicious flows. For the signature extraction a generalized suffix tree based algorithm has been used. It finds Longest Common Substring (LCS) using suffix tree and considers that as content for IDS signature generation. Let string *X* of length *m* represented as *X(1…m),* and string *Y* of length *n*

represented as *Y(1..n),* longest common substring finds the longest string (or strings) that is a substring (or are substrings) of two or more strings.

Suffix trees [Ukk95] are used to store the large strings and extract the substrings. These may be used in compression, searching, pattern matching, and many other problems. A suffix tree *T* for an *m*-character string *S* is a rooted directed tree with exactly *m* leaves numbered *1* to *m*. Each internal node, other than the root has at least two children and each edge is labeled with a nonempty substring of *S*. No two edges out of a node can have edge labels beginning with the same character. The key feature of the suffix tree is that for any leaf *i*, the concatenation of the edge-labels on the path from the root to leaf *i* exactly spell out the suffix of *S* that starts at position *i*. It is convention to place a termination character at the end of the string. This character should not appear anywhere else in the string and is usually taken to be *$*. This ensures that no suffix will be a prefix of another suffix. Figure 3.12 shows the suffix tree for the string *S* = *abcab$*.

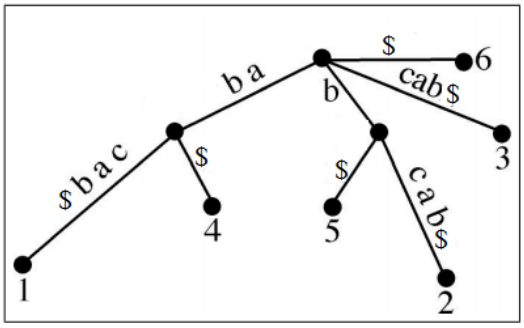


Figure 3.12: Suffix tree for string *S* = *abcab$*

A generalized suffix tree is a tree that encodes the suffixes of multiple strings. A generalized suffix tree can be used to represent all suffixes of a set of strings. There are two major significances of holding the suffixes of multiple strings in the same suffix tree. First, to determine which suffix comes from which string and secondly, there may be a suffix that is common to more than one string. In generalized suffix trees, leaves may be labeled with more than one pair of numbers, one to represent the string in which the suffix is contained and the other to represent the start position in that string. Figure 3.13 shows a generalized suffix tree of two strings *S1=abc$* and *S2= cdc$*.

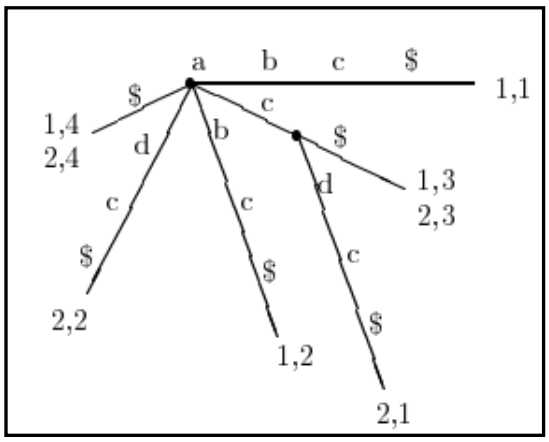


Figure 3.13: Generalized suffix tree of two strings *S1=abc$* and *S2= cdc$*

There are many algorithms which can build suffix tree in linear time. Signature Generation Engine has been implementation using Ukkonen's [Ukk95] algorithm a linear time algorithm for construction of suffix tree. Given two strings, *S1* of length *p* and *S2* of length *q*, longest common substrings of *S1* and *S2* can be found in *O(p+q)* time with the help of a generalized suffix tree. SGE makes use of regular expressions and pattern matching to generate message part in *Snort* signatures. SGE also takes care of quality and redundancy of signature. If new signature is same as existing signature in repository, no new signature is generated. If it is a superset of already existent signature, or it is not present already then new signature is appended to repository.

3.11 Workflow of HIDESIGN

The overall workflow of HIDESIGN has been depicted with the help of an activity diagram as shown in Figure 3.14. The vertical partitions called swim lanes in activity diagram are showing the major components of HIDESIGN. Every incoming packet destined to honeypot from network is received and logged by honeypot's emulated server. These logs are inspected by Misuse Detection Engine. MDE detects and filters out the packets which contain malicious content. This is accomplished by matching signature of packets with the ones stored in signature repository of IDS.

If a match is found the alarm would be generated and packet is logged or dropped. The rest of packets are sent to Anomaly Detection Engine which comprises of two sub components, namely, offline and online component. The offline component is responsible for building and maintain normal profile repository of ADE. It can be done in offline mode or in parallel to the previous steps performed till misuse detection. The packets which were declared non-malicious by MDE are actually uncertain packets and will further be analyzed for any abnormality. The online component of ADE will perform the anomaly detection on these packets by comparing them with the normal behavior profile. The outliers will be identified and alarms would be generated. These packets are added to malicious pool. This is a pool that contains the packets which are candidates for signature generation. SGE will take this malicious pool as input. A suffix tree based Longest Common Substring algorithm has been used to find the contents which are occurring in at least λ percent of packets and length l should be at least greater than or equal to a threshold value *'TH'*. This content is used to generate the signature in a format that is understood by IDS and is stored in signature repository.

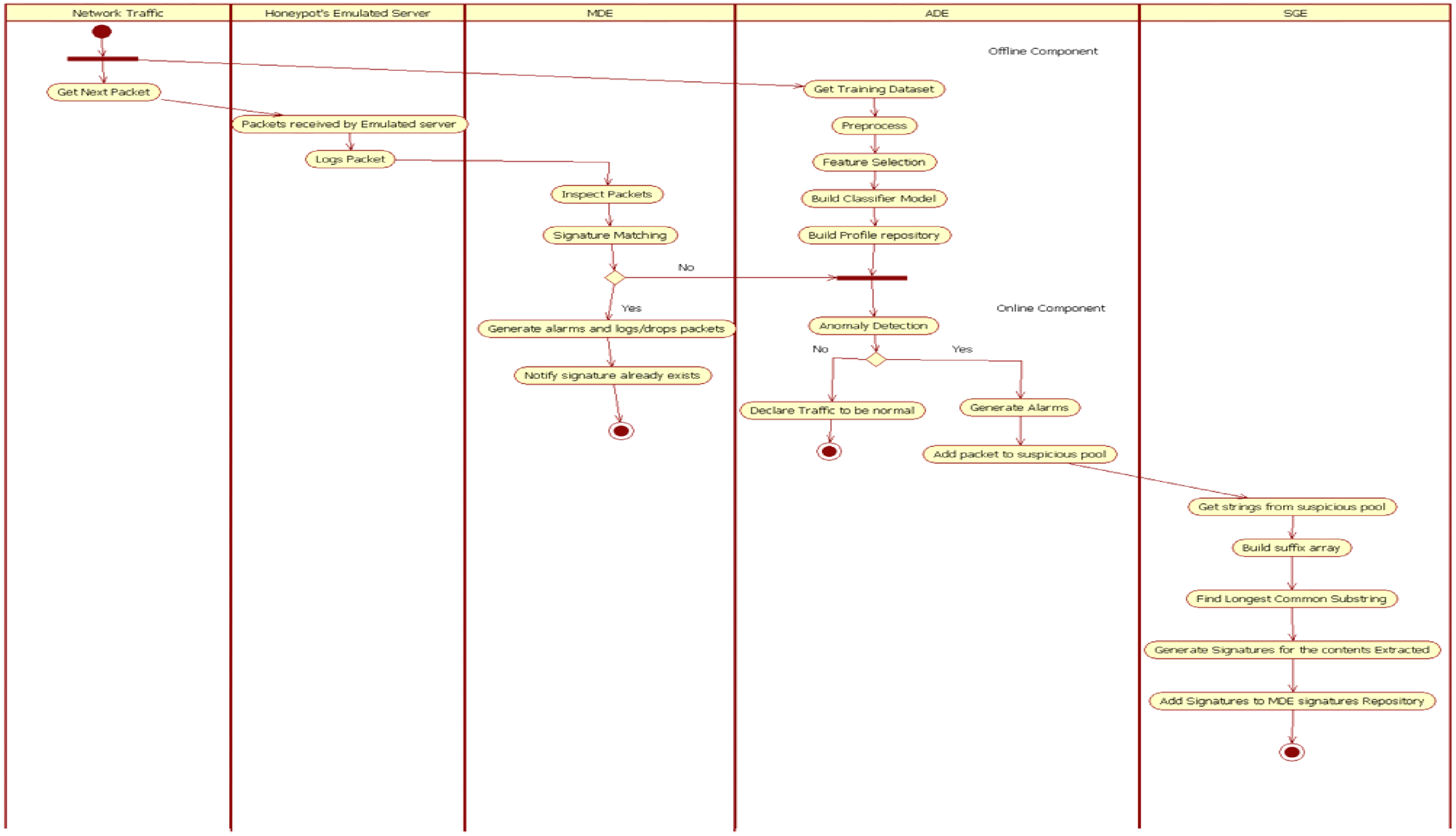


Figure 3.14: Activity diagram describing workflow of HIDESIGN

Chapter Summary

In this chapter the details of HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator) for automatic generation and tracking of attack signatures in context with HTTP and SMTP traffic have been discussed. It is designed to generate signatures so as to update signature repository of Intrusion Detection Systems proactively. In this chapter, an introduction followed by background and attack vector details of HTTP and SMTP protocols has been presented. Various types of HTTP attacks such as SQL injection, command injection, directory traversal, cross site scripting attacks *etc.* have been discussed in detail. SMTP attacks and their details are also discussed in this chapter. Followed by detailed discussion of HTTP and SMTP attacks, the proposed HIDESIGN framework for attack signature generation and tracking is presented. This chapter elaborated the detailed architectural design and work flow of proposed system and its components.

Major components of HIDESIGN include honeypot server, Misuse Detection Engine (MDE), Anomaly Detection Engine (ADE) and Signature Generation Engine (SGE). Incoming packet stream from network traffic is captured by honeypot server. Signature based detection is accomplished by MDE to detect and filter out known attacks from this traffic stream. Any deviation from the normal behavior is detected by ADE and a malicious pool is populated. Finally, SGE generates signatures for an alert from malicious pool of data. These signatures are used to update signatures repository of NIDS and may further be used to safeguard the network from intrusions. The combination of signature and anomaly based detection in conjunction with honeypots makes it an effective defense mechanism that detects new attacks within minimum time of their launch and with minimum damage to information.

Chapter 4

Implementation details of HIDESIGN

4.1 Introduction

This chapter contains the details about deployment of HIDESIGN in actual network environment. Details of experimental set up along with configuration of honeypot, implementation details of misuse detection engine and anomaly detection engine are provided in this chapter. This chapter also explains how signature generation engine has been implemented and its outcome. It elaborates various experiments performed. In the last section, all modules are jointly presented to explain the overall working details of HIDESIGN. The chapter concludes with a summary of all the sections.

4.2 Deployment of HIDESIGN in Actual Network

The experimental test environment deployment is depicted with the help of diagram as shown in Figure 4.1. Production web server is placed in De-Militarized Zone (DMZ). Layer3 firewall and NIDS (Network Intrusion Detection System), placed between web server and outer network, protects the internal network from known intrusions. In parallel, emulated web/mail server on honeypot is deployed in internal network. Emulated honeypot server receives packets and logs them for further processing. These logs are read and analyzed by Detection Engine (DE). DE is responsible for identifying suspected attack candidates from these logs automatically. First, Misuse Detection Engine (MDE) filters packets by detecting known attack patterns. Then remaining traffic is analyzed by Anomaly Detection Engine (ADE). It will detect unknown attack patterns which are not there in NIDS repository. Once an attack is detected by DE, it is forwarded to Signature Generation Engine (SGE). SGE then generates signature patterns and stores them in signature repository looked up by MDE. These signatures are also used to update signature repository of NIDS placed in the network and will prevent attacks to be launched on production web server. It detects attacks at an early stage and corresponding signatures will prevent any further damage to actual web/mail server.

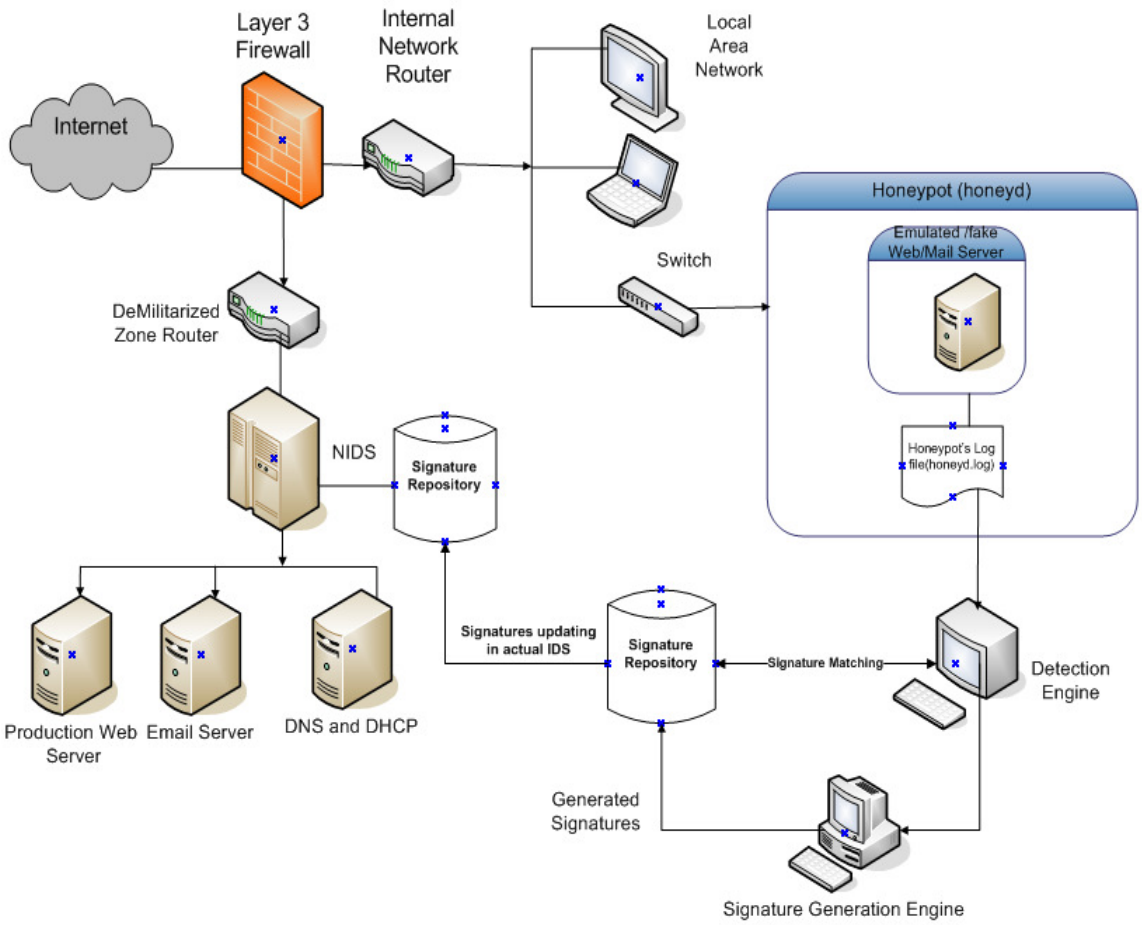


Figure 4.1: Deployment Details of HIDESIGN

4.3 Experimental Setup Details

For experimentation purpose, *Snort* has been used as NIDS in a local environment. Honeypot is deployed in *Backtrack* Linux machine. It is configured to emulate *Microsoft IIS* web server. It also fakes *Microsoft Exchange* server that is an email server. Figure 4.2 depicts the experimental details of one of the experiments to generate HTTP attacks signatures. For experimental purpose original web logs of an educational institute's *Microsoft IIS 7.0* web server have been used. The *GET* requests from these logs are extracted and stored in a file by log digging engine implemented in *Java*. These *GET* requests are then replayed on emulated web server of honeypot using *wget* utility. Misuse Detection Engine detects known attacks from these requests. Anomaly Detection Engine uses machine learning to analyze the remaining packets and detect anomalies, if any. Signature Generation Engine generates *Snort* compatible signatures to be stored in *Snort's* signature repository named *Snort.rules*.

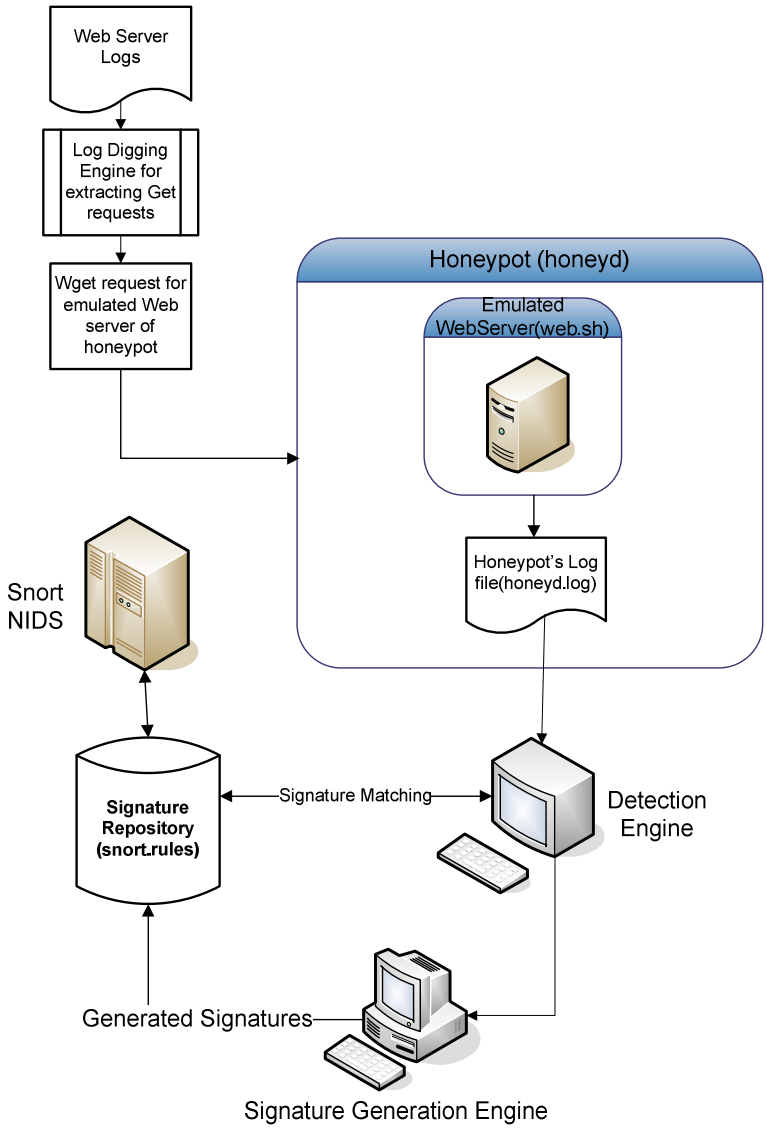


Figure 4.2: Experimental setup using a low interaction honeypot

4.4 Honeypot Configuration

A low interaction honeypot named *honeyd* has been used for configuration of web and email server. The script named *webmin.sh* is customized for emulation of *Microsoft IIS* web server hosted on *Microsoft Windows XP Professional Service Pack 1*. The following statements are representing a part of configuration file of *honeyd* named *honeyd.conf*. Here, *webmin.sh* is the script file for emulation of web server.

```
create windows
set windows personality "Microsoft Windows XP Professional SP1"
set windows uptime 1728650
set windows maxfds 35
add windows tcp port 80 "sh /scripts/webmin.sh $ipsrc $sport $ipdst $dport"
add windows tcp port 137 open
add windows tcp port 139 open
set windows default tcp action reset
set windows default udp action reset
set windows default icmp action open
set windows ethernet "00:0c:29:e8:30:333c"
bind 172.31.6.41 windows
create default
set default default tcp action block
set default default udp action block
set default default icmp action block
```

For SMTP server configuration, instead of using *webmin.sh* as script, following line is added in the configuration file. Here, *exchange-smtp.sh* has been configured so as to emulate fake *Microsoft Exchange* server.

```
add windows tcp port 25 "sh /scripts/exchange-smtp.sh"
```

4.5 Data Capturing

Honeyd logs information about attacks in different log files. All connections to and from the honeypot are logged in files named *honeyd.log* and *msg.log*. The format, in which information is stored in *honeyd.log*, comprises of various fields. The structure of *honeyd.log* is shown below.

```
<date&time><protocol><connectioninfo><src_IP><src_port><dst_IP><dst_port><packet_size>
<flags><OS fingerprint>
```

The third field may either be *S, E* or -. *S* means the start of a new connection, *E* the end of a connection and - if a packet is not belonging to any connection. On lines with *E*, *honeyd* logs the

amount of data received and sent at the end of the line. A snippet of *honeyd.log* entries are as shown below.

```
2013-01-15-16:21:16.9308 honeyd log started ------
2013-01-16-06:35:41.5336 tcp(6) S 172.31.46.43 2064 172.31.6.109 445 [Windows XP SP1]
2013-01-16-06:35:41.5442 tcp(6) - 172.31.46.43 2064 172.31.6.109 445: 52 FA [Windows XP SP1]
2013-01-16-06:35:41.5457 tcp(6) S 172.31.46.43 2065 172.31.6.109 139 [Windows XP SP1]
2013-01-16-06:35:41.5458 tcp(6) - 172.31.46.43 2065 172.31.6.109 139: 48 S [Windows XP SP1]
2013-01-16-06:35:41.5643 tcp(6) E 172.31.46.43 2064 172.31.6.109 445: 0 0
2013-01-16-06:35:41.5643 tcp(6) - 172.31.46.43 2064 172.31.6.109 445: 52 A [Windows XP SP1]
2013-01-16-06:35:41.6549 tcp(6) - 172.31.46.43 2065 172.31.6.109 139: 124 PA [Windows XP SP1]
2013-01-16-06:35:57.2963 igmp(2) - 224.0.0.1 172.31.6.1: 28
2013-01-16-06:35:57.2963 igmp(2) - 224.0.0.1 172.31.6.1: 28
2013-01-16-06:35:57.6609 udp(17) - 172.31.6.21 137 172.31.6.255 137: 78
2013-01-16-06:35:57.6610 udp(17) - 172.31.6.21 137 172.31.6.255 137: 78
2013-01-16-06:35:57.9157 igmp(2) - 224.0.0.22 172.31.6.4: 32
2013-01-16-06:35:57.9157 igmp(2) - 224.0.0.22 172.31.6.4: 32
2013-01-16-06:35:58.2255 igmp(2) - 224.0.0.2 172.31.6.4: 32
2013-01-16-06:35:58.2256 igmp(2) - 224.0.0.2 172.31.6.4: 32
2013-01-16-06:35:58.9437 tcp(6) E 172.31.46.43 2065 172.31.6.109 139: 72 0
2013-01-16-06:35:58.9438 tcp(6) - 172.31.46.43 2065 172.31.6.109 139: 40 RA [Windows XP SP1]
2013-01-16-06:36:03.8389 udp(17) - 172.31.6.203 631 255.255.255.255 631: 174
```

The structure of *msg.log* is different from *honeyd.log*. Each instance of an attack is logged between `--MARK-- and --ENDMARK--`. An instance of packet captured by HIDESIGN is as shown below.

```
--MARK--,"Tue Jan 29 11:56:29 IST
2013","webmin/HTTP","172.31.6.129","172.31.6.41",8295,80,
"GET
/inexistent_file_name.inexistent0123450987.cfm%20%3Cscript%3Ealert(12345)%3C/script%3
E HTTP/1.0
User-Agent: Wget/1.11.4
```

```
Accept: */*
Host: 172.31.6.41
Connection: Keep-Alive
",
--ENDMARK--
```

Here, "%3Cscript%3Ealert(12345)%3C/script%3E" is equivalent to <script>alert(12345)</script>, which is an attempt of Cross Site Scripting attack. Below is another instance captured by emulated SMTP server.

```
--MARK--,"Thu Jan 17 17:37:47 IST 2013","exchange/SMTP","","",,,
"ehlo
mail from:sa@gmail.com
MAILFROM
MAILFROM SAN
MAILFROM=s@gmail.com
MAILFROM=gmail.com
MAILFROM:GMAIL.COM
",
--ENDMARK—
--MARK--,"Mon Jan 28 18:43:51 IST 2013","exchange/SMTP","","",,,
"helo
ehlo
mail_to
",
--ENDMARK--
```

Due to controlled environment of University's network, honeypot could not be exposed to outside world. For the purpose of data capturing, original web server logs of an educational institute for duration of 10 months have been used. These original requests were then replayed on to customized and emulated replica web server on honeypot. The details of steps followed are explained in the following subsections.

4.5.1 W3C Logs of Actual Web Server

For experimental purpose original web logs of an educational institute's *Microsoft IIS 7.0* web server have been used. These logs were in *W3C* format. *W3C* log format is default log file format on *IIS* server. It is a flexible, highly configurable web log format developed by the *World Wide Web Consortium (W3C)* as a common standard to support the needs of servers, clients, and proxies. Each *W3C* Extended format log file displays a header containing information about the data types recorded, as well as the version of the extended log file format used at the beginning of each log file. Entries consist of a sequence of fields relating to a single HTTP transaction. Fields are separated by whitespace or tab characters. If a field is unused in a particular entry dash "-" marks the omitted field. Directives record information about the logging process itself. Lines beginning with the # character contain directives. Table 4.1 enlists the directives defined in *W3C* log format.

Table 4.1 Brief description of directives in W3C log format

Sr. No.	Directive	Explanation
1.	Version: <integer>.<integer>	The version of the extended log file format used.
2.	Fields: [<specifier>...]	Specifies the fields recorded in the log.
3.	Software: string	Identifies the software which generated the log.
4.	Start-Date: <date> <time>	The date and time at which the log was started.
5.	End-Date:<date> <time>	The date and time at which the log was finished.
6.	Date:<date> <time>	The date and time at which the entry was added.
7.	Remark: <text>	Comment information.

Each log file entry consists of a sequence of fields separated by whitespace and terminated by a CR or CRLF sequence. The *#Fields* directive lists a sequence of field identifiers specifying the information recorded in each entry. These fields can be customized. But the general standard fields include *date, time, s-ip, cs-method, cs-uri-stem, cs-uri-query, s-port, cs-username, c-ip, cs(User-Agent), sc-status, sc-substatus, sc-win32-status* and *time-taken*. The brief description of these fields is given in Table 4.2.

Table 4.2 Brief description of fields in W3C log format

Field	Appears As	Description	Default Y/N
Date	Date	The date on which the activity occurred.	Y
Time	Time	The time, in coordinated universal time (UTC), at which the activity occurred.	Y
Client IP Address	c-ip	The IP address of the client that made the request.	Y
User Name	cs-username	The name of the authenticated user who accessed your server. Anonymous users are indicated by a hyphen.	Y
Service Name and Instance Number	s-sitename	The Internet service name and instance number that was running on the client.	N
Server Name	s-computername	The name of the server on which the log file entry was generated.	N
Server IP Address	s-ip	The IP address of the server on which the log file entry was generated.	Y
Server Port	s-port	The server port number that is configured for the service.	Y
Method	cs-method	The requested action, for example, a GET method.	Y
URI Stem	cs-uri-stem	The target of the action, for example, Default.htm.	Y
URI Query	cs-uri-query	The query, if any, that the client was trying to perform. A Universal Resource Identifier (URI) query is necessary only for dynamic pages.	Y
HTTP Status	sc-status	The HTTP status code.	Y
Win32 Status	sc-win32-status	The Windows status code.	N
Bytes Sent	sc-bytes	The number of bytes that the server sent.	N
Bytes Received	cs-bytes	The number of bytes that the server received.	N
Time Taken	time-taken	The length of time that the action took, in milliseconds.	N
Protocol Version	cs-version	The protocol version that the client used (HTTP or FTP)	N
Host	cs-host	The host header name, if any.	N
User Agent	cs(User-Agent)	The browser type that the client used.	Y
Cookie	cs(Cookie)	The content of the cookie sent or received, if any.	N
Referrer	cs(Referrer)	The site that the user last visited. This site provided a link to the current site.	N
Protocol Substatus	sc-substatus	The substatus error code.	Y

The following snippet shows partial web log used for experimentation in this work.

```
#Fields: date time s-ip cs-method cs-uri-stem cs-uri-query s-port cs-username c-
ip cs(User-Agent) sc-status sc-substatus sc-win32-status time-taken
2012-06-06 00:02:52 172.31.4.32 GET /images/hpic1.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 535
2012-06-06 00:02:52 172.31.4.32 GET /images/logo.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 268
2012-06-06 00:02:52 172.31.4.32 GET /images/hpic3.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 539
2012-06-06 00:02:53 172.31.4.32 GET /images/spacer.gif - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 265
2012-06-06 00:02:53 172.31.4.32 GET /images/logo.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 269
2012-06-06 00:02:53 172.31.4.32 GET /images/hpic4.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 1063
2012-06-06 00:02:53 172.31.4.32 GET /images/hpic5.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 1062
2012-06-06 00:02:53 172.31.4.32 GET /images/hpic6.jpg - 80 - 172.31.1.6
Opera/9.80+(J2ME/MIDP;+Opera+Mini/4.1.13907/24.871;+U;+en)+Presto/2.5.25+Version/
10.54 200 0 0 1063
```

4.5.2 GET Requests Extraction

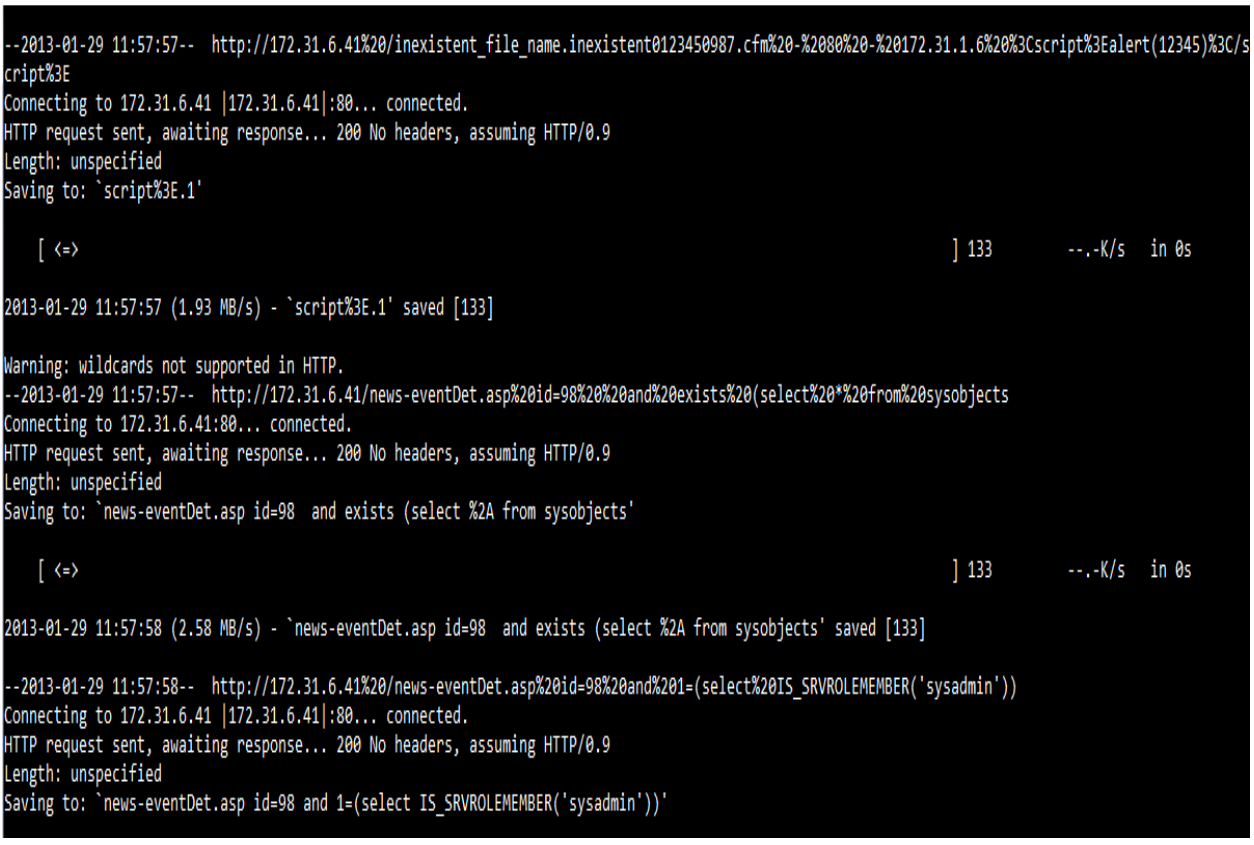
The *GET* requests from these logs are extracted and stored in a file by log digging engine implemented in *Java*. Example get request extracted for the purpose of replaying is as shown in the snippet below.

```
GET /js-collapse.js
GET /ddlevelsmenu-sidebar.css
GET /ddaccordion.js
```

```
GET /jquery-1.2.2.pack.js
GET /ddlevelsmenu.js
GET /images/hpic4.jpg
GET /images/hpic5.jpg
GET /images/hpic3.jpg
GET /ddaccordion.js
GET /images/hpic1.jpg
GET /images/hpic2.jpg
GET /images/dottedline.gif
```

4.5.3 Replaying GET Requests on Honeypot

These *GET* requests are replayed using *wget* utility on emulated web server of honeypot. Snapshot 4.1 shows snippet of execution of *wget* command on client machine to send *GET* requests to emulated web server.



Snapshot 4.1: Execution of *wget* utility on client machine

4.6 Misuse Detection

While requests are being replayed, MDE raises alarms for any known attacks. These alarms are stored in *MySQL* database. *Snort* has been used for this purpose. After running the *Snort* in NIDS mode, *Snort* stores the alerts generated by it in the *Snort* database. The alerts stored are retrieved from the tables of *Snort* database using *MySQL* queries. The anomalous packets are then filtered out from whole set of packets in *honeyd* logs on the basis of certain parameters matching criteria like date, timestamp *etc.* Remaining packets which are declared safe, *i.e.*, unidentified by MDE are sent to ADE for further inspection.

4.7 Anomaly Detection

Anomaly detection can be performed in many ways as reported in literature. In this thesis work, machine learning has been used for detecting any abnormal behavior. Machine learning based network intrusion detection is a valuable technology to protect target systems and networks against malicious activities. Despite the variety of such methods described in the literature in recent years, security tools incorporating anomaly detection functionalities are just starting to appear. Here, supervised machine learning approach has been followed in which a classifier is built at the time of training using labeled data. This classifier is used for the detection of abnormalities in the unlabeled data and classifies this data as per the learned classes.

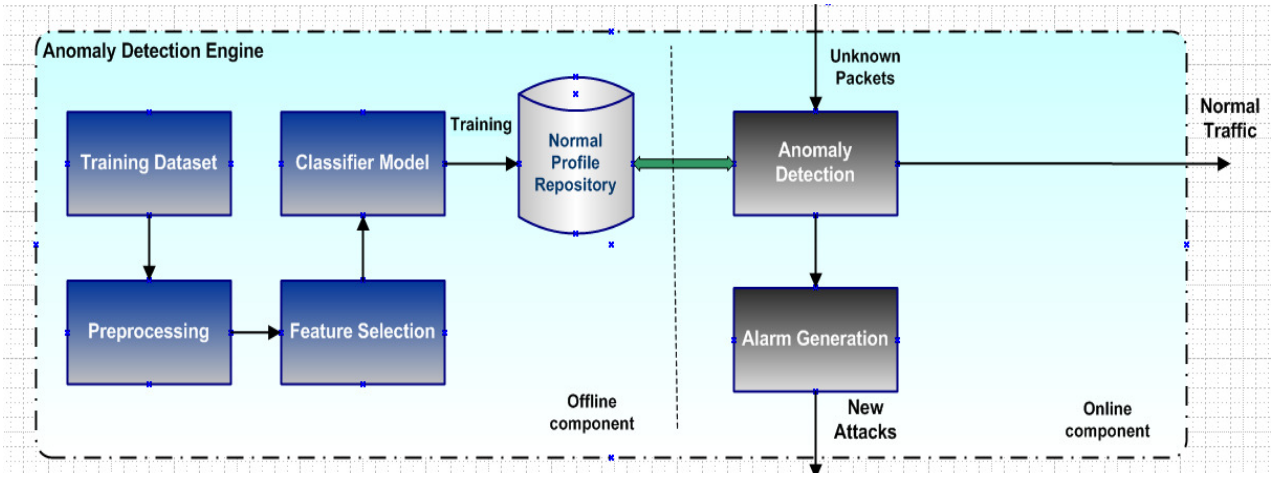


Figure 4.3: Architecture of ADE

Figure 4.3 can be referred for architecture of ADE. Details of components implementation and experiments performed are discussed in the following sub sections.

4.7.1 Data Set

Dataset plays an important role in the evaluation of Intrusion Detection Systems. Mostly, the evaluation is performed by standard datasets developed by some research group or self generated datasets. These may be captured and prepared using live network environment or automatically generated datasets containing synthetic attacks using some tools [Rub04].

For training of HTTP attacks, HTTP CSIC 2010 dataset has been used. It was developed at the "Information Security Institute" of Spanish Research National Council, CSIC (Consejo Superior de Investigaciones Científicas). This dataset contains thousands of web requests automatically generated for the testing of web attack protection systems. The HTTP dataset CSIC 2010 contains the generated traffic targeted to an e-Commerce web application developed at CSIC. In this web application, users can buy items using a shopping cart and register by providing some personal information. There are total 61065 HTTP packet objects with 223585 instances. The dataset includes attacks such as SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, Cross Site Scripting attack (XSS), parameter tampering *etc*. In this dataset, there are three categories of anomalous requests, namely, static attacks, dynamic attacks and unintentional illegal requests. Here, the static attacks comprise of requests to hidden resources such as obsolete files, session id in URL rewrite, configuration files, default files, while dynamic attacks are those which modify valid request arguments like SQL injection, CRLF injection, Cross Site Scripting, buffer overflows. The unintentional illegal requests are the ones that do not have malicious intention but have abnormal behavior. Snapshot 4.2 depicts some part of dataset in *.csv* format. It is a labeled data with various instances of HTTP packets categorized into two classes. The class labels for these two classes are '*norm*' and '*anom*' for normal and anomalous traffic, respectively. This dataset has 18 attributes including class label. The attributes are in an extended form of the HTTP/1.1 protocol [Fie09].

index	method	url	protocol	userAgent	pragma	cacheCont	accept	acceptEnc	acceptCha	acceptLa	host	connect	content	conter	cookie	payload	label
0	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	ID=1F767F17239C9E	norm
1	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	B1=Aï¿½adir al carri	norm
2	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	B1=Entrar	norm
3	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	id=2	norm
4	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	ID=42B53DB3F82E6	norm
5	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	errorMsg=Credencia	norm
6	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	ID=42B23D5507A66	norm
7	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	B1=Pasar por caja	norm
73	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	id=1%20	anom
74	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	idA=1	anom
75	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	ID=6F5B028AB85EA	anom
76	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	errorMsg=any%0D%	anom
77	GET	http://loca	HTTP/1.1	Mozilla/5.	no-cache	no-cache	text/xml,a	x-gzip, x-d	utf-8, utf-8	en	localhos	close	null	null	JSESSION	errorMsgA=Credenc	anom

Snapshot 4.2 : Snippet of HTTP CSIC 2012 dataset

The columns names are *index, method, url, protocol, userAgent, pragma, cacheControl, accept, acceptEncoding, acceptCharset, acceptLanguage, host, connection, contentLength, contentType, cookie, payload* and *label*. These features along with their description and count of unique and distinct occurrence in CSIC 2010 dataset has been given in Table 4.3.

Table 4.3: Attributes description and statistics in CSIC 2010 dataset

Sr. No.	Attribute	Description	Distinct values	Unique values
1.	Index	Index number to track the HTTP packet	36000	5412
2.	method	Methods like GET, POST, PUT	3	0
3.	url	URL of Host	1643	709
4.	Protocol	HTTP protocol	1	0
5.	userAgent	Request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated recognition of user agents	1	0
6.	pragma	General-header field is used to include implementation-specific directives that might apply to any recipient along the request/response chain	1	0

7.	cacheControl	Used to specify directives that MUST be obeyed by all caching mechanisms along the request/response chain	1	0
8.	accept	Used to specify certain media types which are acceptable for the response	1	0
9.	acceptEncoding	Similar to Accept, but restricts the content-codings that are acceptable in the response	1	0
10.	acceptCharset	Used to indicate what character sets are acceptable for the response	1	0
11.	acceptLanguage	Similar to Accept, but restricts the set of natural languages that are preferred as a response to the request	1	0
12.	host	Request-header field specifies the Internet host and port number of the resource being requested,	2	0
13.	connection	General-header field allows the sender to specify options that are desired for that particular connection and MUST NOT be communicated by proxies over further connections	1	0
14.	contentLength	Entity-header field indicates the size of the entity-body, in decimal number of OCTETs.	383	0
15.	contentType	Entity-header field indicates the media type of the entity-body	2	0
16.	cookie	Small piece of data sent from a website and stored in a user's web browser	61065	37184
17.	payload	Payload is the actual data that is carried on behalf of an application	471	0
18.	label	Class label: normal or anomalous	2	0

To train the offline component of ADE for detection of SMTP attacks, a subset of NSL-KDD dataset has been used. The NSL-KDD data set has been derived from original KDD CUP'99 data set, which itself was originally derived from DARPA Intrusion Detection evaluation dataset [Lip00], [Tho08], a well known and most widely used data set for anomaly detection and evaluation of IDS. NSL-KDD consists of selected records of the complete KDD data set. It differs from and is better than KDD CUP'99 dataset as it comprises of non-redundant, reasonable number of records in train and test set [Tav09].

It comprises of 4 categories of attacks, namely, DoS (Denial of Service), Probe, R2L (Remote to Local), and U2R (User to Root). A brief description of these attack categories is presented below.

- Denial of service (DOS): It contains category of attacks which prevent normal operation, such as causing the target host or server to crash, blocking network traffic, denying legitimate users access to a machine, making some computing / memory resource too busy or too full to handle legitimate requests.
- Probes: Probing attack is an attempt to test a potential target to gather information about a network and its resources for the apparent purpose of circumventing its security controls. These attacks are common and usually harmless unless some vulnerabilities are discovered and later exploited.
- Remote to local (R2L): In this category of attacks, an unauthorized user is able to bypass normal authentication and execute commands on the target. R2L attack occurs when an unauthorized attacker exploits some vulnerability to gain local access as a user of a machine.
- User to root (U2R): In this class of exploits, the attacker starts with access to a normal user account on the system possibly gained by sniffing passwords, a dictionary attack, or social engineering and is able to exploit some vulnerability to gain root access to the system.

This dataset consists of normal and attack instances related to various protocols and services. We have used a subset of this dataset and preprocessed it to extract only SMTP related instances. There were total 7313 instances for training categorised as normal or anomalous. There were 7029 normal and 284 anomolous instances. The NSL-KDD data includes 41 features enlisted in Table 4.4.

Table 4.4: NSL-KDD dataset attributes description

Sr. No.	Attribute	Description
1	duration	Duration of the connection
2	protocol_type	Connection protocol (*e.g.* tcp, udp)

3	service	Destination service (*e.g.* telnet, ftp)
4	flag	Status flag of the connection
5	src_bytes	Bytes sent from source to destination
6	des_bytes	Bytes sent from destination to source
7	Land	1 if connection is from/to the same host/port; 0 otherwise
8	wrong_fragment	number of wrong fragments
9	Urgent	number of urgent packets
10	hot	number of "hot" indicators
11	num_failed_logins	number of failed logins
12	logged_in	1 if successfully logged in; 0 otherwise
13	num_compromised	number of "compromised'' conditions
14	root_shell	1 if root shell is obtained; 0 Otherwise
15	su_attempted	1 if "su root'' command attempted; 0 otherwise
16	num_root	number of "root'' accesses
17	num_file_creations	number of file creation operations
18	num_shells	number of shell prompts
19	num_access_files	number of operations on access control files
20	num_outbound_cmds	number of outbound commands in an ftp session
21	is_host_login	1 if the login is a “host” list; 0 otherwise
22	is_guest_login	1 if the login is a "guest'' login; 0 otherwise
23	Count	number of connections to the same host as the current connection in the past two seconds
24	srv_count	number of connections to the same service as the current connection in the past two seconds

25	serror_rate	% of connections that have "SYN'' errors
26	srv_serror_rate	% of connections that have "SYN'' errors
27	rerror_rate	% of connections that have "REJ'' errors
28	srv_rerror_rate	% of connections that have "REJ'' errors
29	same_srv_rate	% of connections to the same Service
30	diff_srv_rate	% of connections to different services
31	srv_diff_host_rate	% of connections to different hosts
32	dst_host_count	count of connections having the same destination host
33	dst_host_srv_count	count of connections having the same destination host and using the same service
34	dst_host_same_srv_rate	% of connections having the same destination host and using the same service
35	dst_host_diff_srv_rate	% of different services on the current host
36	dst_host_same_src_port_rate	% of connections to the current host having the same src port
37	dst_host_srv_diff_host_rate	% of connections to the same service coming from different hosts
38	dst_host_serror_rate	% of connections to the current host and specified service that have an S0 error
39	dst_host_srv_serror_rate	% of connections to the current host and specified service that have an S0 error
40	dst_host_rerror_rate	% of connections to the current host that have an RST error
41	dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error
42	Class	Class label as normal or anomaly

4.7.2 Feature Selection

Feature selection is used to find an optimal subset of features that are most relevant to data mining task. We have used Ranker's algorithm for removal of irrelevant features for fast and effective prediction model. In HTTP CSIC dataset, out of 17 (excluding class label) features only 8 have been selected for building the classifier as other attributes has negligible role in predicting the traffic as normal or anomalous. The InfoGain score for the attributes selected by Ranker's algorithm for CSIC dataset have been given in Table 5.1 of Section 5.3. Feature selection was also applied on NSL-KDD dataset. Here, out of 41 features 26 have been selected on the basis of InfoGain score as given in Table 5.5 of Section 5.4.

4.7.3 Building Classifier Model

ADE uses Naive Bayes for building classifier model to train the system. Naive Bayes classification has been used in many of machine learning based Intrusion Detection Systems and .has proved to be very promising. Naive assumes the conditional probabilities of the independent variables. Let *X* is an object to be classified and *Ci'*s are the possible classes in which object *X* can be classified, the classifier calculates *P(Ci | X)* for all the classes and assigns *X* to the class that has the highest conditional probability. PseudoCode 4.1 describes the process of learning using Naive Bayes classifier.

PseudoCode 4.1: Training phase using Naive Bayes classifier

begin

Given the set of training packet object instances *T*

for (Each target value of class c_i (c_1,c_2,c_3…….c_n))

Estimate P(C=c_i) with example instances in *T*

end-for

for (Every data variable *X*)

Estimate P (*X*| *C*=*ci*) with example instances in *T*

end-for

Construct conditional probability tables.

end

For training the system with Bayesian classifiers, WEKA has been used. WEKA [Hal09] is known open-source machine learning and data mining environment and can serve as a good tool for a preliminary test of the Naive Bayes classifier on network data. Anomaly detection engine has been trained using 10 fold cross validation approach. In *k*-fold cross-validation, the data is randomly divided into *k* disjoint subsets of approximately equal size. One of the subsets is then used as the test set and remaining *k-1* sets are used for building the classifier. The test set is then used to estimate the accuracy. This is done repeatedly *k* times so that each subset is used as test subset once. The accuracy estimate is then the means of the estimates for each of the classifiers. A value of 10 for *k* has been found to be adequate and accurate.

4.7.4 Testing

Here the classifier, which was built for the training of system, is used for classification of unknown packets. Let us denote an instance of packet object as *A* and possible classes to which *A* can be assigned as *Ci*. The classifier calculates *P(Ci | A)* for all classes and will assign *A* to the class that has the highest conditional probability. The classification using Naive Bayes is given in PseudoCode 4.2.

PseudoCode 4.2: Classification of unknown packet objects

begin
  for (Each unknown instance *i* of packet object *A)*
    Traverse probability tables to assign class *C'* to *Ai*
    if [P(*Ai*|*C'*) >P(*Ai*|*C*)] where *C = (C1, C2, . . ., Cn), C != C'*
      Return *C'* as the class label of *Ai*;
    end-if
  end-for
end

The newly detected anomalies are forwarded to SGE so that the new intrusion patterns can be built for misuse detection.

4.8 Signature Generation

Signature generation is indeed an important task of HIDESIGN. The major objective of HIDESIGN is generation of robust and valid signatures. Signature Generation Engine generates content based signatures. It has been implemented in *Java*. Given the set of malicious packet pool, SGE is responsible for extraction of content strings, building signatures and storing them in signatures repository of NIDS.

The procedure of generation and storage of signatures in described in PseudoCode 4.3 and PseudoCode 4.4. The input to this procedure is set of strings in malicious pool. Here, generalized suffix tree is built to find out Longest Common Substring which is occurring in at least λ percent of pool and its length l should be at least greater than or equal to a threshold value *'TH'*. This function returns string *str*, used to generate the signatures in a format that is understood by *Snort* NIDS. PseudoCode 4.4 explains the procedure of building and appending of signatures to repository. These signatures are appended in rule file represented as *SigFile*. SGE also takes care of quality and redundancy of signature. If new signature is same as existing signature in repository, no new signature is generated. If it is a superset of already existent signature, or it is not present already then new signature is appended to repository.

PseudoCode 4.3: GenerateSignatures()

```
begin
    for (each string Mi in malicious pool M)
        Build suffix tree using Ukkonen Algorithm.
        Find content string str using Longest Common Substring algorithm
        calculate count_per, length(str)
        if (length(str)>=TH AND count_per >= λ)
            return str;
        end if
        BuildAppend Signatures(str);
    end for
end
```

PseudoCode 4.4: BuildAppend Signatures(str)

Given: Signature Repository *SigFile*

INPUT: String *str* returned by LCS

OUTPUT: Signature *NewSig*

begin

for (Each signature Sig_i in *SigFile*)

search *str* in signature repository *SigFile*

return Sig_i

found = True

end for

if (found)

Extract *content* from Sig_i

if (strcmp(*str*, *content*)!=0)

NewSig= BuildNewSignature()

else

continue

end if

else

NewSig=BuildNewSignature()

end if

append signature NewSig to SigFile

end

BuildNewSignature() generates the signatures in *Snort* Rule format. SGE makes use of regular expressions and pattern matching to generate message part in *Snort* signatures. For instance, if LCS finds "<script>" or "%3Cscript%3E" or any other combination of this as a resultant string, Signature Generation Engine will generate *Snort* alert with a message of "Cross Site Scripting Attack" because "%3C" is hexadecimal equivalent for opening angle bracket and "%3E" is hexadecimal equivalent for closing angle bracket.

The *Java* code snippet for this matching through regular expression can be one shown in (4.1).

```
Pattern
pattern=Pattern.compile("((%3c)|(%3C)|<|(&lt;))"+"script"+"((%3e)|>|(%3E)|(&gt;))"+"(
.*?)"+"((%3c)|<|(%3C)|(&lt;))"+"((/)|(%2F)|(%2f))"+"script"+"((%3e)|>|(%3E)|(&gt;))",
Pattern.CASE_INSENSITIVE);                                                    (4.1)
```

If no such match is found it simply adds a default message "*New Intrusion Detected*" to *Snort*'s signature. These rules are appended in m*yrules.rules* file of *Snort* rules directory for future reference as shown in Snapshot 4.3. This file has been referred to as *SigFile* in PseudoCode 4.4.

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"SQL injection attack"; flow:established,to_server; content:"sel
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Command Execution ATTACK"; content:"/winnt/system32/cmd.exe"; r
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"/dir-desk-conf-work.asp
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"ilang=eng&SID=2%22%3Csc
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"rtn=1"><script>alert("4
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"do_search=Search&search
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"<script>alert("4078643
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"%3Cscript%3Ealert('wvs-
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"test=<script>alert(1234
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" PGV_BASE_DIRECTORY=../.
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:"/..0x5c..0x5c..0x5c..0x5
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:"/..%5c..%5c..%5c..%5c..%
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:"/..%5c..%5c..%5c..%5c..%
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:"/..%2f..%2f..%2f..%2f..%
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" url=../../../../../../.
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:"select%20*%20from%20[thprdb]..
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:"select%20*%20from%20[thprdb]..
```

Snapshot 4.3: Signatures being appended to *myrules.rules*

This file is to be included in configuration file of *Snort* named as *snort.conf*. The process of signature generation and appending is repeated after regular intervals so that the repository gets updated in a timely manner.

4.9 Overall Working of HIDESIGN

The working of HIDESIGN is elaborated in PseudoCode 4.5. The overall objective of HIDESIGN is to generate signatures of malicious packets in the network traffic which were earlier unknown by NIDS. For this purpose it makes used of hybrid approach for detection of newely posed attacks to the network and then automatically generate the signatures for these attacks.

In order to understand the overall working of HIDESIGN, let us consider,

P: {P1, P2, P3,…} be the set of all incoming packets.

S: {S1, S2, S3,…} be the suspicious pool of traffic.

N: {N1, N2, N3...} be the normal pool of traffic.

Then, $P = S \wedge N$.

PseudoCode 4.5: Overall working of HIDESIGN

Input: Suspicious traffic pool *S= {S1, S2, S3,...}* and normal traffic pool *N = {N1, N2, N3,...}*

Output : Set of new attack signatures *NewSig = {NSig1, NSig2, NSig3,...}*

begin

for (Each packet *Pi* in the incoming network traffic *P*)

{

receive packet in hoenypot server

log packet in suspicious pool *S* (*S1, S2, S3*,…)

for (each *ith* packet in suspicious pool *S*)

Match signatures of *Si* with signatures Sig_i in IDS repository

if (match(*Si*, Sig_i)==True)

Alert/log/drop packet *Si*

else

// forward packet to ADE

input packet to classifier // Classify packet based on classifier model trained in offline mode of ADE (PseudoCode 4.1)

assign Class *C'* to packet object *Si*

if (class==normal)

Declare packet to be normal

else

Add packet to malicious pool *M(M1, M2, ….)*

end if

end if

end for

end for

// Signature Generation process GenerateSignatures() as given in PseudoCode 4.3 repeated after short intervals.

```
for (each string Mi in malicious pool M)
        Build suffix tree().
        Find content string str using Longest Common Substring algorithm
        calculate count_per, length(str)
        if (length(str)>=TH AND count_per >= λ)
                return str;
        end if
end for
for (Each signature Sig_i in SigFile)
        search str in signature repository SigFile
        return Sig_i
        found = True
end for
if (found)
        Extract content from Sig_i
        if (strcmp(str, content)!=0)
                NewSig= BuildNewSignature()
        else
                continue
        end if
else
        NewSig= BuildNewSignature()
end if
        append signature NewSig to SigFile
end
```

In this procedure, the overall working of HIDESIGN has been explained. It makes use of PseudoCodes for individual working of various modules discussed in the previous sections. All the packets destined to honeypot are considered to be suspicious and are put into Suspicious

traffic pool *S* (*S1, S2, S3*,…). These are searched for the known attacks by Misuse Detection Engine. The remaining packets are inspected by ADE which is trained in offline mode using labeled dataset through supervised machine learning. In online mode also known as testing phase of ADE, the unknown packets are classified to be normal or anomalous by the classifier. Once the classes are assigned, packets which are declared anomalous by ADE are considered to be malicious packets and are appended in malicious pool *M (M1, M2, M3,….).* This pool is taken by Signature Generation Engine as input for building the content of signatures. It is done by using generalized suffix tree based LCS algorithm. The signatures are built and stored in repository for further reference to extend detection capabilities of Intrusion Detection Systems.

Chapter Summary

In this chapter the implementation details of HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator) for automatic generation and tracking of attack signatures have been discussed. Deployment and experimental setup details of HIDESIGN are also presented in this chapter. Elaboration of various implementation steps like honeypot configuration, data capturing, misuse and anomaly detection has been given. The procedural details supporting implementation have been provided using various PseudoCodes. Signature generation procedure and overall working of the system have also been elaborated in this chapter.

Chapter 5
Results and Discussions

5.1 Introduction

This chapter presents the results and discussions on the work carried out in this thesis. This work, as mentioned earlier, has focused on the development of hybrid Intrusion Detection and Signature Generation System named HIDESIGN. The architecture, development process and implementation of HIDESIGN have been explained thoroughly in the previous chapters. Apart from the process that was followed, the outcomes of HIDESIGN have also been discussed. The performance evaluation of HIDESIGN has been presented in this chapter. The testing of HIDESIGN is an important aspect in order to establish the usefulness of the system. The evaluation of proposed system has been performed using various evaluation metrics explained in the next section. These metrics have been selected in such a way that the proposed system can be tested from several aspects of Intrusion Detection System. The results of proposed work on various metrics like Sensitivity, Specificity, False Positive Rate, Accuracy, Detection Rate, F-Measure and ROC (Receiver Operating Characteristics) are very encouraging. The results of this experimentation are presented in subsequent sections.

5.2 Evaluation Metrics

The evaluation of Intrusion Detection Systems is important and depends upon several factors. The most basic of these factors are the false alarm rate and detection rate. However, the information provided by detection rate and false alarm rate alone might not be enough for performance evaluation of an IDS. Mere accuracy metric for evaluating Intrusion Detection Systems may also lead to some misconceptions. Several performance metrics which do not depend on the size of the test set are generally used for evaluating these systems. These include metrics such as Precision, Recall, F-Measure and Area under ROC Curve [Bhu14].

To evaluate and validate the performance of proposed framework, HIDESIGN, various metrics have been used. There are several notations used in the following subsections like TP, TN, FP and FN. Their intuitive meanings are as per following definitions.

TP (True Positive) is the number of attacks that are correctly detected, FN (False Negative) is the number of attacks that are not detected, TN (True Negative) is the number of normal traffic packets that are correctly classified, and FP (False Positive) is the number of normal traffic packets that are incorrectly detected as attack. The experimental results to measure the performance of prediction models have been evaluated by using following metrics.

- **Sensitivity**

This metric is used to measure correctness of the Intrusion Detection System. Sensitivity, also known as True Positive Rate (TPR) specifies the percentage of malicious packets which are correctly classified as anomalous. This measure is also called as Recall and specifies what fraction of attack class is correctly detected. Sensitivity is calculated using the formula as given in (5.1).

```
Sensitivity = TP / (TP + FN)                                    (5.1)
```

- **Specificity**

Specificity, also known as True Negative Rate (TNR) is the number of legitimate packets which are correctly identified as normal. Specificity can be defined as given in (5.2).

```
Specificity = TN / (FP + TN)                                    (5.2)
```

- **Precision**

Precision is defined as the ratio of correctly predictable attacks (True Positives) to the number of all the recognized attacks (both True Positives and False Positives). It is the percentage of positive predictions those are correct. It is also known as Detection Rate and can be calculated as given in (5.3).

```
Precision = TP / (TP + FP)
```
(5.3)

- **F-Measure**

F-Measure is a measure of the accuracy of a test that maintains the balance between precision and recall. The F-Measure values are computed as a harmonic mean of the precision and recall as shown in (5.4). The highest value of F-Measure would be considered as the best case whereas lowest value is evaluated as worst case.

```
F-Measure = 2 * Precision * Recall / (Precision + Recall)
```
(5.4)

- **ROC**

ROC (Receiver Operating Characteristic) area is a graphical plot to illustrate the performance of a classifier system based on variation in discrimination threshold value. It is created by plotting the fraction of true positives out of the total actual positives at various threshold settings. The accuracy of prediction approaches can be estimated by comparing their ROC curves in graphical approach. The Area Under Curve (AUC) method is used to access the ROC curve as given in (5.5).

$$\text{AUC}=\int_0^1 \text{TPR} * (\text{FPR})\, d\text{FPR} \in [0,1] \quad (5.5)$$

Here, TPR is the True Positive Rate and FPR specifies False Positive Rate. The maximum value of area under curve signifies the best predictor. A random IDS has an area of 0.5 whereas an ideal system has an area of one.

- **Accuracy**

A commonly used IDS evaluation metric on test data is the overall accuracy. It is the percentage of predictions that is correct. It can be calculated as a ratio of all true predictions, *i.e.*, both True Positives and True Negatives by all predictions as given in (5.6).

```
Accuracy = (TP + TN) / (TP + TN + FP + FN)
```
(5.6)

- **Confusion Matrix**

It is a matrix that visualizes the outcome of a supervised machine learning algorithm. The structure of a typical confusion matrix is shown in Figure 5.1. Each column of the matrix represents the instances in a predicted class (depicted with Classified as + or Classified as -), while each row represents the instances in an actual class (depicted with Is + or Is -). The aim is to compare the actual class labels against the predicted ones. The diagonal represents correct classification. The confusion matrix for intrusion detection is defined as a 2 X 2 matrix, since there are only two classes known as anomalous and normal.

	Classified as +	**Classified as –**
Is +	true positives (tp)	false negatives (fn)
Is –	false positives (fp)	true negatives (tn)

Figure 5.1: General structure of a Confusion Matrix

5.3 Results and Discussions in context of HTTP attacks

For detection of web attacks offline components of Anomaly Detection Engine (ADE) has been trained with Naive Bayes classifier. Here, CSIC 2010 dataset has been used for training. There are total 61065 HTTP packet objects with 223585 instances. The dataset includes attacks such as SQL injection, buffer overflow, information gathering, files disclosure, CRLF injection, Cross Site Scripting attack (XSS), parameter tampering *etc.* These HTTP requests are labeled as normal or anomalous. In this dataset, there are three categories of anomalous requests namely, static attacks, dynamic attacks and unintentional illegal requests. The details about this dataset along with attributes description are elaborated in Section 4.7.1.

There are total 17 features in HTTP CSIC 2010 dataset (excluding class label). Out of these only 8 have been selected for training the classifier based on information gain score. Information gain gives an instinct of importance of a given attribute of the feature vectors. The information gain scores of these features are given in Table 5.1. The top eight scorers have been selected as in other cases the InfoGain was negligible.

Table 5.1: Selected features with InfoGain score for HTTP dataset

Feature name	InfoGain Score
Index	0.29471
Method	0.01273
URL	0.12669
ContentLength	0.10206
ContentType	0.00492
Cookie	0.99649
Payload	0.13828
Host	0.00892

The prediction results of classifier model have been presented in the form of confusion matrix as shown in Table 5.2. Out of 104001 normal instances, 103977 were correctly classified whereas 23 were incorrectly classified. There were 119586 total anomalous request instances out of which 117432 were correctly classified and 2153 were incorrectly classified.

Table 5.2: Confusion matrix of experimental results on HTTP dataset

Class	Classified as Normal	Classified as Anomalous
Is Normal	103977	23
Is Anomalous	2153	117432

The overall accuracy statistics in terms of correctly classified instances, incorrectly classified instances, Kappa statistic, mean absolute error, root mean squared error, relative absolute error and root relative squared error of classifier are given in Table 5.3.

Table 5.3: Overall accuracy statistics of HTTP training data

Overall Accuracy Statistics	Description	Result
Correctly Classified Instances	Instances those were correctly classified	99.0268%

Incorrectly Classified Instances	Instances those were incorrectly classified	0.9732%
Kappa statistic	Kappa statistic is a statistical measure of degree to which two or more raters, examining the same data, agree when it comes to assigning the qualitative data to categories.	0.9805
Mean absolute error	Mean Absolute Error (MAE) is an average of the absolute errors.	6.5%
Root mean squared error	Root-mean-square error is a measure of the differences between values predicted by a model and the values actually observed.	11.61%
Relative absolute error	The relative absolute error is very similar to the relative squared error. This is just the total absolute error instead of the total squared error into consideration.	13.0548%
Root relative squared error	The relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the predictor	23.2779%

It was observed that the classifier model could classify 99.0268% instances correctly, whereas 0.97% instances were incorrectly classified. The results of error rates are depicted in Figure 5.2.

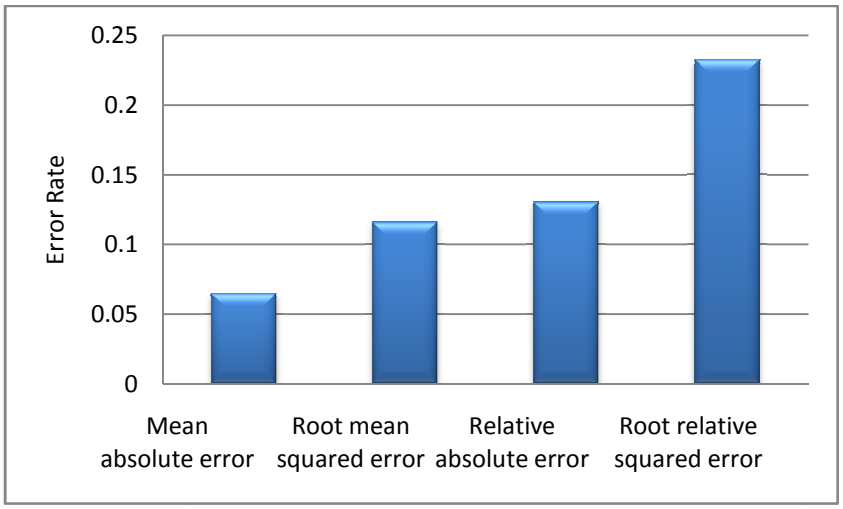


Figure 5.2: Results of offline component training model for HTTP attacks in terms of error rates

The detailed results in terms of TP, TN, FP, FN, Recall (TPR/Sensitivity), False alarms rate (FPR, FNR), Specificity (TNR), Accuracy, Precision (Detection Rate), F-Measure and ROC Area for HTTP are summarized in Table 5.4. Here, TP rate is the percentage of positive labeled

instances those were predicted as positive; FP rate is the percentage of negative labeled instances those were predicted as negative; Precision is the number of correct results divided by the number of all returned results; Recall is the number of correct results divided by the number of results that should have been; F-measure is the measure of a classifier's accuracy that considers both the Precision and the Recall of the classifier to compute the score.

Table 5.4: Detailed Results of the classifier model for HTTP training

Measure	**Intuitive Meaning**	**Value**
True Positive (TP)	True Positive is the total correctly classified objects	103977
True Negative (TN)	True Negative is the total number of objects that did not get classified to a class they did not belong to.	117432
False Positive (FP)	False Positive cases are those that did not belong to a class but were allocated to.	2153
False Negative (FN)	False Negatives are the cases that belong to a class but were not allocated to it.	23
True Positive Rate/ Sensitivity (%)/Recall	The percentage of positive labeled instances those were predicted as positive. TPR=TP/TP+FN	99.97%
False Positive Rate (%)	FPR=FP/FP+TN	1.8%
False Negative Rate (%)	FNR=FN/TP+FN=1-TPR	0.02%
True Negative Rate/ Specificity (%)	The percentage of negative labeled instances that were predicted as negative. TNR=TN/FP+TN=1-FPR	98.19%
Accuracy (%)	The percentage of predictions that is correct. (TP + TN) / (TP + TN + FP + FN)	99.02%
Detection Rate (%)/Precision	The percentage of positive predictions those are correct. TP / (TP + FP)	97.97%
F-Measure (%)	F-Measure = 2 * Precision * Recall / (Precision + Recall)	99%
ROC Area	AUC=$\int_0^1 \mathrm{TPR} * (\mathrm{FPR})\, \mathrm{dFPR} \in [0,1]$	0.99

It could be observed from the results presented in Table 5.4 that classifier model has appreciable Precision and Recall. The model has low False Positive Rate and False Negative Rate.

HIDESIGN has been tested with original web logs of a University's web server for duration of 10 months. These logs contained various known and unknown web attacks including SQL Injection, Cross Site Scripting, Directory Traversal, Command Injection and other web attacks. These attacks were successfully detected by HIDESIGN. The distribution of attacks detected from these web logs in each month has been provided in Figure 5.3.

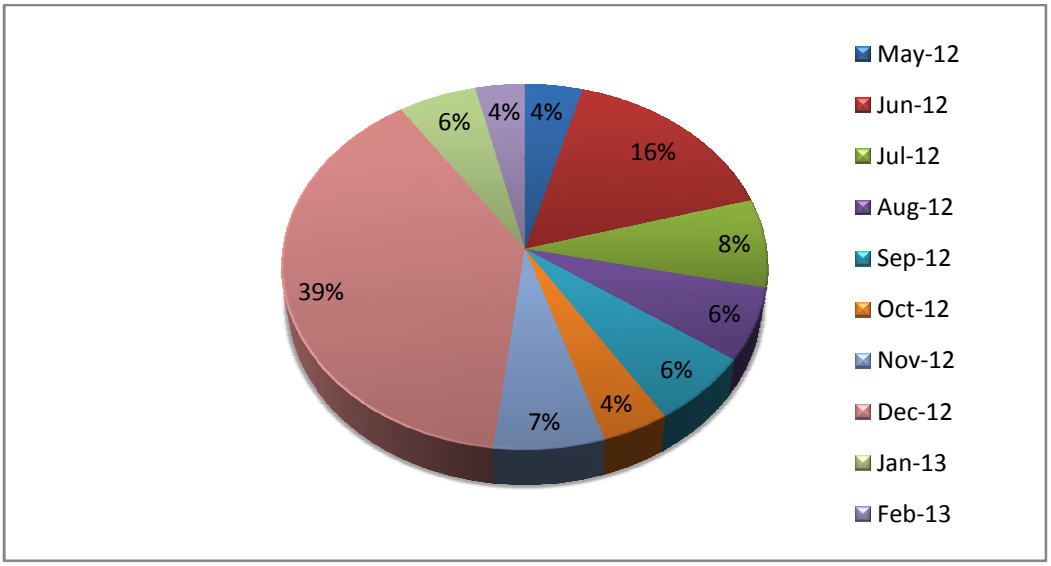


Figure 5.3: Distribution of attack instances reported by HIDESIGN month-wise

Figure 5.4 depicts the trend of various categories of attacks. The most common type of attack found on this web server is SQL injection attack. It has further been observed that the attacks trend is high in the month of June and December. It may be due to semester evaluation in these months.

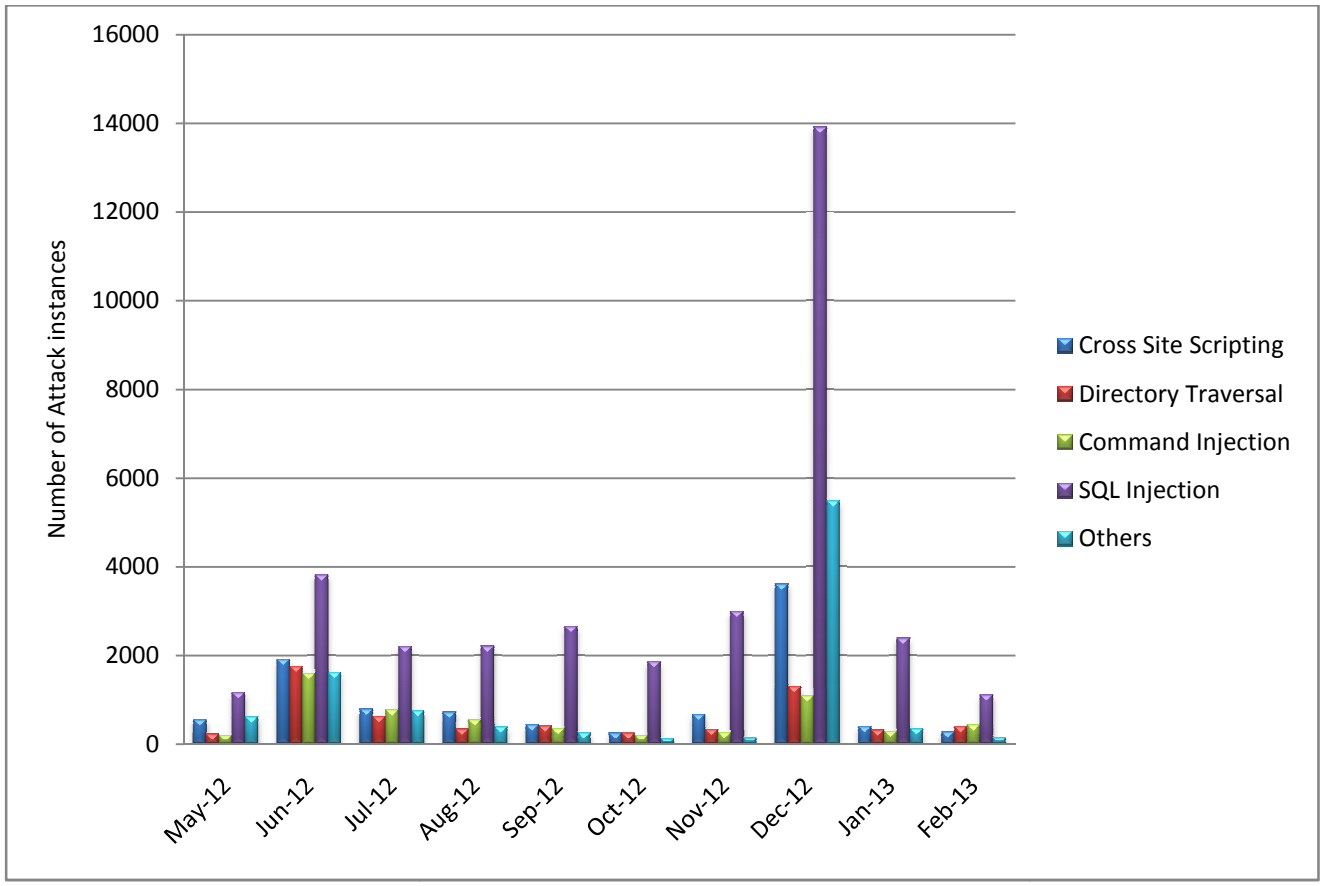


Figure 5.4: Monthly attack instances found category wise

5.4 Results and Discussions in context of SMTP attacks

For detection of SMTP attacks, the offline component of HIDESIGN was trained using subset of NSL-KDD dataset. The NSL-KDD data set has been derived from KDD CUP'99 data set, a well known data set for anomaly detection and evaluation of IDS. NSL-KDD consists of selected records of the complete KDD data set. It comprises of 4 categories of attacks, namely, DoS (Denial of Service), Probe, R2L (Remote to Local), and U2R (User to Root). The SMTP packets from this dataset were extracted. There are total 7313 SMTP instances for training out of which 7029 are normal and 284 are anomolous instances. This dataset consists of 41 features excluding class label. More details about this dataset and its features are given in Section 4.7.1. Here, out of 41 features, 26 have been selected on the basis of InfoGain score as calculated by Ranker's algorithm. The selected features along with their InfoGain score have been presented in Table 5.5.

Table 5.5: InfoGain Score of relevant features selected for SMTP dataset

Feature Name	InfoGain Score
dst_bytes	0.182291
src_bytes	0.176489
Flag	0.160737
Count	0.158816
logged_in	0.152726
dst_host_serror_rate	0.129643
dst_host_srv_serror_rate	0.123746
srv_serror_rate	0.121369
serror_rate	0.119564
dst_host_same_srv_rate	0.103696
srv_count	0.092935
diff_srv_rate	0.091059
same_srv_rate	0.089119
dst_host_srv_count	0.081697
srv_diff_host_rate	0.043205
dst_host_diff_srv_rate	0.039936
dst_host_rerror_rate	0.038664
dst_host_srv_rerror_rate	0.026866
rerror_rate	0.025725
srv_rerror_rate	0.023097
duration	0.022906
dst_host_count	0.022235
dst_host_same_src_port_rate	0.016201
dst_host_srv_diff_host_rate	0.006073
+num_file_creations	0.000589
num_access_files	0.00055

For building SMTP classifier model also, the offline component is trained using Naive Bayes classifier. The confusion matrix of training results is given in Table 5.6.

Table 5.6: Confusion matrix for SMTP datset training

Class	Classified as Normal	Classified as Anomalous
Is Normal	6806	223
Is Anomalous	19	265

The overall accuracy statistics is presented in Table 5.7. It can be observed from the result statistics that 96.6% instances were correclty classified whereas 3.3% were incorrecly classified.

Table 5.7: Overall accuracy statistics for SMTP training

Overall Accuracy Statistics	Description	Results
Correctly Classified Instances	Instances those were correctly classified	96.6908 %
Incorrectly Classified Instances	Instances those were incorrectly classified	3.3092 %
Kappa statistic	Kappa statistic is a statistical measure of degree to which two or more raters, examining the same data, agree when it comes to assigning the qualitative data to categories.	0.6703
Mean absolute error	Mean Absolute Error (MAE) is an average of the absolute errors.	3.41%
Root mean squared error	Root-mean-square error is a measure of the differences between values predicted by a model and the values actually observed.	18.22%
Relative absolute error	The relative absolute error is very similar to the relative squared error. This is just the total absolute error instead of the total squared error into consideration.	35.5491 %
Root relative squared error	The relative squared error takes the total squared error and normalizes it by dividing by the total squared error of the predictor	44.286 %

Various error rates in terms of mean absolute error, root mean squared error, relative absolute error and root relative squared error of SMTP prediction model are depicted in Figure 5.5.

Figure 5.5: Error rates for SMTP training

The detailed results in terms of TP, TN, FP, FN, Recall (TPR/Sensitivity), False alarms rate (FPR, FNR), Specificity (TNR), Accuracy, Precision (Detection Rate), F-Measure and ROC Area are summarized in Table 5.8.

Table 5.8: Detailed Results of the classifier model for SMTP

Measure	**Intuitive Meaning**	**Value**
True Positive (TP)	True Positive is the total correctly classified objects	6806
True Negative (TN)	True Negative is the total number of objects that did not get classified to a class they did not belong to.	265
False Positive (FP)	False Positive cases are those that did not belong to a class but were allocated to.	19
False Negative (FN)	False Negatives are the cases that belong to a class but were not allocated to it.	223
True Positive Rate/ Sensitivity (%)	The percentage of positive labeled instances those were predicted as positive. TPR=TP/TP+FN	96.83%
False Positive Rate	FPR=FP/FP+TN	6.69%

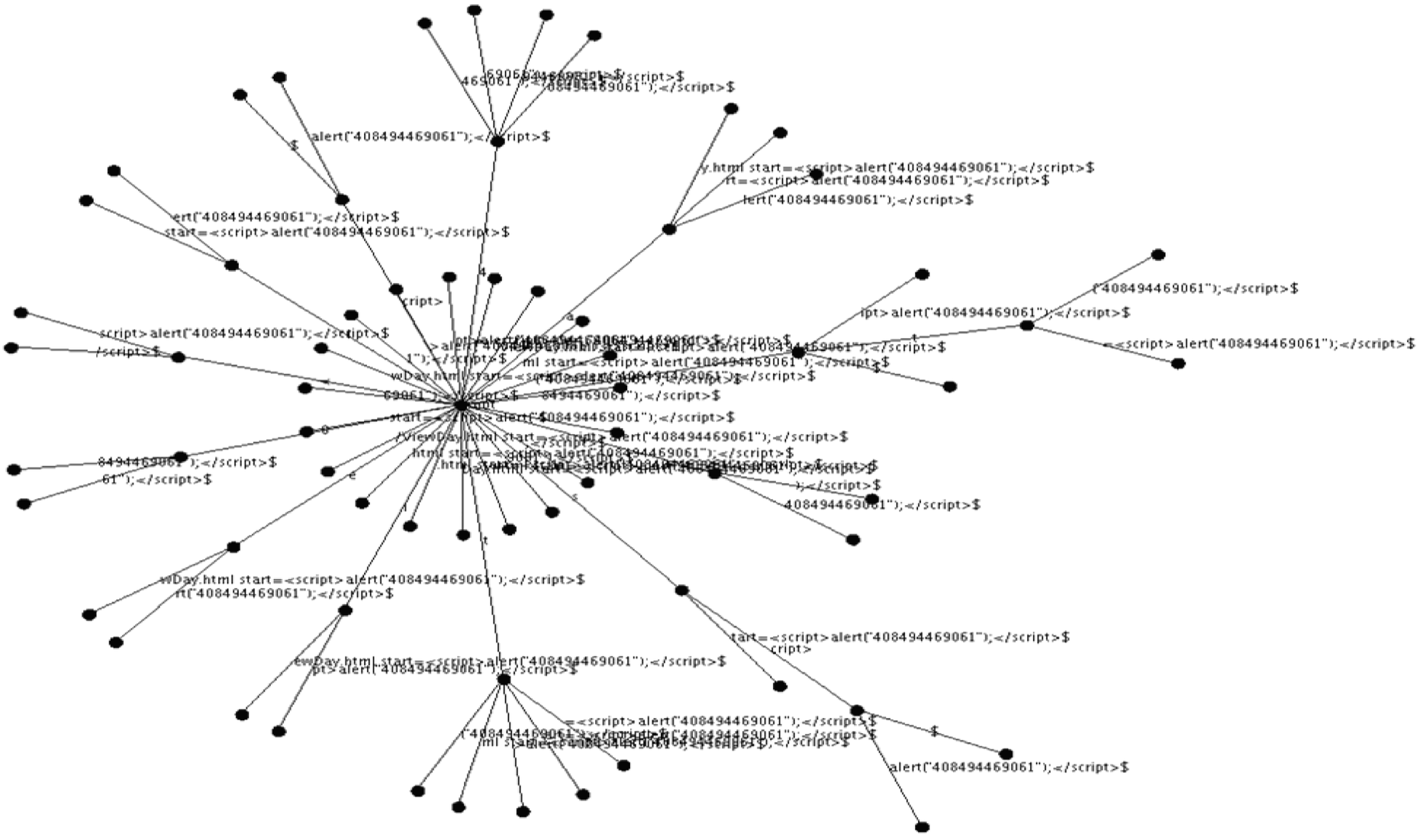
False Negative Rate	FNR=FN/TP+FN=1-TPR	3.17%
True Negative Rate/ Specificity (%)	The percentage of negative labeled instances that were predicted as negative. TNR=TN/FP+TN=1-FPR	93.3%
Accuracy (%)	The percentage of predictions that is correct. (TP + TN) / (TP + TN + FP + FN)	96.69%
Detection Rate (%)	The percentage of positive predictions those are correct. TP / (TP + FP)	99.7%
F-Measure	F-Measure = 2 * Precision * Recall / (Precision + Recall)	97.1%
ROC Area	AUC=$\int_0^1 \mathrm{TPR} * (\mathrm{FPR})\, d\mathrm{FPR} \in [0,1]$	0.983

Once honeypot logs are analyzed and processed by Misuse Detection Engine (MDE) for filtering known attacks, online component of Anomaly Detection Engine (ADE) searches for any abnormalities in the instances declared unidentified by MDE. ADE populates the malicious pool by adding those instances which are declared anomalous by online component based on the predictions of classifier model trained in offline component of ADE. This malicious pool contains possible intrusions. Signature Generation Engine (SGE) extracts attack patterns from this pool and builds signatures. The outcome and results of SGE are described in the following section.

5.5 Signature Generated

SGE makes use of Longest Common Substring (LCS) algorithm based on Generalized Suffix Tree (GST) for extracting string patterns from malicious pool entries. These strings are the candidates for building signatures by SGE. Implementation details of SGE are discussed in Sections 3.10 and 4.8. Suffix tree generated for a string taken from malicious pool given in (5.7) is shown in Snapshot 5.1.

```
/ViewDay.html start=<script>alert("408494469061");</script>$
```
(5.7)



Snapshot 5.1: Suffix Tree generated using Ukkonen algorithm

Once, candidate strings are found by above mentioned algorithm, SGE build the *Snort* compatible signatures. These are used to update *Snort's* rule repository. Some of the attack signatures generated by HIDESIGN are enlisted in Table 5.9.

Table 5.9: Attack signatures generated by HIDESIGN

Sr. No.	Alert Generated	Type of Attack covered
1.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Command Execution ATTACK"; content:" /scripts/..à€¯../..à€¯../..à€¯../winnt/system32/cmd.exe"; nocase; http_uri; sid:1001201;)	Command Injection Attack
2.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:" ilang=eng&SID=2%22%3Cscript%3Ealert(407904364172);%3C/script%3E" ; nocase; http_uri; sid:1001413;)	Cross Site Scripting Attack
3.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" /..0x5c..0x5c..0x5c..0x5c..0x5c..0x5c..0x5c..0x5cetc/passwd "; nocase; http_uri; sid:1000016;)	Directory Traversal ATTACK
4.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:" rtn=1"><script>alert("407894363991");</script><"234"; nocase; http_uri; sid:1001921;)	Cross Site Scripting Attack
5.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20[administrators]%20from%20 "; nocase; http_uri; sid:1000420;)	SQL Injection Attack
6.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20*%20from%20[thdb]..[userinf])%20"; nocase; http_uri; sid:1000107;)	SQL Injection Attack
7.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:" <script>alert("407864363181");</script>"; nocase; http_uri; sid:1000351;)	Cross Site Scripting Attack
8.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:"%20UNION%20ALL%20SELECT%20null,null,(SELECT%20distinct %20TOP%201%20char(126)%2bchar(39)%2bcast(name%20as%20nvarch ar(4000))%2bchar(39)%2bchar(126)%20FROM%20syscolumns%20WHERE %20id=(SELECT%20id%20FROM%20sysobjects%20WHERE%20name=char (68)%2bchar(105)%2bchar(114)%2bchar(101)%2bchar(99)%2bchar(116) %2bchar(111)%2bchar(114)%2bchar(121))%20%20and%20name%20not	SQL Injection Attack

	%20in%20(select%20distinct%20top%208%20name%20from%20syscolumns%20where%20id=(select%20top%201%20id%20from%20sysobjects%20where%20name=char(68)%2bchar(105)%2bchar(114)%2bchar(101)%2bchar(99)%2bchar(116)%2bchar(111)%2bchar(114)%2bchar(121))),null,null,null,null,null,null,null,null,null,null,null"; nocase; http_uri; sid:1000674;)	
9.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:" test=<script>alert(12345)</script>"; nocase; http_uri; sid:1000816;)	Cross Site Scripting Attack
10.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" PGV_BASE_DIRECTORY=../../../../../../../../../../boot.ini "; nocase; http_uri; sid:1000480;)	Directory Traversal ATTACK
11.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20*%20from%20[thdb]..[﷽] "; nocase; http_uri; sid:1000897;)	SQL Injection Attack
12.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" /..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5cwindows/win.ini "; nocase; http_uri; sid:1000976;)	Directory Traversal ATTACK
13.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20len(db_name()))%20between%200%20and%2099999999"; nocase; http_uri; sid:1000854;)	SQL Injection Attack
14.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" /..%2f..%2f..%2f..%2f..%2f..%2f..%2f..%2fetc/passwd"; nocase; http_uri; sid:1000015;)	Directory Traversal ATTACK
15.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" url=../../../../../../../../../../windows/win.ini%00"; nocase; http_uri; sid:1000013;)	Directory Traversal ATTACK
16.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" %20union%20all%20select%20null,null,null "; nocase; http_uri; sid:1000989;)	SQL Injection Attack
17.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20*%20from%20[thdb]..[???])%20"; nocase; http_uri; sid:1000470;)	SQL Injection Attack
18.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Cross Site Scripting Attack"; content:"do_search=Search&searchword=%22%2F%3E%3Cscript%3Ealert%28%22407884363859%22%29%3B%3C%2Fscript%3E%3C%221&catid_search"; nocase; http_uri; sid:1000489;)	Cross Site Scripting Attack

19.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20[û]%20from%20 "; nocase; http_uri; sid:1000056;)	SQL Injection Attack
20.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20[]%20from%20 "; nocase; http_uri; sid:1000098;)	SQL Injection Attack
21.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20[û]%20from%20 "; nocase; http_uri; sid:1000755;)	SQL Injection Attack
22.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Command Execution ATTACK"; content:" /scripts/..À¯../..À¯../..À¯../winnt/system32/cmd.exe"; nocase; http_uri; sid:1000405;)	Command Injection Attack
23.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:"select%20[u_name]%20from%20"; nocase; http_uri; sid:1000341;)	SQL Injection Attack
24.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Command Execution ATTACK"; content:" /samples/..à€¯../..à€¯../..à€¯../winnt/system32/cmd.exe"; nocase; http_uri; sid:1000124;)	Command Injection Attack
25.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" %20and%20exists%20(select%20[admin_user]%20from%20 "; nocase; http_uri; sid:1000945;)	SQL Injection Attack
26.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" Directory Traversal ATTACK"; content:" /..%5c..%5c..%5c..%5c..%5c..%5c..%5c..%5cetc/passwd"; nocase; http_uri; sid:1000761;)	Directory Traversal ATTACK
27.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:" SQL injection attack"; content:" select%20[glmima]%20from%20"; nocase; http_uri; sid:1000410;)	SQL Injection Attack
28.	alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"Command Execution ATTACK"; content:"/winnt/system32/cmd.exe"; nocase; http_uri; sid:1000467;)	Command Injection Attack

The rule file in which these signatures are appended is included in configuration file of *Snort* so that these rules may be referred by NIDS at the time of detection in future. The results presented in the above sections show significant improvement in extending the detection capabilities of IDS.

5.6 Comparison of HIDESIGN with some of the existing systems

There are several existing systems proposed by researchers as reported in Chapter 2. These systems differ from each other on the basis of several aspects and contexts for which signatures have been generated. Existing systems are compared on the basis of various important parameters in Table 2.1 under Section 2.7. HIDESIGN has also been compared with some of the recent systems reported in literature. To accomplish this comparative analysis, the parameters used in Table 2.1, namely, methodology for signature generation, type of signatures generated, attacks/worms covered and false alarms rate have been taken into consideration. The comparison has been summarized in a tabular form given in Table 5.10. It could be observed from the results that HIDESIGN is a system to detect and generate signatures for HTTP and SMTP attacks with low false positives and false negatives.

Table 5.10: Comparison of HIDESIGN with recent existing systems for attack detection and signature generation

Sr. No	Parameter→ System↓	Year of Publication	Methodology for Signature Generation	Type of Signatures generated	Attacks/Worms covered	False Alarms Rate
1	*Hancock*	2009	Uses probability based, disassembly based, and diversity based string signatures with contiguous byte sequence	Single component and Multi component signatures for Anti Viruses	Malware detection	Sufficiently low false positives (Below 0.1%)
2	*Zasmin*	2009	Longest Common Substring	*Snort* signatures	MS06-04010, MS05-039, MS04-01116, MS04-0079, MS04-026 3, MS03-0393. LSASS Vulnerablities related attacks	High
3	*Nebula*	2009	Clustering and Generalized Suffix Tree based Longest Common Substring	*Snort* Compatible	Novel worms such as Conficker	Low false positives
4	*Auto-Sign*	2010	By ranking the candidate signature based on entropy, probability and distance	Signatures compatible with NIDS/NIPS operating as malware filtering devices	Malware executables including worm emails, virus, Trojans, Email flooder, DOS attack, exploits, worms, P2P attacks	Low false positives
5	*F-Sign*	2011	Entropy based selection after creating CFL (Common Functions Library)	Compatible with eDare (early detection, alert, and response) framework	Malware Detection	Low false positives Provided CFL size is large (Below 0.4% with 1675 MB CFL)
6	Hybrid Honeyfarm based approach	2012	Longest Common Subsequence along with protocol based packet header anomaly detection technique	Local signature detection and generation engine	Metasploit generated attack patterns	4% false alarms rate

7	HIDESIGN	2014	Hybrid approach using honeypots, Misuse (Signature based) and Anomaly Detection (Supervised Machine Learning Based), Generalized Suffix Tree based Longest Common Substring	*Snort* compatible signatures	SQL Injection, Cross Site Scripting, Directory Traversal, Command Injection and other HTTP and SMTP related attacks	1.8% false positives and 0.02% false negatives for HTTP attacks, 6.69% false positives and 3.17% false negatives for SMTP attack detection

Chapter Summary

In this chapter the results obtained as outcome of the work carried out in this thesis are discussed. The performance evaluation of HIDESIGN has been presented in this chapter. The evaluation of proposed system has been performed using various evaluation metrics. These metrics have been selected in such a way that the proposed system can be tested from several aspects of good IDS. The results of various metrics like Sensitivity, Specificity, False Positive Rate, Accuracy, Detection Rate, F-Measure and ROC for both HTTP and SMTP attack detection are very encouraging. The proposed system is capable of successfully detecting and generating attack signatures of various HTTP attacks with a sensitivity and specificity of 99.97% and 98.19% for HTTP attack detection training model. The sensitivity and specificity for SMTP attacks is 96.83% and 93.3% respectively. It has reported 1.8% false positives and 0.02% false negatives for HTTP attacks whereas for SMTP attack detection 6.69% false positives and 3.17% false negatives have been reported. The auto generated signatures in *Snort* format have also been presented in this chapter. Further, HIDESIGN has been compared with existing recent systems proposed in literature.

Chapter 6

Conclusions and Future Scope

6.1 Conclusions

In this thesis, in-depth study of existing tools and techniques for intrusion detection and automated signature generation systems along with their comparative analysis has been done in literature review. After an extensive review of literature about current security solutions and various mechanisms, a need for automatic signature generation of HTTP and SMTP attacks has been identified. The work carried out in this thesis is focused on automatic attack signature generation, tracking and analysis. For the accomplishment of proposed objectives a proactive hybrid framework for attack detection and signature generation has been designed, implemented and tested.

A major outcome of this research work is proposed framework named as HIDESIGN (Hybrid Intrusion DEtector and SIGNature generator) that uses both misuse and anomaly based approach for detection of attacks. It is designed to generate signatures so as to update signature repository of Intrusion Detection Systems proactively. The combination of signature and anomaly based detection in conjunction with honeypots makes it an effective defense mechanism that detects new attacks within minimum time of their launch and with minimum damage to information.

Major components of HIDESIGN include honeypot server, Misuse Detection Engine (MDE), Anomaly Detection Engine (ADE) and Signature Generation Engine (SGE). Incoming packet stream from network traffic is captured by honeypot server. Signature based detection is accomplished by MDE to detect and filter out known attacks from this traffic stream. Any deviation from the normal behavior is detected by ADE and a malicious pool is populated. Finally, SGE generates signatures for an alert from malicious pool of data. These signatures are used to update signatures repository of NIDS and may further be used to safeguard the network from intrusions.

This proposed system has been implemented using various modules developed in *Java*. It makes use of honeypot technology along with *Snort,* a signature based NIDS and anomaly detection based on supervised machine learning using Naive Bayes classifier.

ADE is trained using two datasets, namely, HTTP CSIC 2010 and a subset of NSL-KDD dataset for training of HTTP and SMTP detection models. The evaluation is performed using standard IDS evaluation metrics like detection rate, TP Rate, FP Rate, Precision, Recall, F-Measure and ROC Area. The proposed system is capable of successfully detecting and generating attack signatures of various HTTP attacks with a sensitivity and specificity of 99.97% and 98.19% for HTTP attack detection training model. The sensitivity and specificity for SMTP attacks is 96.83% and 93.3% respectively. It has reported 1.8% false positives and 0.02% false negatives for HTTP attacks whereas for SMTP attack detection 6.69% false positives and 3.17% false negatives have been reported.

6.2 Future Scope

The results of proposed system demonstrate successful implementation of HIDESIGN. There are certain factors on which system can further be improved. In future, this work can be enhanced by incorporating following research directions into the proposed system.

- Offline component of Anomaly Detection Engine (ADE) can be trained on a more recent, updated and larger dataset for detection of attacks and exploits targeting current vulnerabilities.
- For the generation of signatures a high performance pattern extraction algorithm may be adopted. A parallel processing approach using GPUs (Graphical Processing Units) can be used for its implementation.
- A high interaction honeypot may used to capture some complex attacks.
- In future, a system based on this framework can be implemented to handle attacks in real time so that this solution can be used as a plug-in to *Snort* for automatic updating of signatures repository.
- This framework may further be improved to reduce the false alarm rate.

- This system can be extended to generate signatures compatible with other Intrusion Detection Systems, like *Bro*.

To conclude, improvement always lead to betterment and a lot more can be contributed in this area of network security research.

Change is vital, improvement the logical form of change.

James Cash Penney

References

[Abd14] Abdulla, S., Ramadass, S., Altyeb, A. A., and Al-Nassiri, A. (2014). Employing Machine Learning Algorithms to Detect Unknown Scanning and Email Worms. International Arab Journal of Information Technology, 11(2), 140-148.

[Aho75] Aho, A. V., and Corasick, M. J. (1975). Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM, 18(6), 333-340.

[All07] Allot Communications, (2007). Digging Deeper Into Deep Packet Inspection (DPI), White Paper. Available Online at: http://www.datakom.de/fileadmin/docs/Allot/Whitpaper/DPI.pdf

[Amo04] Amor, N. B., Benferhat, S., and Elouedi, Z. (2004). Naive Bayes vs Decision Trees in Intrusion Detection Systems. In ACM symposium on Applied Computing, 420-424.

[Amz11] Amza, C., Leordeanu, C., and Cristea, V. (2011). Hybrid Network Intrusion Detection. In IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), 503-510.

[And72a] Anderson, J. P. (1972). Computer Security Technology Planning Study, Vol. I. Technical Report ESC–TR–73–51, ESD/AFSC, Hanscom AFB, Bedford.

[And72b] Anderson, J. P. (1972). Computer Security Technology Planning Study, Vol. II. Technical Report ESC–TR–73–51, ESD/AFSC, Hanscom AFB, Bedford.

[And80] Anderson, J. P. (1980). Computer Security Threat Monitoring and Surveillance. Technical Report, James P Anderson Co, Fort Washington.

[And95] Anderson, D., Frivold, T., and Valdes, A. (1995). Next-Generation Intrusion Detection Expert System (NIDES): A Summary. SRI International, Computer Science Laboratory.

[Ann05] Annual Report 2004-2005 (2005). Centre for Development of Advanced Computing. Available Online at: http://cdac.in/index.aspx?id=pdf_Annual_Report_04_05

[Ayd09] Aydın, M. A., Zaim, A. H., and Ceylan, K. G. (2009). A Hybrid Intrusion Detection System Design for Computer Network Security. Computers & Electrical Engineering, 35(3), 517-526.

[Bae06] Baecher, P., Koetter, M., Holz, T., Dornseif, M. and Freiling, F. (2006). The Nepenthes Platform: An Efficient Approach to Collect Malware. In Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 165-184.

[Bal05] Balas, E., and Viecco, C. (2005). Towards A Third Generation Data Capture Architecture for Honeynets. In 6th Annual IEEE Information Assurance Workshop (IAW'05), 21-28.

[Bal98] Balasubramaniyan, J. S., Garcia-Fernandez, J. O., Isacoff, D., Spafford, E., and Zamboni, D. (1998). An Architecture for Intrusion Detection Using Autonomous Agents. In 14th Annual IEEE Computer Security Applications Conference, 13-24.

[Bar01] Barbara, D., Couto, D. J., Jajodia, S., and Wu, N. (2001). ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection. ACM SIGMOD Record, 30(4), 15-24.

[Bar02] Barbara, D., Jajodia, S. (2002). Applications of Data Mining in Computer Security. Kluwer Academic Publishers.

[Ben07] Ben Gal, I. (2007). Bayesian Networks. In Encyclopedia of Statistics in Quality and Reliability, Ruggeri, F., Kenett, R. S. and Faltin, F. (Editors-in-Chief), Wiley, UK.

[Bhu14] Bhuyan, M. H., Bhattacharyya, D. K., and Kalita, J. K. (2014). Network Anomaly Detection: Methods, Systems and Tools. IEEE Communications Surveys & Tutorials, 16(1), 303-336.

[Bru08] Brumley, D., Newsome, J., Song, D., Wang, H., and Jha, S. (2008). Theory and Techniques for Automatic Generation of Vulnerability-Based Signatures. IEEE Transactions on Dependable and Secure Computing, 5(4), 224-241.

[Cav94] Cavnar, W. B., and Trenkle, J. M. (1994). N-gram-based text categorization. ANN Arbor MI, 48113(2), 161-175.

[Cha04] Chamales, G. (2004). The Honeywall CD-ROM. IEEE Security & Privacy, 2(2), 77-79.

[Cha12] Chaudhary, A., Misra, M., and Sardana, A. (2012). An Efficient Fuzzy Controller Based Technique for Network Traffic Classification to Improve QoS. In 5th International Conference on Security of Information and Networks, ACM, 95-102.

[Che05] Chebrolu, S., Abraham, A., and Thomas, J. P. (2005). Feature Deduction and Ensemble Design of Intrusion Detection Systems. Computers & Security, 24(4), 295–307.

[Chh09] Chhabra, A., and Singh, G. (2009). Knowledge-Based Modeling Approach for Performance Measurement of Parallel Systems. International Arab Journal of Information Technology, 6(1), 77-84.

[Cis99] Cisco Systems, Inc., (1999). NetRanger User's Guide. Available Online at: http://www.cisco.com/application/pdf/en/us/guest/products/ps2113/c1626/ccmigration_09186a00800ee98e.pdf

[Coh98] Cohen, F. (1998). A Note on The Role of Deception in Information Protection, Computers & Security, 17(6), 483-506.

[Cor06] Coret, J. A., (2006). Kojoney-A honeypot for the SSH Service. Available Online at: http://kojoney.sourceforge.net/

[Cos05] Costa, M., Crowcroft, J., Castro, M., Rowstron, A., Zhou L., Zhang L. and Barham P. (2005). Vigilante: End-to-end containment of Internet worms. In 20th ACM Symposium on Operating Systems Principles (SOSP '05), New York, USA, 133-147.

[Cui07] Cui, W., Peinado, M., Wang, H. J., and Locasto, M. (2007). ShieldGen: Automated Data Patch Generation for Unknown Vulnerabilities with Informed Probing. In IEEE Symposium on Security and Privacy, Berkley, CA, 252-266.

[Dag04] Dagon, D., Qin, X., Gu, G., Lee, W., Grizzard, J., Levine, J., and Owen, H. (2004). HoneyStat: Local Worm Detection Using Honeypots. In Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 39-58.

[Dan10] Danev, B., Luecken, H., Capkun, S., and El Defrawy, K. (2010). Attacks on Physical-Layer Identification. In 3rd ACM Conference on Wireless Network Security, ACM, 89-98.

[Den87] Denning, D. E. (1987). An Intrusion-Detection Model. IEEE Transactions on Software Engineering, (2), 222-232.

[Den85] Denning, D. E., and Neumann, P. G. (1985). Requirements and model for IDES - A Real-Time Intrusion Detection Expert System. Document A005, SRI International.

[Dha06] Dharmapurikar, S., and Lockwood, J. W. (2006). Fast and Scalable Pattern Matching for Network Intrusion Detection Systems. IEEE Journal on Selected Areas in Communications, 24(10), 1781-1792.

[Dom97] Domingos, P., and Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss. Machine Learning, 29, 103-130.

[Ert04] Ertoz, L., Eilertson, E., Lazarevic, A., Tan, P. N., Kumar, V., Srivastava, J., and Dokas, P. (2004). Minds- Minnesota Intrusion Detection System. Next Generation Data Mining, 199-218.

[Est04] Estévez-Tapiador, J. M., Garcıa-Teodoro, P., and Dıaz-Verdejo, J. E. (2004). Measuring Normality in HTTP Traffic for Anomaly-Based Intrusion Detection. Computer Networks, 45(2), 175-193.

[Far10] Farid, D. M., Harbi, N., and Rahman, M. Z. (2010). Combining Naive Bayes and Decision Tree for Adaptive Intrusion Detection. International Journal of Network Security & Its Applications, 2(2), 12-25.

[Fie09] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (2009). RFC 2616, Hyper Text Transfer Protocol–http/1.1, 1999. Available Online at: http://www. rfc. net/rfc2616.html

[Fri97] Friedman, N., Geiger, D., and Goldszmidt, M. (1997). Bayesian Network Classifiers. Machine Learning 29 (2-3), 131–163.

[Gal08] Gallagher, B., and Eliassi-Rad, T. (2008). Classification of HTTP Attacks: A Study on the ECML/PKDD 2007 Discovery Challenge. In Center for Advanced Signal and Image Sciences (CASIS) Workshop, 1-8.

[Gar09] Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., and Vázquez, E. (2009). Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges. Computers & Security, 28(1), 18-28.

[Gho09] Ghorbani, A. A., Lu, W., and Tavallaee, M. (2009). Network Intrusion Detection and Prevention: Concepts and Techniques. Springer, Vol. 47.

[Gle96] Gleichauf, B., and Teal, D. (1996). NetRanger High-level Overview Version 1.1.WheelGroup Corp.

[Gom09] Gómez, J., Gil, C., Padilla, N., Baños, R., and Jiménez, C. (2009). Design of a Snort-Based Hybrid Intrusion Detection System. In Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living, Springer Berlin Heidelberg, 515-522.

[Gré07] Grégio, A., Santos, R., and Montes, A. (2007). Evaluation of Data Mining Techniques for Suspicious Network Activity Classification Using Honeypots Data. In Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security (SPIE 6570), 657006.

[Gri09] Griffin, K., Schneider, S., Hu, X. and Chiueh, T. (2009). Automatic Generation of String Signatures for Malware Detection. In 12th International Symposium on Recent Advances in Intrusion Detection, Berlin, Heidelberg. Springer Press, Springer-Verlag, 101–120.

[Gut95] Guttman, B., and Roback, E. A. (1995). An Introduction to Computer Security: The NIST Handbook, DIANE Publishing.

[Hal09] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. ACM SIGKDD Explorations Newsletter, 11(1), 10-18.

[Her90] Heberlein, L. T., Dias, G. V., Levitt, K. N., Mukherjee, B., Wood, J., and Wolber, D. (1990). A Network Security Monitor. In IEEE Computer Society Symposium on Research in Security and Privacy, IEEE, 296-304.

[Hoc93] Hochberg, J., Jackson, K., Stallings, C., McClary, J. F., DuBois, D., and Ford, J. (1993). NADIR: An Automated System for Detecting Network Intrusion and Misuse. Computers & Security, 12(3), 235-248.

[HTT09] HTTP Basics: Online Tutorial (2009). Available Online at: http://www.ntu.edu.sg/home/ehchua/programming/webprogramming/HTTP_Basics.html.

[Hus12] Hussein, S. M., Ali, F. H. M., and Kasiran, Z. (2012). Evaluation Effectiveness of Hybrid IDS using Snort with Naive Bayes to Detect Attacks. In 2nd International Conference on Digital Information and Communication Technology and it's Applications (DICTAP), 256-260.

[Hwa07] Hwang, K., Cai, M., Chen, Y., and Qin, M. (2007). Hybrid Intrusion Detection with Weighted Signature Generation over Anomalous Internet Episodes. IEEE Transactions on Dependable and Secure Computing, 4(1), 41-55.

[IBM13] IBM Corporation (2013). IBM Internet Security Systems X-Force 2012 Mid-year trend statistics. Available Online at: http://www.ibm.com/ibm/files/I218646H25649F77/Risk_Report.pdf

[Ilg93] Ilgun, K. (1993). USTAT: A Real-Time Intrusion Detection System for UNIX. In IEEE Computer Society Symposium on Research in Security and Privacy, 16-28.

[Int04] IntruPro IPS, Inline Intrusion Prevention, White Paper, 2004. Available Online at: http://usacac.army.mil/cac2/cew/repository/papers/IntruPro_IPS.pdf

[Jac09] Jacobson, V., and McCanne, S. (2009). libpcap: Packet capture library. Lawrence Berkeley Laboratory, Berkeley, CA.

[Jai12] Jain P. and Sardana A. (2012). Defending against Internet Worms using Honeyfarm. In CUBE International Information Technology Conference (CUBE 2012), Pune, India, 795-800.

[Jol05] Jolliffe, I. (2005). Principal Component Analysis. John Wiley & Sons, Ltd.

[Jon00] Jones, A. K., and Sielken, R. S. (2000). Computer System Intrusion Detection: A Survey. Computer Science Technical Report, 1-25.

[Jos13] Joshi, A., Lal, R., Finin, T., and Joshi, A. (2013). Extracting Cybersecurity Related Linked Data From Text. In IEEE 7th International Conference on Semantic Computing, IEEE, 252-259.

[Kab05] Kabiri, P., and Ghorbani, A. A. (2005). Research on Intrusion Detection and Response: A Survey. International Journal of Network Security, 1(2), 84-102.

[Kar02] Karkera, A. N. (2002). Design and Implementation of A Policy-Based Intrusion Detection System–Generic Intrusion Detection Model for A Distributed Network. Doctoral Dissertation, University of Florida.

[Kar13] Karame, G. O., Danev, B., Bannwart, C., and Capkun, S. (2013). On the Security of end-to-end Measurements Based on Packet-Pair Dispersions. IEEE Transactions on Information Forensics and Security, 8(1), 149-162.

[Kar87] Karp, R. M., and Rabin, M. O. (1987). Efficient Randomized Pattern-Matching Algorithms. IBM Journal of Research and Development, 31(2), 249-260.

[Kau02] Kaufman, C., Perlman, R., & Speciner, M. (2002). Network Security: Private Communication in a Public World. Prentice Hall Press.

[Kim04] Kim, H.A., and Karp, B. (2004). Autograph: Toward Automated, Distributed Worm Signature Detection. 13th Usenix Security Symposium (Security 2004), San Diego, CA, 271-286.

[Kim09] Kim, I., Kim, D., Choi, Y., Kang, K., Oh, J., and Jang, J. (2009). Validation Methods of Suspicious Network Flows for Unknown Attack Detection. International Journal of Computers, 3(1), 104-114.

[Kle08] Klensin, J. (2008). Simple Mail Transfer Protocol, RFC5321. Available Online at: http://tools.ietf.org/html/rfc5321?u=878041af

[Koh09] Kohlrausch, J. (2009). Experiences with the NoAH Honeynet Testbed to Detect new Internet Worms. In 5th International Conference on IT Security Incident Management and IT Forensics, Stuttgart, Germany, 13-26.

[Kre04] Kreibich, C., and Crowcroft, J. (2004). Honeycomb: Creating Intrusion Detection Signatures Using Honeypots. ACM SIGCOMM Computer Communication Review, 34(1), 51–56.

[Law90] Lawrence, C. E., and Reilly, A. A. (1990). An Expectation Maximization (EM) Algorithm for the Identification and Characterization of Common Sites in Unaligned Biopolymer Sequences. Proteins: Structure, Function, and Bioinformatics, 7(1), 41-51.

[Law93] Lawrence, C. E., Altschul, S. F., Boguski, M. S., Liu, J. S., Neuwald, A. F., and Wootton, J. C. (1993). Detecting Subtle Sequence Signals: A Gibbs Sampling Strategy for Multiple Alignment. Science, 262(5131), 208-214.

[Laz05] Lazarevic, A., Kumar, V., and Srivastava, J. (2005). Intrusion Detection: A Survey. In Managing Cyber Threats, Springer US, 19-78.

[Lee07] Lee, S. M., Kim, D. S., and Park, J. S. (2007). A Hybrid Approach for Real-Time Network Intrusion Detection Systems. In International Conference on Computational Intelligence and Security, IEEE, 712-715.

[Lee98] Lee, W., Stolfo, S. J., and Mok, K. W. (1998). Mining Audit Data to Build Intrusion Detection Models. In KDD, 66-72.

[Lee99] Lee, W., Stolfo, S. J., and Mok, K. W. (1999). A Data Mining Framework for Building Intrusion Detection Models. In IEEE Symposium on Security and Privacy, IEEE, 120-132.

[Li06] Li, Z., Sanghi, M., Chen, Y., Kao, M. and Chavez, B. (2006). Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In IEEE Symposium on Security and Privacy (S&P'06), IEEE Computer Society, Washington, DC, USA, 32–47.

[Li07] Li, Z., Wang, L., Chen, Y., and Fu, Z. (2007). Network-based and Attack-resilient Length Signature Generation for Zero-day Polymorphic Worms. In IEEE International Conference on Network Protocols (ICNP 2007), 164-173.

[Lia05a] Liang, Z., and Sekar, R. (2005). Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers. In 12th ACM Conference on Computer and Communications Security, 213-222.

[Lia05b] Liang. Z. and Sekar R. (2005). Automatic Generation of Buffer Overflow Attack Signatures: An Approach Based on Program Behavior Models. In 21st Annual Computer Security Applications Conference, Tucson, Arizona, USA, 1-10.

[Lin13] Lin, C. H., Liu, C. H., Chien, L. S., and Chang, S. C. (2013). Accelerating Pattern Matching Using a Novel Parallel Algorithm on GPUs. IEEE Transactions on Computers, 62(10), 1906-1916.

[Lip00] Lippmann, R. P., Fried, D. J., Graf, I., Haines, J. W., Kendall, K. R., McClung, D., ... and Zissman, M. A. (2000). Evaluating Intrusion Detection Systems: The 1998 DARPA off-line intrusion detection evaluation. In DARPA Information Survivability Conference and Exposition (DISCEX'00), IEEE, 12-26.

[Luo06] Luo, H., Fang, B., and Yun, X. (2006). Anomaly Detection in SMTP Traffic. In 3rd International Conference on Information Technology: New Generations (ITNG 2006), IEEE, 408-413.

[Mah01] Mahoney, M. V., and Chan, P. K. (2001). PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic. Technical Report, Department of Computer Sciences, Florida Institute of Technology, Melbourne, FL, USA.

[Mah02] Mahoney, M. V., and Chan, P. K. (2002). Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. In 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 376-385.

[Mah03] Mahoney, M. V. (2003). Network Traffic Anomaly Detection Based On Packet Bytes. In ACM Symposium on Applied computing, ACM, 346-350.

[McD03] McDaniel, M., and Heydari, M. H. (2003). Content Based File Type Detection Algorithms. In 36th Annual Hawaii International Conference on System Sciences, IEEE, 1-10.

[McG07] McGeehan, R., and Engert, R. (2007). GHH-The "Google Hack" Honeypot.

[Mil04] Miliefsky, G. (2004). A Guide to Proactive Network Security. ZDNet News.

[Moh03] Mohajerani, M., Moeini, A., and Kianie, M. (2003). NFIDS: A Neuro-Fuzzy Intrusion Detection System. In 10th IEEE International Conference on Electronics, Circuits and Systems (ICECS 2003), 348-351.

[Moh08] Mohammed, M. M. Z. E., Chan, H. A. and Ventura, N. (2008). Honeycyber: Automated Signature Generation for Zero-Day Polymorphic Worms. In IEEE Military Communications Conference (MILCOM), San Diego, CA, 1-6.

[Moh10] Mohammed, M. M. Z. E., Chan, H. A., Ventura, N., Hashim, M., Amin, I., and Bashier, E. (2010). Detection of Zero-day Polymorphic Worms using Principal Component Analysis. In 6th International Conference on Networking and Services, 277-281.

[Mud11] Muda, Z., Yassin, W., Sulaiman, M. N., and Udzir, N. I. (2011). Intrusion Detection Based on *k*-means Clustering and Naive Bayes Classification. In 7th International Conference on Information Technology in Asia (CITA 11), 1-6.

[Muk94] Mukherjee, B., Heberlein, L. T., and Levitt, K. N. (1994). Network Intrusion Detection. IEEE Network, 8(3), 26-41.

[Nan08] Nan, L., Chunhe, X., Yi, Y., and Haiquan, W. (2008). An Algorithm for Generation of Attack Signatures Based on Sequences Alignment. In International Conference on Computer Science and Software Engineering, IEEE , 964-969.

[Nee70] Needleman, S. B., and Wunsch, C. D. (1970). A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. Journal of Molecular Biology, 48(3), 443-453.

[Net03] Nethercote, N., and Seward, J. (2003). Valgrind: A Program Supervision Framework. Electronic Notes in Theoretical Computer Science, 89(2), 44-66.

[New05a] Newsome, J. and Song, D. (2005). Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In 12th Annual Network and Distributed System Security Symposium (NDSS), San Diego, California, 1-17.

[New05b] Newsome, J., Karp, B. and Song, D. (2005). Polygraph: Automatically Generating Signatures for Polymorphic Worm. In IEEE Symposium on Security and Privacy, IEEE Press, Oakland, CA, USA, 226–241.

[Ngu11] Nguyen, H. T., Torrano-Gimenez, C., Alvarez, G., Petrovic, S., and Franke, K. (2011). Application of the Generic Feature Selection Measure in Detection of Web Attacks. In 4th International Conference, Computational Intelligence in Security for Information Systems (CISIS 11), 25–32.

[Nin03] Ning, P., and Jajodia, S. (2003). Intrusion Detection Techniques. The Internet Encyclopedia.

[Olg09] Olguín, O., and Medina, M. (2009). GIDRE: Grid-based Detection Intrusion and Response Environment. In ISSE Securing Electronic Business Processes, 172-180.

[Ope13] Open Web Application Security Project (OWASP) Top 10 (2013), Available Online at:
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

[Ore09] Oreku, G. S., and Mtenzi, F. J. (2009). Intrusion Detection Based on Data Mining. In 8th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC'09), IEEE, 696-701.

[OWA14] OWASP Testing Guide V4 (2014). Testing for IMAP/SMTP Injection, Available Online at: https://www.owasp.org/index.php/Testing_for_IMAP/SMTP_Injection_(OTG-INPVAL-011)

[Pal11] Palmer, J. (2011). Naive Bayes Classification for Intrusion Detection Using Live Packet Capture. Data Mining in Bioinformatics (Report), Available Online at: http://www.cse.usf.edu/~xqian/courses/CIS6930_prml/reports/report_palmer.pdf

[Pan07] Panda, M. and Patra, M. R. (2007). Network intrusion detecction using Naive Bayes. International Journal of Computer Science and Network Security, 7(12), 258-263.

[Pan09] Panda, M., and Patra, M. R. (2009). Semi-Naive Bayesian Method for Network Intrusion Detection System. In Neural Information Processing, Springer Berlin Heidelberg, 614-621.

[Pax99] Paxson, V. (1999). Bro: A System for Detecting Network Intruders in Real-Time. Computer Networks, 31(23-24), 2435-2463.

[Pfl02] Pfleeger, C. P., and Pfleeger, S. L. (2002). Security in Computing. Prentice Hall Professional Technical Reference.

[Por06] Portokalidis, G., Slowinska, A. and Bos, H. (2006). Argos: An Emulator for Fingerprinting Zero-Day Attack. In International Conference of ACM SIGOPS EUROSYS, Leuven, Belgium, 15-28.

[Por07] Portokalidis, G. and Bos, H. (2007). SweetBait: Zero-Hour Worm Detection and Containment Using Honeypots. Computer Networks, Special Issue on Security through Self-Protecting and Self-Healing Systems, 51(5), 1256–1274.

[Por08] Portokalidis, G. and Bos, H. (2008). Eudaemon: Involuntary and On-Demand Emulation Against Zero-Day Exploit. In 3rd International Conference on ACM SIGOPS/EuroSys European Conference on Computer Systems, New York, USA, 287-299.

[Por97] Porras, P. A., and Neumann, P. G. (1997). EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances. In 20th National Information Systems Security Conference, 353-365.

[Pra13] Prathapani, A., Santhanam, L. Agrawal, D. P. (2013). Detection of Blackhole Attack in a Wireless Mesh Network using Intelligent Honeypot Agents. The Journal of Supercomputing, 64(3), 777-804.

[Pro03a] Project, T. H. (2003). "Know your enemy: Sebek.". Available Online at: http://old.honeynet.org/papers/sebek.pdf

[Pro03b] Provos, N. (2003). Honeyd - A Virtual Honeypot Daemon. In 10th DFN-CERT Workshop, Hamburg, Germany.

[Pro07] Project, T. H., High Interaction Honeypot Analysis Toolkit (HIHAT). Available Online at: http://hihat.sourceforge.net/

[Qaz12] Qazanfari, K., Mirpouryan, M. S., and Gharaee, H. (2012). A Novel Hybrid Anomaly Based Intrusion Detection Method. In 6th International Symposium on Telecommunications (IST), IEEE, 942-947.

[Rab03] Rabek, J. C., Khazan, R. I., Lewandowski, S. M., and Cunningham, R. K. (2003). Detection of Injected, Dynamically Generated, and Obfuscated Malicious Code. In ACM workshop on Rapid malcode, ACM, 76-82.

[Rab81] Rabin, M. O. (1981). Fingerprinting by Random Polynomials. Center for Research in Computing Technology, Aiken Computation Laboratory, Univ., 15-18.

[Ran98] Ranum, M. J., Landfield, K., Stolarchuk, M., Sienkiewicz, M., Lambeth, A., and Wall, E. (1998). Implementing a Generalized Tool for Network Monitoring. Information Security Technical Report, 3(4), 53-64.

[Rea97] RealSecure Release 1.2 for UNIX (1997). A User Guide and Reference Manual, Internet Security Systems.

[Reh03] Rehman, R. U. (2003). Intrusion Detection Systems with Snort: Advanced IDS Techniques Using Snort, Apache, MySQL, PHP, and ACID. Prentice Hall Professional.

[Res06] Research Infrastructures Action, (2006) D1.2: Attack Detection and Signature Generation Technical Report under NoAH Project, Available Online at: http:// www.fp6-noah.org/publications/deliverables/D1.2.pdf

[Roe99] Roesch, M. (1999). Snort – Lightweight Intrusion Detection for Networks. In 13th Conference on Systems Administration, Seattle, WA, 229–238.

[Rub04] Rubin, S., Jha, S., and Miller, B. P. (2004). Automatic Generation and Analysis of NIDS Attacks. In 20th Annual Computer Security Applications Conference, IEEE, 28-38.

[Sac10] Sachdeva, M., Singh, G., Kumar, K., and Singh, K. (2010). DDoS Incidents and their Impact: A Review. International Arab Journal of Information Technology, 7(1), 14-20.

[Sco04] Scott, S.L. (2004). A Bayesian Paradigm for Designing Intrusion Detection Systems, Computational Statistics & Data Analysis, 45, 69–83.

[Seb88] Sebring, M. M., Shellhouse, E., Hanna, M., and Whitehurst, R. (1988). Expert Systems in Intrusion Detection: A Case Study. In 11th National Computer Security Conference, 74-81.

[Sei06] Seifert, C., and Steenson, R. (2006). Capture-Honeypot Client (Capture-HPC) Available Online at: https://projects. honeynet. org/capture–hpc

[Sen06] Sen, S. (2006). Performance Characterization & Improvement of Snort as An IDS. Bell Labs Report.

[Sha11] Shabtai, A., Menahem, E., and Elovici, Y. (2011). F-Sign: Automatic, Function-Based Signature Generation for Malware. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews, 41(4), 494-508.

[Sin03] Singh, S., Estan, C., Varghese, G., and Savage, S. (2003). Earlybird System for Real-Time Detection of Unknown Worms. Department of Computer Science and Engineering, University of California, San Diego.

[Sin04] Singh, S., Eitan, C., Varghese, G. and Savage, S. (2004). Automated Worm Fingerprinting. In 6th conference on Symposium on Opearting Systems Design & Implementation (OSDI), USENIX Association, Berkeley, CA, USA, 45–60.

[Sma88] Smaha, S. E. (1988). Haystack: An Intrusion Detection System. In 4th IEEE Aerospace Computer Security Applications Conference, 37-44.

[Sma94] Smaha, S. E., and Winslow, J. (1994). Misuse Detection Tools. Computer Security Journal, 10, 39-49.

[Smi05] Smirnov, A., and Chiueh, T. C. (2005). DIRA: Automatic Detection, Identification and Repair of Control-Hijacking Attacks. In Network and Distributed System Security Symposium (NDSS).

[Smi81] Smith, T. F., and Waterman, M. S. (1981). Identification of Common Molecular Subsequences. Journal of Molecular Biology, 147(1), 195-197.

[Sna91a] Snapp, S., Brentano, J., Dias, G. V., Goan, T. L., Grance, T., Heberlein, L. T., ... and Smaha, S. E. (1991). A System for Distributed Intrusion Detection. COMPCOM Spring, 91, 170-176.

[Sna91b] Snapp, S. R., Brentano, J., Dias, G. V., Goan, T. L., Heberlein, L. T., Ho, C. L., ... and Mansur, D. (1991). DIDS (Distributed Intrusion Detection System)-Motivation, Architecture, and an Early Prototype. In 14th National Computer Security Conference, 167-176.

[Sno10] Snort Team, Snort Users Manual 2.8.6 (2010). Available Online at: http://www.snort.org/assets/166/snort_manual.pdf

[Spi03] Spitzner, L. (2003). Honeypots: Tracking Hackers (Vol. 1), Reading: Addison-Wesley.

[Sta02] Staniford, S., Paxson, V., and Weaver, N. (2002). How to Own the Internet in Your Spare Time. In 11th USENIX Security Symposium, San Francisco, CA, 149-167.

[Sta96] Staniford-Chen, S., Cheung, S., Crawford, R., Dilger, M., Frank, J., Hoagland, J., ... and Zerkle, D. (1996). GrIDS-A Graph Based Intrusion Detection System for Large Networks. In 19th National Information Systems Security Conference, 361-370.

[Sti11] Still, M., and McCreath, E. C. (2011). DDoS Protections for SMTP Servers. International Journal of Computer Science and Security (IJCSS), 4(6), 537.

[Tah10] Tahan, G., Glezer, C., Elovici, Y. and Rokach, L. (2010). Auto-Sign: An Automatic Signature Generator for High-Speed Malware Filtering Devices. Journal in Computer Virology, 6(2), 91–103.

[Tan05] Tang, Y. and Chen, S. (2005). Defending Against Internet Worms: A Signature Based Approach. In IEEE INFOCOM'2005, IEEE Press, Miami, 1384–1394.

[Tan06] Tang, Y., Lu, X. C., Hu, H. P., and Zhu, P. D. (2006). Automatic Generation of Attack Signatures Based on Multi-Sequence Alignment. Chinese Journal of Computers-Chinese Edition, 29(9), 1531-1539.

[Tan11] Tang, Y., Xiao, B., and Lu, X. (2011). Signature Tree Generation for Polymorphic Worms. IEEE Transactions on Computers, 60(4), 565-579.

[Tav09] Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A Detailed Analysis of the KDD CUP 99 Data Set. In 2nd IEEE Symposium on Computational Intelligence for Security and Defence Applications.

[Tha05] Thakar, U., Varma, S., and Ramani, A. K. (2005). HoneyAnalyzer–Analysis and Extraction of Intrusion Detection Patterns & Signatures Using Honeypot. In 2nd International Conference on Innovations in Information Technology, 1-7.

[Tho08] Thomas, C., Sharma, V., and Balakrishnan, N. (2008). Usefulness of DARPA Dataset for Intrusion Detection System Evaluation. In Data Mining, Intrusion Detection, Information Assurance, and Data Networks Security (SPIE 6973), 1-8.

[Thu10] Thuraisingham, B., Khan, L., Awad, M., and Wang, L. (2010). Design and Implementation of Data Mining Tools. CRC Press.

[Tor07] Toro, D. (2007). A Snort-Based Approach for the Development and Deployment of Hybrid IDS. IEEE Latin America Transactions, 5(6), 386-392.

[Tsa09] Tsai, C. F., Hsu, Y. F., Lin, C. Y., and Lin, W. Y. (2009). Intrusion Detection by Machine Learning: A Review. Expert Systems with Applications, 36(10), 11994-12000.

[Ukk95] Ukkonen, E. (1995). On-line Construction of Suffix Trees. Algorithmica, 14(3), 249-260.

[Usm13] Usmani, K., Mohapatra, A. K. and Prakash, N. (2013). An Improved Framework for Incident Handling. Information Security Journal: A Global Perspective, 22(1), 1-9.

[Vac89] Vaccaro, H. S., and Liepins, G. E. (1989). Detection of Anomalous Computer Session Activity. In IEEE Symposium on Security and Privacy, 280-289.

[Vig98] Vigna, G., and Kemmerer, R. A. (1998). NetSTAT: A Network-Based Intrusion Detection Approach. In 14th Annual Computer Security Applications Conference, IEEE, 25-34.

[Wan04] Wang, K. and Stolfo, S. J. (2004). Anomalous Payload-based Network Intrusion Detection. In Recent Advances in Intrusion Detection, Springer Press, Sophia Antipolis, French Riviera, France, 203–222.

[Wan05] Wang, K., Cretu, G., and Stolfo, S. J. (2005). Anomalous Payload-based Worm Detection and Signature Generation. In Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 227-246.

[Wan06] Wang, K., Parekh, J. J., and Stolfo, S. J., (2006). Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 226-248.

[Wan12] Wang, Y., Xiang, Y., Zhou, W., Yu, S. (2012). Generating Regular Expression Signatures for Network Traffic Classification in Trusted Network Management. Journal of Network and Computer Applications, 35(3), 992–1000.

[War05] Waraich R., (2005). Automated Attack Signature Generation: A Survey, Technical Report, Swiss Federal Institute of Technology, Zurich, Computer Engineering and Networks Laboratory.

[War70] Ware, W. (1970). Security Controls for Computer Systems (U): Report of Defense Science Board Task Force on Computer Security. Rand Report R609–1, The RAND Corporation, Santa Monica. Available Online at: http://seclab.cs.ucdavis.edu/ projects/history/CD/ware70.pdf

[Wer06] Werner, T. Honeytrap (2006). Available Online at: http://sourceforge.net/projects/honeytrap.

[Wer09] Werner, T., Fuchs, C., Gerhards-Padilla, E., and Martini, P. (2009). Nebula-Generating Syntactical Network Intrusion Signatures. In 4th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, 31-38.

[Wes03] West-Brown, M. J., Stikvoort, D., Kossakowski, K. P., Killcrece, G., and Ruefle, R. (2003). Handbook for Computer Security Incident Response Teams (CSIRTS).

[Wil06] William, S., and Stallings, W. (2006). Cryptography and Network Security, 4/E, Pearson Education India.

[Xia08] Xiang, C., Yong, P. C., and Meng, L. S. (2008). Design of Multiple-Level Hybrid Classifier for Intrusion Detection System using Bayesian Clustering and Decision Trees. Pattern Recognition Letters, 29, 918–924.

[Yan10] Yang, J., Chen, X., Xiang, X. and Wan., J. (2010). HIDS-DT: An Effective Hybrid Intrusion Detection System Based on Decision Tree. In International Conference on Communications and Mobile Computing, Shenzhen, 70-75.

[Yeg05] Yegneswaran, V., Giffin, J. T., Barford, P. and Jha, S. (2005). An Architecture for Generating Semantics-Aware Signatures. In 14th USENIX Security Symposium, USENIX Association, Baltimore, MD, USA, 97–112.

[Yer14] Yerima, S.Y., Sezer, S., and McWilliams, G. (2014). Analysis of Bayesian Classification-Based Approaches for Android Malware Detection. IET Information Security, 8(1), 25–36.

[Zha08] Zhang, J., Zulkernine, M., and Haque, A. (2008). Random-Forests-Based Network Intrusion Detection Systems. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, 38(5), 649-659.

[Zha10] Zhao, D., Xu, Q., and Feng, Z. (2010). Analysis and Design for Intrusion Detection System Based on Data Mining. In 2nd International Workshop on Education Technology and Computer Science (ETCS), IEEE, 339-342.

[Zou04] Zou, C. C., Towsley, D., and Gong, W. (2004). Email Worm Modeling and Defense. In 13th International Conference on Computer Communications and Networks (ICCCN 2004), IEEE, 409-414.

List of Publications

1. Sanmeet Kaur and Maninder Singh, Automatic Attack Signature Generation Systems: A Review, IEEE Security and Privacy, May 2013, Volume 11, Issue 6, pp 54-61. [SCI Indexed]

2. Sanmeet Kaur and Maninder Singh, Policy Scripts to Detect Network Intrusions, International Journal of Scientific & Engineering Research, August 2013, Volume 4, Issue 8, pp 1-7.

3. Sanmeet Kaur and Maninder Singh, A Proactive Technique for Automatic Detection and Signature Generation for Zero Day HTTP Attacks on Web Server of an Educational Institute, Computer Fraud & Security, Elsevier, July 2014 [Accepted].

4. Sanmeet Kaur and Maninder Singh, Machine Learning Based Intrusion Detection for HTTP attacks using Bayesian Classifiers, International Arab Journal of Information Technology [Under Review].